Imperial College of Science, Technology and Medicine
Faculty of Engineering
Department of Computing

Identifying Safety-Critical Attacks Targeting Cyber-Physical Systems: a Systems Theoretic Approach

Luca Maria Castiglione

Submitted in part fulfilment of the requirements for the degree of
Doctor of Philosophy in Computing of Imperial College of Science, Technology and Medicine.
January 2024

‘Bisogna saper scegliere il tempo,
non arrivarci per contrarietà.’

Francesco Guccini

‘Science is magic that works.’

Kurt Vonnegut

Copyright Declaration

The copyright of this thesis rests with the author. Unless otherwise indicated, its contents are licensed under a Creative Commons Attribution-Non Commercial 4.0 International Licence (CC BY-NC). Under this licence, you may copy and redistribute the material in any medium or format. You may also create and distribute modified versions of the work. This is on the condition that: you credit the author and do not use it, or any derivative works, for a commercial purpose. When reusing or sharing this work, ensure you make the licence terms clear to others by naming the licence and linking to the licence text. Where a work has been adapted, you should indicate that the work has been changed and describe those changes. Please seek permission from the copyright holder for uses of this work that are not included in this licence or permitted under UK Copyright Law.

Statement of Originality

I hereby declare that this thesis and the research presented in it are my own work and that all information derived from the work of others is properly acknowledged.

Models employed to describe the behaviour of application components of the power system in the use case *"Security aware hazard analysis for smart grids"* were reviewed in collaboration with Dr. Pudong Ge and Dr. Fei Teng from the Control and Power (CAP) Research Group, Department of Electrical and Electronic Engineering, Imperial College London.

Models employed to describe the behaviour of application components in the representative flight management system discussed in the use case *"Analysing the impact of attacks on the safety of flight management systems"* were developed in collaboration with Philipp Stassen from Airbus.

Luca Maria Castiglione

Abstract

Over the last decades, society has witnessed a sharp increase in the use of complex and interconnected computer systems to monitor and assist in several aspects of everyday life. As operators of safety-critical systems deploy network-enabled devices aiming to enhance connectivity and streamline remote operations, the attack surface of these systems has increased. Whilst more attacks are becoming possible, only some of them will impact safety. Identifying such critical attacks is our priority; unfortunately, the complexity of modern cyber-physical systems (CPSs) renders this task challenging. Sophisticated attacks often rely on the effect of apparently legitimate commands, which can trigger cascading effects within the CPS itself, rendering it vulnerable to further attacks and causing harm. To help prevent these scenarios, tools and methodologies need to be developed that support integrated safety and security analysis in the context of CPS also considering their behaviours and internal dynamics.

We present Cassandra, a novel methodology to identify safety-critical threat scenarios and reason about their risk and applicable security measures in specific deployment contexts. Unlike other methodologies, Cassandra leverages existing relations between high-level threats and the system architecture to identify safety-critical attack paths. The qualitative and quantitative analysis of the paths thus found allows us to estimate the risk associated with safety-critical attacks, identify applicable security controls, and evaluate their effectiveness. Cassandra offers an integrated set of tools that enable the automated derivation of safety-critical sequences of threats and their respective attack paths. This provides an important step towards making integrated safety and security analyses less subjective, more reproducible and thus more suitable for applications in safety-critical contexts.

We have applied Cassandra to analyse the safe operation of safety-critical systems in three distinct use cases, including railway traffic control, power grid, and avionics. The scenarios analysed progressively increase in complexity and mirroring of real-world conditions.

Acknowledgements

Approaching the finish line means finally seeing the moment you craved the most when the path was the roughest. During this journey, there were plenty of these moments, especially with a pandemic that struck out of nowhere - and then just moved on. Nevertheless, getting through these tough moments was an unexpected learning experience. I consider myself extremely lucky, not only for this opportunity but, most importantly, for the extraordinary individuals around me who have consistently eased the *sting of difficulties*.

I am deeply thankful to my supervisor, Prof. Emil C. Lupu, for his invaluable advice and unwavering support throughout this PhD. His heartfelt knowledge of the subject and his exceptional patience and availability, even at the busiest times, profoundly impacted my research experience and encouraged me throughout my studies. The freedom he granted in allowing me to shape my research independently and, at the same time, his thorough and insightful guidance played a crucial role in my professional and personal growth.

I am extremely grateful to the EPSRC Centre for Doctoral Training in High-Performance Embedded and Distributed Systems (grant EP/L016796/1) and to the European Union for supporting my research.

My gratitude goes to Prof. Simon Pietro Romano and Prof. Francesco Franco for their encouragement and moral support during these years. To Erisa and Luis, who welcomed me when I first joined the Resilient Information Systems Security Group (RISS) and provided (and are still providing) me with precious insights, guidance and help as I was taking my first steps at Imperial. Special thanks are also ought to Amani El-Kholy, Luna Targett, Jamie Perrins and Bridget Gundry for their invaluable and continued support.

Over these years, I was lucky to collaborate with brilliant researchers: Zhongyuan Hau (Aaron), Cora Perner, Daniel Pereira, Kenneth T. Co, Jukka Soikkeli, Philipp Stassen, Gustavo de Carvalho Bertoli, Pudong Ge, Fei Teng. To all of them, I am deeply thankful.

Special thanks go to my great colleagues and friends Georgios and Conor, whom I met on my first day at Imperial and who supported me through these years. To Aaron, Jukka, Stefan, Andrea, Ivan, Javi, Claudio[2] and Kenny, for our drinks, coffee breaks and stimulating conversations.

Grazie a chi c'è sempre stato. Agli amici del liceo, Luca, Bea, Riccardo, Michele, agli amici incontrati più avanti, Luca, Marina, Nonno, agli amici dell'università e agli amici e ai colleghi che mi hanno dato tanto e mi hanno spinto a migliorare. Grazie agli amici sparsi nel mondo, a Francesco e Roberta che, seppur geograficamente lontani, sono sempre vicini ogni giorno.

Sono immensamente grato ai miei genitori che non hanno mai rifiutato nemmeno una telefonata - anche quando si erano appena seduti a tavola - ma soprattutto per il grande amore per la cultura, per la conoscenza e per l'umiltà che hanno sempre messo al primo posto nelle loro professioni e nelle loro vite. Grazie all'impagabile sostegno di Fernanda, che mi ha accompagnato con la sua ingenuità, praticità e spontaneità. A Toto che mi insegna quotidianamente ad alleggerire anche le situazioni più buie e che non smette di prendermi a modello. La sua immensa passione per la scrittura è stata di stimolo e di ispirazione anche nello scrivere questi *acknowledgements*. *"Un fratello maggiore non ha sempre ragione, solo qualche anno in più"*.

Sobre todo, estoy profundamente agradecido a mi *funderwol* Ana por su amor y apoyo constante, y por haberme mantenido cuerdo (más o menos) durante estos años. Gracias, Ana, por ser mi correctora, traductora, narradora preferita, alarma de todos los dias, y maestra cafetera. Sin vos, estos últimos años habrían sido inmensamente más difíciles. *Me haces bien.*

Contents

Copyright iii

Statement of Originality v

Abstract v

Acknowledgements ix

1 Introduction 1

1.1 Motivation and Objectives 3

1.2 Contributions 6

1.3 Dissemination 9

1.4 Organisation of the thesis 10

2 Background 11

2.1 Definitions 11

2.2 Run-through example: a Communication-Based Train Control system 14

2.3 Safety analysis 15

2.4 Threat models 19

2.5 Integrated safety & security analysis of cyber-physical systems 22

2.5.1 Approaches based on systems engineering 25

2.5.2 Methods based on trees and graphs 31

2.5.3 Expressing systems dynamics 34

2.5.4 Discussion 38

2.6 System Theoretic Accident Models and Processes (STAMP) 42

2.7 System Theoretic Process Analysis (STPA) 46
2.8 Summary & Conclusions 50

3 Cassandra: A Novel Methodology for Safety & Security Analysis 52

3.1 Overview of Cassandra 54
3.2 An integrated multi-layer approach 56
3.2.1 The functional model at the centre 59
3.2.2 Model of architecture 61
3.3 Structure of the methodology 61
3.4 Practical Examples of Applications 67
3.5 Summary & Conclusions 70

4 Cascading Effects 72

4.1 Safe behavioural model 73
4.1.1 Formalisation 74
4.1.2 Approaches to modelling cascading effects 75
4.1.3 Verification 78
4.2 Full-Knowledge approach 78
4.3 Partial-Knowledge approach 100
4.4 Summary & Conclusions 115

5 Automatic Identification of Threat Scenarios 117

5.1 Process overview 118
5.2 Threat Model 120
5.2.1 Model of adversarial actions 120
5.3 Discovery of Threat Scenarios 123
5.3.1 Formalisation 124
5.3.2 Attack enumeration 125
5.3.3 Threat scenarios tree 127
5.3.4 Implementation 129

5.4 Application to the communication-based train control scenario 130
5.4.1 Synthesis of adversary behaviour 131
5.4.2 Enumeration of threat scenarios 132
5.5 Summary & Conclusions 136

6 Threat Scenarios Feasibility and Risk Analysis 137

6.1 Threat model 138
6.1.1 From hazards to privileges: derivation of safety-critical vulnerabilities . . 140
6.1.2 Threats-Privileges mapping 142
6.2 Safety-critical attack graphs 143
6.2.1 Graph generation 147
6.2.2 AGTools library 148
6.2.3 Extraction of directed acyclic subgraph 149
6.3 Selection of security controls 150
6.3.1 Minimal Safety Set 151
6.3.2 Extraction of paths leading to Minimal Safety Set 154
6.3.3 Example of application 157
6.4 Likelihood of threat scenarios 161
6.4.1 Extending AG representation 162
6.4.2 BAG Extraction 166
6.4.3 Inference on the BAG 170
6.4.4 Example of application 171
6.5 Summary & Conclusions 171

7 Case Study: Communication-Based Train Control 173

7.1 Preliminaries 174
7.1.1 Safety Analysis 175
7.1.2 Behavioural Model 178
7.1.3 Attacker synthesis 186
7.2 Discovery & Enumeration of threat scenarios 188

7.3 Analysis of safety-critical attack paths . . . 191
7.3.1 Architecture . . . 193
7.3.2 Minimal safety set . . . 195
7.3.3 Graph path analysis . . . 195
7.4 Discussion . . . 200
7.5 Summary & Conclusions . . . 201

8 Case Study: Power Grid 202
8.1 Testbed architecture and safety model . . . 203
8.2 Analysis of the microgrid section . . . 207
8.2.1 Safe behavioural model . . . 210
8.2.2 Attacker synthesis . . . 213
8.3 Discovery & Enumeration of threat scenarios . . . 215
8.4 Analysis of safety-critical attack paths . . . 216
8.4.1 Safety-critical attack paths . . . 217
8.5 Discussion . . . 220
8.6 Summary & Conclusions . . . 222

9 Case Study: Flight Management System 223
9.1 Preliminaries . . . 224
9.1.1 Safety Model . . . 226
9.1.2 Definition of the scope of the analysis . . . 227
9.1.3 Derivation of control structure . . . 228
9.1.4 Identification of Unsafe Control Actions . . . 229
9.1.5 Identification of Threat Scenarios (STPA-Sec) . . . 230
9.1.6 Safe behavioural model . . . 231
9.1.7 Threat Model . . . 234
9.1.8 Formal verification . . . 235
9.2 Discovery & Enumeration of threat scenarios . . . 235
9.3 Analysis of safety-critical attack paths . . . 236

9.3.1 Generation of the attack graph . . . 238
9.3.2 Likelihood of Threat Scenarios . . . 242
9.4 Discussion . . . 244
9.5 Summary & Conclusions . . . 245

10 Conclusion **247**
10.1 Summary of thesis achievements . . . 249
10.2 Applications . . . 252
10.3 Limitations & future work . . . 254

A Details on safety-critical paths (CBTC) **257**

Bibliography **267**

List of Tables

2.1 Overview of related works 23

2.2 Overview of related works 39

2.3 Table of Hazards (refined) for the CBTC system. 43

2.4 Safety Constraints - Communication Based Train Control 43

2.5 Context Table of Zone Controller (ZC) and Train Controller (TC). r is the value for which CA is unsafe, v is the speed limit. 48

2.6 Table of Unsafe Control Actions - CBTC 48

3.1 Main formalism adopted in Cassandra 56

3.2 Processes of Cassandra involved in the analysis of safety-critical scenarios. . . . 70

4.1 Verification UCA - Propagation model 114

5.1 Resulting attack library 131

5.2 Attack traces violating safety specifications. Size of trace set (TS). Size of reduced trace set (RTS). Verification time (TE). 134

6.1 Minimal safety sets CBTC system 159

7.1 Table of Hazards for the CBTC system. 175

7.2 Context Table of Zone Controller (ZC) and Train Controller (TC). r is the value for which CA is unsafe, v is the speed limit. 177

7.3 Resulting Attack Library 187

7.4 Attack traces violating safety specifications. Size of trace set (TS). Size of reduced trace set (RTS). Verification time (TE). 188

7.5 Privileges on train and ground networks components 194

7.6 Minimal safety sets CBTC system 195

8.1 Table of accidents - Electric Power Grid . . . 205

8.2 Hazards leading to accidents A_1, A_2 (Electric Power Grid). . . . 206

8.3 Context table (CT) of the AC MG Controller. PE: Provided Early. PL: Provided Late. . . . 210

8.4 Table of Unsafe Control Actions MicroGrid . . . 210

8.5 Threats MG . . . 214

8.6 Threat scenarios found. . . . 215

8.7 Privileges AC MG architecture . . . 218

9.1 Table of Hazards for Flight Management System . . . 227

9.2 Control Actions and Feedback in Figure 9.2. . . . 229

9.3 Threat scenarios found. . . . 235

9.4 Threats and privileges in the use case. Spoofing and Tampering threats affect the integrity whereas DoS affects availability. NA: Network access. Exec: Code execution. . . . 237

9.5 Marginal Probabilities . . . 243

A.1 Structure of the attack graph . . . 258

List of Figures

1.1 Safety-critical threat scenarios and attack paths 6

2.1 Communication-Based Train Control System (Overview) 14

2.2 Fault tree - Example with only AND and OR gates 17

2.3 The cyber-kill chain 20

2.4 Safe Control Structure of Communication Based Train Control system. 45

2.5 STPA-Sec Overview 47

3.1 Cassandra - Main stages 54

3.2 Representations of the cyber-physical system considered in Cassandra. 57

3.3 Integrated view of the cyber-physical system. Arrows show our mapping across models. 58

3.4 Integration of views of cyber-physical systems 60

3.5 High Level Overview of the Structure of Cassandra. 62

3.6 Derivation of the model of adversarial actions. 65

3.7 Process of adversarial instrumentation 66

3.8 Processes involved in the methodology 68

4.1 Full-knowledge mapping 81

4.2 Control action (Feedback) - Full Knowledge (Explicit) 85

4.3 Control action (Feedback) - Plain Mapping Case 2 86

4.4 Merging control actions (feedback) when component when $a, c \in C'$ and $b \notin C'$. 89

4.5 Merging control actions (feedback) when component when $a, c \in C'$ and $b \notin C'$. 90

4.6 Zone controller 96

4.7 Train controller . . . 97
4.8 Physical process . . . 97
4.9 Communication Channel . . . 98
4.10 Observer 1 . . . 98
4.11 Partial-Knowledge approach mappings . . . 102
4.12 Zone controller - Propagation model . . . 110
4.13 Train controller - Propagation model . . . 111
4.14 Location unit - Propagation model . . . 111
4.15 Radio and GNSS Components - Propagation model . . . 112
4.16 Speed controller (actuator) - Propagation model . . . 113
4.17 control actions Model - Propagation model . . . 113
4.18 Control actions model - Propagation model . . . 114
4.19 Non reachability of TCTL queries under non adversarial conditions . . . 115

5.1 Cassandra - Overview . . . 118
5.2 Overview of the proposed methodology. . . . 119
5.3 Derivation of the model of adversarial actions. . . . 122
5.4 Process of adversarial instrumentation . . . 123
5.5 Example of threat scenarios tree. . . . 128
5.6 Example of threat scenarios tree - synthetic representation. . . . 128
5.7 Implementation of attack enumeration procedure. . . . 129
5.8 MAA for the CBTC use case. . . . 132
5.9 Threat scenarios tree (H_1) . . . 135

6.1 Cassandra - Overview. . . . 138
6.2 Accident-Threat-Vulnerability model . . . 139
6.3 Example of logical attack graph . . . 145
6.4 *Attack path* and *graph path* compared on the same attack graph. . . . 146
6.5 Derivation of Attack Graph . . . 148
6.6 UML Diagram AGTools . . . 149

6.7 Hazards in functions of privileges. . . . 152

6.8 Deployment architecture of the CBTC system 158

6.9 Attack graph (CBTC) 160

6.10 Adversarial actions (orange) and threat scenarios (red) on threat scenarios tree (6.10a) and attack graph (6.10b). . . . 165

6.11 Transformation from directed acyclic graph to Bayesian attack graph. . . . 167

6.12 Overview of the proposed methodology. . . . 170

7.1 Safe Control Structure of Communication Based Train Control system. . . . 176

7.2 Zone Controller 181

7.3 Train Controller 182

7.4 Location Unit 182

7.5 Physical Process 183

7.6 Communication Channel 183

7.7 Observer 1 184

7.8 Observer 2 184

7.9 Model of Adversarial Actions (CBTC) 187

7.10 Threat scenarios tree (A_1) 192

7.11 Communication-Based Train Control System (Overview) 193

7.12 Attack graph (CBTC) 197

7.13 Attack graph (CBTC) - Common nodes 199

8.1 Electric Power and Intelligent Control (Overview) 204

8.2 Power dependencies of the main loads in the EPIC testbed. . . . 204

8.3 STAMP model of EPIC 207

8.4 Safe Control Structure Microgrid 208

8.5 Behaviour of AC MG Controller 211

8.6 Behaviour of channel in the AC MG 212

8.7 Behaviour of monitor for AC MG 212

8.8 Behaviour of attacker 214

8.9 Threat scenarios tree . . . 216
8.10 Network topology of micro-grid. . . . 217
8.11 Attack graph to root privileges in micro-grid controller . . . 219
8.12 Attack graph derived via manual analysis STPA-Sec . . . 221

9.1 Aircraft Computer Network . . . 225
9.2 Safe Control Structure of Avionics - Highlighted elements are analysed in the next Section using Cassandra. . . . 228
9.3 Flight Control System (TA) . . . 232
9.4 Position Signal (TA) . . . 232
9.5 Navigation System (TA) . . . 233
9.6 Threat scenarios tree (UCA_1) . . . 236
9.7 Attack Graph - (Scenario 1). . . . 239
9.8 Attack Graph - (Scenario 2). . . . 241
9.9 Attack Graph - (Configuration 1). . . . 242

10.1 Integration of Cassandra in the V development process . . . 252

A.1 Colours encode different subsections of the attack graph . . . 259
A.2 Safety-critical attack paths - Section 1 (grey) . . . 260
A.3 Safety-critical attack paths - Section 2 (blue) . . . 261
A.4 Safety-critical attack paths - Section 3 (red) . . . 262
A.5 Safety-critical attack paths - Section 4 (yellow) . . . 263
A.6 Safety-critical attack paths - Section 5 (green) . . . 264
A.7 Safety-critical attack paths - Section 6 (light blue) . . . 265
A.8 Safety-critical attack paths - Section 7 (purple) . . . 266

Chapter 1

Introduction

Over the last few decades, our society has witnessed a sharp increase in the use of complex and interconnected computer systems to monitor, assist and regulate several aspects of everyday life. Digital communication has been integrated with cyber-physical systems (CPS) to enhance connectivity and streamline operations while minimising costs across many sectors, including industrial control systems (ICS) for manufacturing processes, supply chains, and critical national infrastructures (CNI) such as ground and maritime transportation, aviation, power and water supply, and many others.

As operators of safety-critical CPS deploy network-enabled Operational Technology (OT) devices to increase their connectivity and more IoT devices are interfaced with the OT, the attack surface of these systems also increases, allowing for a consistently growing number of attacks of different varieties [Joh11]. In February 2021, an individual gained unauthorised access to the computer network managing Florida's water supply and tried to abuse the treatment software to inject dangerous levels of lye in the water [BBC21]. This incident, along with reports from similar situations such as [Slo18] [ESE22] [Ros22], serves as a reminder that attacks against cyber-physical systems have the potential to impact the safety of the controlled processes, and can result in monetary losses, significant physical damages to infrastructure, and loss of life.

Additional concerns arise from the growing number of Advanced Persistent Threat (APT) groups that have recently been observed targeting critical sectors with OT-specific malware

such as BlackEnergy [Mit22], Crashoveride [Inc22], and Pipedream [Dra22].

Whilst more attacks are becoming possible, only some of them will impact the safety of the system. For example, losing the availability of non-critical components can reduce the quality of the provided service but not threaten safety. Similarly, a breach of confidentiality can result in a data loss, which, alone, is not generally a safety-critical event. There is no doubt that identifying the attacks that lead to safety violations is a priority. However, the complexity of modern CPSs renders this task extremely difficult. Sophisticated attacks often rely on the effect of apparently legitimate commands that, interleaved with system behaviour, trigger cascading effects within the CPS with the aim of rendering it vulnerable to further attacks and eventually causing losses. For example, in [SS11], the adversary relied on a vulnerability affecting the alternate mode of functioning of an unmanned aerial vehicle (UAV) to hijack it. By changing the perception of the environment around the aircraft (communication jamming), they were able to trigger a legitimate transition of operating mode, where a spoofing attack would become successful. To prevent these scenarios, tools and methodologies need to be developed that support complex integrated analyses of safety and security in the context of the behaviour of the system.

We propose Cassandra, a novel methodology to identify safety-critical threat scenarios and reason about their risk and applicable security measures given a specific deployment context. In Greek mythology, Cassandra is a Trojan priestess of Apollo whose tragic prophecies were truthful yet disregarded by the most [Ave62]. Likewise, our proposed methodology *foresees* potentially *tragic* (i.e., safety-critical) scenarios resulting from adversaries targeting the CPS, whereas authors such as [WT99] have expressed concern that users, operators, and engineers of CPS tend to sacrifice security to achieve better usability. Thus exposing the system to avoidable attacks despite recommendations.

Unlike traditional methodologies, Cassandra leverages existing relations between high-level threats and the system architecture to identify safety-critical attack paths, i.e., sequences of vulnerabilities that the adversary can exploit to cause harm. The qualitative and quantitative analysis of the paths found allows us to estimate the risk associated with safety-critical attacks, identify applicable security controls, and evaluate their effectiveness.

Furthermore, Cassandra offers an integrated set of tools that enable the automated derivation of safety-critical sequences of threats and their respective attack paths. This provides an important step towards making integrated safety and security analyses less subjective, more reproducible and thus more suitable for applications in safety-critical contexts. We have applied Cassandra to analyse the safe operations of safety-critical systems in three distinct scenarios: a communication-based train control system, an electric power grid, and a flight management system. Through the study of these use cases, we prove the effectiveness of our proposed methodology to deal with realistic scenarios progressively increasing in complexity and fidelity with respect to real-world conditions.

1.1 Motivation and Objectives

The security and resilience of cyber-physical systems are of paramount importance for national security and for the security and safety of its citizens. The increase of communication and computing elements combined with a growing number of APT groups targeting critical sectors renders these systems vulnerable to frequent and often sophisticated cyber-attacks.

The overarching objective of our work is the identification and assessment of threats to system-level safety properties. Our approach starts with the investigation of the following key research questions

RQ-1 Can we systematically identify the threats leading to violations of the system-level safety properties?

RQ-2 Can we automatically enumerate complex sequences of threats leading to safety violations under specific circumstances or (system) behaviours?

RQ-3 Can we uncover the attack paths that allow such safety-critical threats and sequences of threats in a specific deployment context? Or, which are the weaknesses and the vulnerabilities that enable safety-critical attack paths in a specific deployment context?

RQ-4 Which vulnerabilities could be remediated first to mitigate these safety-critical attack paths in a specific deployment context?

RQ-5 How do we quantify the risk associated with safety-critical attack paths in a specific deployment context?

Finding an answer to research questions $RQ-1$ requires modelling threats on a safety model of the system. Since our effort focuses on the identification of attacks provoking harm, our threat model primarily considers threats targeting integrity and availability; these are the threats generally considered of concern to safety [Int96]. Two are the difficulties behind $RQ-1$. First, unlike in traditional software systems, the unauthorised execution of legitimate commands can also provoke harm in cyber-physical systems, depending on the current state of the process. Second, adversaries can leverage cascading effects that are inherent in system behaviours. In other words, the exploitation of a threat, while not causing immediate harm, can be aimed at rendering the system vulnerable to other attacks and provoking harm through cascading effects. Thus, with $RQ-2$, we investigate the possibility of automatically identifying and successively enumerating safety-critical attacks. The main challenge behind $RQ-2$ lies in relating the safety model of the system, where hazards and threats are defined, with a behavioural model, which enables automatic inference. We elaborate on and answer $RQ-1$ and $RQ-2$ towards the end of Chapter 2 (Section 2.7), and in Chapters 4 and 5.

Not all safety-critical threats are feasible, and, as a consequence, not all the attacks found are viable. The investigation of $RQ-3$ aims to analyse the viability of the threat scenarios found given a specific deployment context. We leverage the integration between the safety model of the system and its architecture to identify attack packs that enable safety-critical threat scenarios.

The qualitative analysis of attack paths leading to safety-critical threat scenarios allows the identification of vulnerabilities and weaknesses that should be resolved to prevent harm, thus answering $RQ-4$.

While $RQ-3$ and $RQ-4$ might look similar at first glance, the main difference between the two

is in their intended objective. In $RQ-3$, we investigate vulnerabilities and weaknesses leading to all safety-critical information with the objective of gaining insights into the structure of the attack, such as pivotal points, critical hosts, etc. On the other hand, in $RQ-4$ we analyse attack paths leading towards a subset of privileges whose protection we proved to guarantee safety. The objective of $RQ-4$ is to prioritise mitigations aiming to prevent the reachability of safety-critical attacks.

While studying $RQ-4$ we realised that a qualitative answer might not always be feasible to find due to the size of the network and the resulting attack graph. Thus, in $RQ-5$ we use a Bayesian approach to quantify the risk associated with a specific threat scenario. We analyse and answer $RQ-3$, $RQ-4$, and $RQ-5$ in Chapter 6.

Two additional concerns were raised with respect to the usability of our approach in real-world use scenarios. These are expressed with the following two questions.

> **RQ-**6 Is the approach proposed by Cassandra applicable to real-world cyber-physical systems?
>
> **RQ-**7 Can the outcome of Cassandra be used while building assurance cases for safety-critical systems?

$RQ-6$ relates to the applicability of our approach to real-world systems, both in terms of input data required and in terms of computational complexity.

On the other hand, $RQ-7$ aims to investigate the applicability to produce assurance cases often used in safety-critical industries. Finding an answer to this question was behind the choice of grounding Cassandra in System Theoretic Process Analysis (STPA) and its extension for security STPA-Sec. In fact, STPA-Sec is an acceptable means of compliance in the aviation industry [EUR18]. The case studies that we present in Chapters 7, 8, and 9 were motivated by $RQ-6$ and $RQ-7$. Through the proposed analyses we empirically answer the two research questions.

In summary, the objective of our work is threefold. We identify and enumerate safety-critical

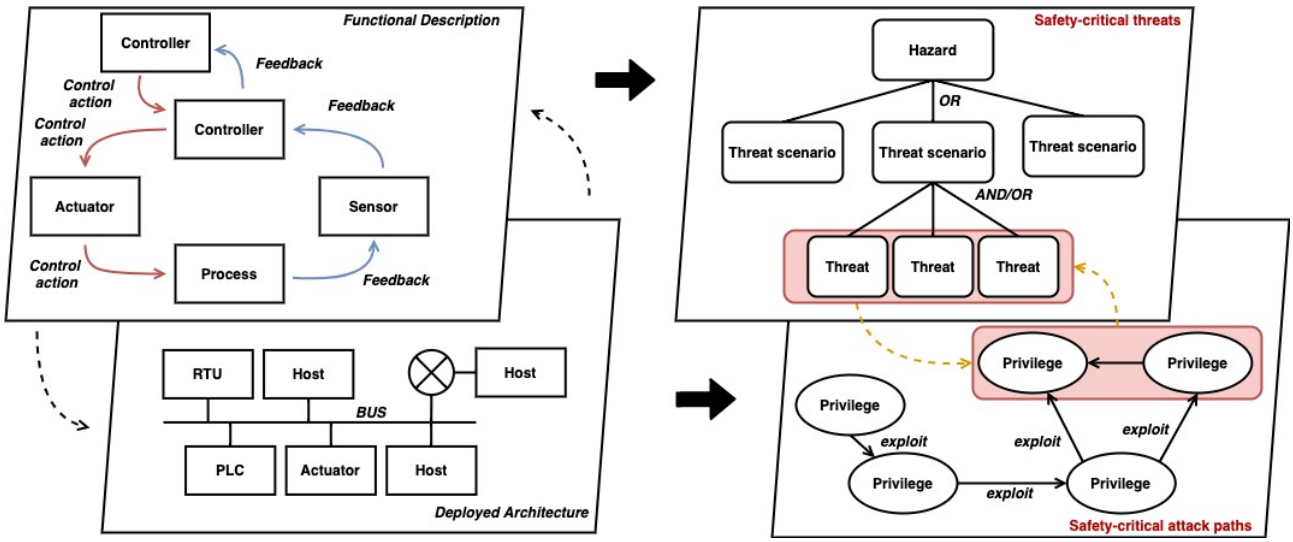


Figure 1.1: Safety-critical threat scenarios and attack paths

threat scenarios at the functional level. Thus, we leverage the integration with system architecture to uncover safety-critical attack paths (Figure 1.1). Lastly, we analyse qualitatively and quantitatively found paths to verify the viability of found paths, identify possible mitigations, and evaluate their associated risk.

1.2 Contributions

The research underpinning our proposed methodology advances systems theoretic approaches to integrated safety and security analysis through several original contributions.

We combine System Theoretic Process Analysis for Security (STPA-Sec) with threat modelling and model-checking in an entirely novel way. By analysing the combined models, we are able to enumerate complex attacks leading to system-level losses. These include multistage attacks that rely on the impact of malicious actions interleaved with system behaviour and on cascading effects to provoke harm, which had not been achieved in the related works in a systematic and comprehensive way. The first steps towards the automatic identification of high-level threat scenarios are the formalisation of the underlying safety model, in our case Systems Theoretic Accident Model and Processes (STAMP), and the systematic identification of threats with potential implications on system-level safety properties. In particular, we introduce a

formalisation of the *safe control structure* of the CPS, i.e., its control hierarchy, and use STRIDE to systematically model threats against this latter, thus answering $RQ-1$.

The application of model-checking to automatise the discovery of threat scenarios requires the definition of the following elements:

a) a model of the behaviour of the system components, which also maps to the underlying safety model, in our case Systems Theoretic Accident Model and Processes (STAMP).

b) a behavioural model, which represents the behaviour of an adversary.

c) a set of mappings to translate STPA safety properties in formal statements, such that they can be verified by the model-checker.

We define two approaches to modelling the behaviour of the CPS, namely, *partial-knowledge* and *full-knowledge*, to be used exclusively, depending on the amount of information available at design time and on the level of detail that needs to be captured. These approaches extend the state-of-the-art and are closely related to the safety model of the CPS (Chapter 4). The formal definition of these two approaches, which model the impact of adversarial actions on the CPS and, at the same time, also establish rigorous relations with STAMP and STPA is also a novel contribution of this thesis. We use the threats found following the application of STRIDE on the safe control structure of the CPS to build the behavioural representation of the adversary. The systematic derivation of the behavioural model of an attacker, starting from a systematic threat modelling activity on the safety model, is also an original contribution of our work. Lastly, we define the necessary mappings to verify STPA safety properties on the approaches of behavioural models that we have developed.

With Cassandra we also design a strategy based on the verification of reachability properties over multiple iterations to exhaustively enumerate threat scenarios, thus answering $RQ-2$ (Chapter 5).

An additional novel aspect of our work stems from the integration of the results obtained from the high-level analysis of the functional model with attack graphs. While the former

returns safety-critical sequences of threats against functional elements, the latter tells how these can be exploited at the architectural level. In particular, we introduce the *Accident-Threat-Vulnerability* model, which links objects from the STAMP domain to vulnerabilities and weaknesses in the architectural model (Chapter 6). This is a crucial contribution and allows us to relate high-level hazards to individual low-level exploitations.

Cassandra integrates with attack graphs to produce safety-critical attack paths, i.e., sequences of exploitations of vulnerabilities and weaknesses leading to safety-critical threats in the system architecture. Through the qualitative analysis of these paths, we can verify the feasibility of safety-critical attacks and evaluate the importance that individual vulnerabilities have in enabling these attacks, thus answering $RQ-3$.

We introduce the minimal safety set (MSS), i.e., the minimum set of privileges that should be defended to guarantee safety. The broad concept of finding minimal groups of objects (e.g., components) which must operate reliably to guarantee safety is often adopted in the literature related to safety. Nevertheless, the identification of such privileges, whose protection guarantees safety, is unprecedented to the best of our knowledge and crucial to later prioritise mitigations. The qualitative analysis of attack paths leading to privileges in the MSS, tells which vulnerabilities should be remediated first to guarantee safety, thus answering $RQ-4$.

As part of our contributions, we also extend the attack graph representation with a new set of facts, predicates, and rules. These allow us to express safety-critical threats and hazards on the attack graph, together with vulnerabilities and privileges. Our methodology integrates with Bayesian inference, which we use to quantify the risk of an adversary causing safety-critical attacks. To facilitate the integration, we also develop an algorithm to extract a Bayesian network starting from a logical attack graph. Through this application, we are able to answer $RQ-5$.

We address the concern expressed in $RQ-6$ by employing our approach to analyse the safe operations in *three* representative use cases at increasing levels of complexity and mirroring of real-world conditions. Of these scenarios, one features a Flight Management System (FMS) and was developed in close collaboration with Airbus. The work was presented as a full paper

at $42-nd$ edition of the AIAA/IEEE Digital Avionics Systems Conference (DASC) [CSP+23]. The presentation was well received by an audience largely composed of safety experts and the paper was awarded the *best paper in session award*. In doing so, we have also empirically answered $RQ-7$.

Overall, to the best of our knowledge, our work is the first to propose a methodology that analyses the impact of attacks on CPS emergent safety properties and, at the same time, also provides detailed and formal mappings between functional and behavioural models, relies on a robust safety and security models to enumerate all possible generic threats, and closely relates to system architecture to identify safety-critical attack paths.

1.3 Dissemination

The work presented in this thesis also contributed to the following publications:

- **Castiglione, Luca Maria**, and Emil C. Lupu. "Hazard Driven Threat Modelling for Cyber Physical Systems." *Proceedings of the 2020 Joint Workshop on CPS&IoT Security and Privacy*. 2020 [CL20].

- **Castiglione, Luca Maria**, Zhongyhuan Hau, Ge Pudong, Kenneth T. Co, Luis Muñoz-González, Fei Teng, and Emil C. Lupu. "HA-Grid: Security Aware Hazard Analysis for Smart Grids". *2022 IEEE International Conference on Communications, Control, and Computing Technologies for Smart Grids (SmartGridComm)*. IEEE, 2022 [CHC+22].

- **Castiglione, Luca Maria**, Philipp Stassen, Cora Perner, Daniel Pereira, Gustavo de Carvalho Bertoli, and Emil C Lupu. "Don't Panic! Analysing the Impact of Attacks on the Safety of Flight Management Systems" *2023 AIAA/IEEE 42nd Digital Avionics Systems Conference (DASC)*. IEEE, 2023. ***(Winner of the best paper award for the session Cybersecurity I)***. [CSP+23].

- **Castiglione, Luca Maria**, and Emil C. Lupu. "Which Attacks Lead to Hazards?

Combining Safety and Security Analysis for Cyber-Physical Systems". *IEEE Transactions on Dependable and Secure Computing*, IEEE, 2023 [CL23].

1.4 Organisation of the thesis

The rest of the thesis is organised as follows. In Chapter 2, we present relevant background information and elaborate on the state-of-the-art integrated safety and security analyses applied to safety-critical systems. Chapter 3 provides an overview of our proposed methodology and its application to three distinct use cases. In Chapter 4, we introduce our formal relationships between the safe control structure (functional model) of the cyber-physical system and its behaviour; we elaborate on the key stages of Cassandra in Chapters 5 and 6. Chapters 7, 8, and 9 are each dedicated to a different use case. In particular, Chapter 7 combines in a single analysis the paragraphs dedicated to our run-through example over the previous chapters. Chapters 8 and 9 showcase the application of our methodology on an electric power system and on a flight management system respectively. Finally, in Chapter 10 we discuss the thesis achievements and possible directions for future work.

Chapter 2

Background

The intersection of security and safety in cyber-physical systems has been addressed in literature from several different perspectives. In this chapter, we introduce the relevant scientific background in the field. We present an overview of the main approaches for integrated safety and security analysis and discuss their differences with our proposed methodology.

This chapter is organised as follows. We introduce the terminology adopted in this thesis in Section 2.1 and, in Section 2.2, we provide a high-level description of the use case that we adopt as a run-through example over the next chapters. In Sections 2.3 and 2.4, we produce an overview of relevant methodologies for safety analysis and threat modelling, respectively. In Section 2.5 we discuss the literature at the intersection of security and safety and provide an overview of approaches related to ours. Thus, in Sections 2.6 and 2.7, we provide a deep overview of System Theoretic Accident Models and Processes (STAMP) and System Theoretic Process Analysis (STPA) [Lev16], which are the basis of our proposed approach. Finally, Section 2.8 concludes this chapter.

2.1 Definitions

The terms *safety* and *security* are characterised by a rather polysemic nature. Similarly, multiple definitions of *accidents* and *hazards* can be found in the literature. In order to reduce

ambiguities and ensure a consistent use of the terminology throughout this thesis, this section provides definitions for the main terminology adopted.

Safety & Security We rely on the definition of *safety* provided in [PM99] as *freedom from unacceptable risk*, where the *risk* is defined as *a combination of the likelihood of harm and severity of that harm* [Sta07] [Hel12]. At the same time, the term *security* refers to the protection of *confidentiality*, *integrity*, and *availability* of the information within the cyber-physical system.

Our research is at the intersection of security and safety; we investigate how the loss of security attributes such as *confidentiality*, *integrity* and *availability* can impact safety properties. We have mentioned that, in safety-critical systems, the loss of safety is usually associated with the loss of integrity or availability of the information [Int96]. Loss of confidentiality can, in some instances, also lead to unacceptable risk, e.g., if the identity of an undercover police officer is revealed as a consequence of a data leak. However, while acknowledging that loss of confidentiality can also jeopardise safety, we narrow the focus of our analysis primarily on the integrity and availability attributes.

Accident & Hazard We use the following definitions of *accident* and *hazard* provided by [Lev16, SQR+19].

> ***Accident****: An undesired and unplanned event that results in a loss, including a loss of human life or human injury, property damage, environmental pollution, mission loss, financial loss, etc.*

> ***Hazard****: A system state or set of conditions that, together with a worst-case set of environmental conditions, will lead to an accident and its related losses.*

Safety Requirements We use the following definitions of *safety requirements* and *safety constraints* proposed in [How04].

> ***Safety Requirements*** *describe what the system must do in order to achieve functional goals.* ***Safety constraints*** *describe what the system must not do in order to maintain safe operation.*

One of the objectives of the methodology proposed in this thesis is to assist in the formulation of *security requirements* for cyber-physical systems. Security requirements prescribe what the system must do to protect the confidentiality, integrity and availability of the information in the system [SHB06].

Threat Scenario, Threat Scenarios Tree & Attack Paths We rely on the following definition of *threat* adapted from [SKB+08]

> *Any circumstance or event with the potential to adversely impact agency operations (including mission, functions, image, or reputation), agency assets, individuals, other organisations, or the Nation through an information system via unauthorised access, destruction, disclosure, modification of information, and/or denial of service.*

We define a *safety-critical threat scenario* as the sequence of threats leading to catastrophic accidents. Considering the position of this thesis at the intersection of security and safety, we employ the terms *threat scenario* and *safety-critical threat scenario* interchangeably and we explicitly mention when *threat scenarios* are not safety-critical. We define the *threat scenarios tree* as the tree that shows all the scenarios leading to the same safety violation (Section 5.3.3). On the other hand, we use *attack paths* to refer to the sequences of exploitations leading to a threat condition (Section 6.2).

While notions of *threat scenarios tree* and *attack paths* might sound akin to each other, they are not. In fact, they are defined at different levels of abstraction. The former shows the high-level threats that the adversary can use to cause a safety violation, whereas attack paths refer to the vulnerabilities and weaknesses that they exploit on the system architecture to realise threats.

2.2 Run-through example: a Communication-Based Train Control system

In this Section, we provide a high-level description of the use case, which we use as a run-through example throughout this thesis, aiming to demonstrate the individual applicability of the sections composing our proposed methodology.

A Communication-Based Train Control (CBTC) system is a traffic management system typical of wide railway infrastructures such as the European Railway Traffic Management System (ERTMS) [Uni]. This is an example of a safety-critical system, and it is also part of the Critical National Infrastructure (CNI) of several European countries. In our scenario (Figure

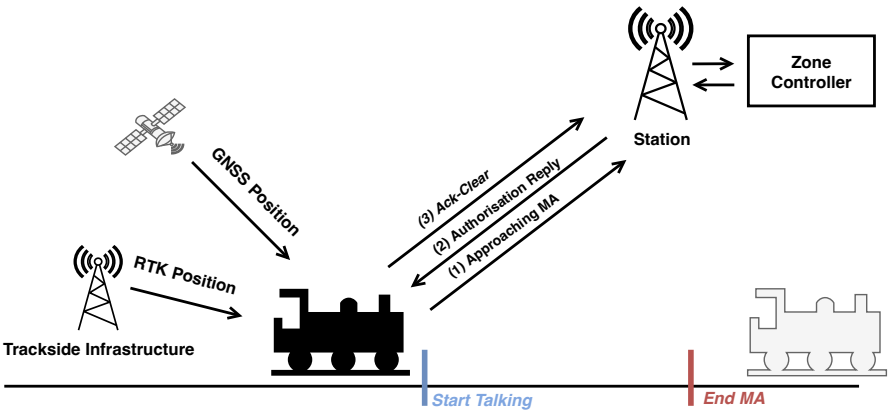


Figure 2.1: Communication-Based Train Control System (Overview)

2.1), N trains share access to the same railway infrastructure which is divided into M *zones*. A zone controller (ZC) ensures that trains access each zone in a mutually exclusive fashion. The zone controller *allows* or *declines* access requests from trains, depending on their current location and the signalling context (stations, junctions, other trains, etc.). Trains are fitted with an onboard Train Controller (TC) that constantly receives the current position from an onboard location unit (LU) and speed from the Speed Controller (SC). The train controller is responsible for sending *access* and *clear* requests to the zone controller and evaluates the replies. In our scenario, the LU combines data from a GNSS receiver with radio beacons to improve the accuracy of the estimated position [Wan04], in a configuration similar to [CFP+20]. A back-off strategy allows the LU to solely rely on GNSS if the radio signal is unavailable. The TC also

receives a reference speed limit from the ZC for the current zone and sets the target speed for SC. Train motion is controlled by the speed controller, which computes the acceleration profile and commands the brake and acceleration unit (AU) over a local bus. Positive values of the acceleration are interpreted as acceleration and negative ones as a brake. A safety mechanism halts the trains if the communication between the zone controller and one of the trains fails.

2.3 Safety analysis

As human beings, we have proven ourselves surprisingly good at designing systems that are safe [IAT22]. Such achievements have been made possible by a culture around safety, which many industries have been developing over the last half-century. In this sense, many techniques for safety analysis and risk assessments have been developed to improve and assist in the design of critical systems. In this section, we provide an overview of some of these methodologies. For an exhaustive collection of safety analysis techniques, we refer the reader to [E+15]. More recently, [BTB+19] proposes a survey which also elaborates on industry standards for assurance.

Failure Mode and Effects Analysis (FMEA) describes the systematic analysis of potential failure modes of the distinct components in a complex system. FMEA is a bottom-up process and is widely accepted across many industries, being recognised as a valid tool for the creation of assurance cases by industry standards from all sectors [IEC18]. During FMEA sessions, experts analyse all the potential failure modes of low level components of the system. The objective of the analysis is to identify and evaluate the effect(s) of functions, components, and, more in general, subsystems of the CPS with the aim of preventing accidents. Ericson et al. elaborates on three approaches for applying FMEA: on system *functions* (functional application), on its *components* (structural approach), or sequentially on both (hybrid approach) [E+15]. Failure Mode, Effects and Criticality Analysis (FMECA) extends FMEA by also considering the *criticality* of a failure mode as well as its rate of occurrence and detectability. In FMEA/FMECA, we also compute a risk priority number (RPN) for failure modes, which is defined as the product of the probability of occurrence (usually estimated on product history),

the severity of the failure, and its detectability rate. A number of additional extensions have been proposed to FMEA, some of them are summarised in [HYLS20].

The main limitation of FMEA stems from the fact that both the identification of failure modes of components and the analysis of the consequences are manual activities driven by expert knowledge. To this extent, the output of the analysis is heavily dependent on the expertise of the analyst, thus it is subjective and not reproducible. Moreover, FMEA looks at individual failures and does not provide the tools to systematically study the impact of combinations of failures.

The application of FMEA to hardware systems is generally more straightforward compared to its application on software-intensive systems [E+15]. The issue, which is generic and not specifically associated with FMEA, has to do with the inherent complexity of software systems whose behaviour can sometimes be unpredictable. The following work by Leveson elaborates on safety issues in software-intensive systems [Lev86].

Hazard and Operability Analysis (HAZOP) is an approach to the systematic identification and analysis of hazards and operational concerns at system-level [E+15]. In particular, the objective of HAZOP is to identify operability issues that may arise as a result of deviations from the original design intent and assess the respective hazards. Different parts of the facility (or system) are scrutinised against a list of *guide words*. The latter acts as a checklist to recognise deviations from specifications. The key to a successful HAZOP is an expert leader of the process and a multi-disciplinary team of analysts. In this sense, HAZOP shares the same limitations as FMEA concerning being manual and driven by expert knowledge.

Fault Tree Analysis (FTA) provides the tools for a systematic analysis of fault trees aiming to determine causes behind safety-critical events [E+15, CT15]. A fault tree is a logical model which can represent combinations of events leading to accidents, failures, and, more generally, undesired top-level events. The formalism has been expanded over the years with different types of logical *gates* to describe the propagation of events happening simultaneously, in order, or under specific conditions. FTA offers a bottom-up approach, and the type of analysis performed can be qualitative, e.g., what are the causes of a fault, as well as quantitative, e.g., what is the

probability of an undesired event happening. For a more detailed overview of fault trees and FTA, we refer the reader to [E+15, EL99].

The fault tree in Figure 2.2 shows the events leading to the undesired top-level event *Power cut to critical load* in a fictional and representative microgrid. The event can be the result of a failure of the circuit breaker or of a broader failure in the microgrid. The latter, for simplicity, is assumed to be caused by a failure of the battery together with not enough energy produced by the photovoltaic unit. Given the knowledge of failure rates of the circuit breaker, the battery,

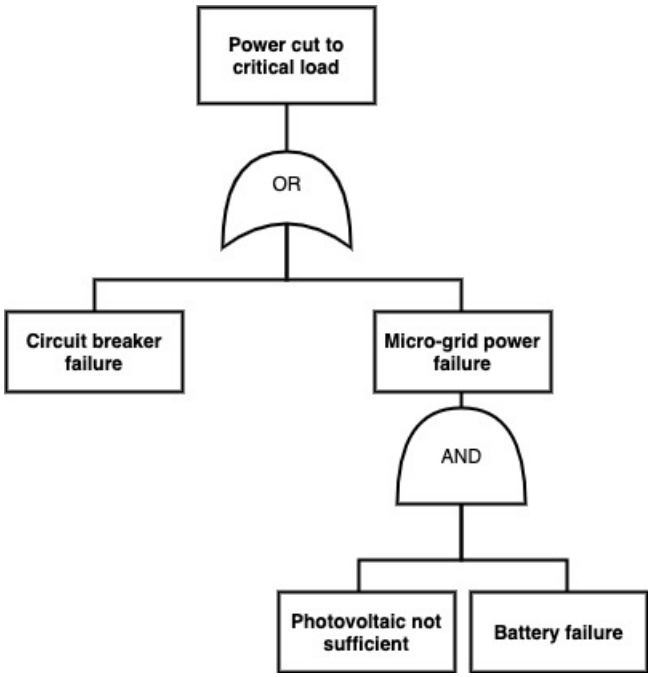


Figure 2.2: Fault tree - Example with only AND and OR gates

and the photovoltaic unit, quantitative FTA allows us to estimate the probability of a power cut to critical load.

The main limitation of FTA stems from the fact that the fault tree is generally constructed manually and the activity is once again driven by expert knowledge. Unlike FMEA, FTA looks at the impact that individual failures have within the whole system context.

The **Hazard Analysis and Risk Assessment (HARA)** is defined by ISO 26262 (Part 3) [iso18] and is specific to the automotive industry. Its objective is the identification of hazards associated with failures of electronics in modern vehicular architectures, their assessment, and the computation of the Automotive Safety Integrity Level (ASIL). Identified hazards and ASIL

are used for the formulation of safety requirements that should be met by the design to prevent damage to vehicle occupiers and road users. HARA is a systematic methodology that develops in *six* steps. In the first phase, we identify the *items* (e.g., components) that are objects of the analysis. We proceed with the *situation analysis hazard identification* where we identify the operational contexts of items and related hazards. ISO 26262 explicitly mentions FMEA and HAZOP as viable methods to identify hazards, which, at this stage, are defined at the system level. The aim of this phase is to capture the cases where the failure of a component would result in hazards. Hazards are then classified according to their *severity*, *probability of exposure*, and *controllability* and an ASIL is determined as a combination of these scores. Given hazards and related ASIL, we determine the safety *goals* (requirements) for the items. An additional step is used to adjust the analysis for vehicles that admit multiple versions, configurations, or modes of operation. Finally, HARA requires the results of the analysis to be manually verified for *compliance, consistency and completeness* [iso18]. Similarly to FMEA, the main limitation of HARA is also related to its manual and expert-driven nature. To this extent, the final step imposed by HARA aims to mitigate errors related to the manual nature of the process.

System Theoretic Accident Models and Process (STAMP) is a safety model grounded in systems theory [Lev16]. Unlike more traditional approaches such as HAZOP, FMEA/FMECA, and FTA, STAMP models safety as a control problem and sees accidents as the result of flaws in the interactions between system components. This peculiarity makes STAMP the optimal candidate model for our methodology, which aims to study the cascading effects of attacks within the CPS. Two are the main safety analysis methodologies developed on the STAMP model, **System Theoretic Process Analysis (STPA)** and **Causal Analysis based on STAMP (CAST)**. The former (STPA) is aimed at the identification of unsafe interactions and the generation of safety requirements. On the other hand, the latter is a forensic analysis methodology which aims to identify the root causes of accidents. Given the central role played by STAMP and STPA in the context of our approach, we provide a detailed overview of their structure and an example of application later in this chapter (Sections 2.6, 2.7).

The integration of safety analysis with formal verification is not uncommon and can provide relevant insights about the capability of system designs to comply with expected safety require-

ments. In particular, we will focus on the application of formal verification to STPA, which we discuss in more detail later in this chapter (Section 2.7).

2.4 Threat models

In this section, we provide an overview of *five* widely adopted methodologies for threat modelling, STRIDE, PASTA, OCTAVE, the Cyber Kill-chain, and the MITRE ATT&CK framework. We also discuss attack trees and attack graphs to represent attack paths. For a more comprehensive description of the threat modelling tools presented, we refer the reader to [SCO+18].

STRIDE is among the most adopted and widely accepted methodologies for threat modelling [SCO+18]. STRIDE is an acronym for the threats considered during the threat modelling activity, namely, spoofing, tampering, repudiation, information disclosure, denial of service, and elevation of privileges [Sho14a]. The threat modelling activity develops in two steps. During the first stage, the system is represented using a data flow diagram (DFD) and a trust boundary is defined. STRIDE models threats against elements of the DFD are then systematically identified during the second stage of the methodology. Currently, there are two versions of STRIDE, depending on how threats are derived during the second stage of the threat modelling activity. STRIDE-per-element iteratively considers STRIDE threats against all the elements of the information flow. A second approach, also known as STRIDE-per-interaction, focuses on the interactions in the system and, for each interaction, considers STRIDE threats to its source, its destination, and to the information flow. A comparative study shows that STRIDE-per-element is quicker, easier to perform by experts and more accurate than the other variant. On the other hand, it relies on expert knowledge, being mainly based on brainstorming [BH15].

The **Process for Attack Simulation and Threat Analysis (PASTA)** is a comprehensive threat modelling tool to quantify the risk of adversarial threats [UM15]. The process comprises *seven* stages defined at different levels. Potential objectives, scope of the analysis and threats are defined at high-level. Thus, during the stage of *vulnerability & weaknesses analysis*, high-level

threats are mapped to vulnerabilities (CVE) [MIT23b] and weaknesses (CWE) and scored using scoring systems such as CVSS [NIS]. Attack trees are then built with the aim of quantifying the risk with respect to the objectives identified at the beginning of the analysis and producing mitigation strategies. Overall, the method proposed by PASTA involves a number of actors from different strategic levels inside an organisation from engineers to governance. To this extent, each step of the methodology is supported by a different set of tools of different nature. Proposed mitigations can also affect different levels, from software development to higher-level business strategies [SCO+18].

The **Operationally Critical Threat, Asset, and Vulnerability Evaluation (OCTAVE)** is a high-level methodology for security risk assessment [ADSW03, SCO+18]. The approach develops in three phases; namely, identification of threats, identification of vulnerabilities, and development of a strategy. It is aimed at evaluating activities at an organisational level and assessing their related risk; we have included it as an example of a high-level threat modelling approach.

The **Cyber kill-chain** describes the stages of a cyber attack. The aim of the approach is to dissect complex attacks into distinct phases, allowing organisations to prevent, detect, and mitigate them [AL15]. The Cyber kill-chain composes of *seven* steps, namely *reconnaissance*, *weaponisation*, *delivery*, *exploitation*, *installation*, *command and control*, and, *action*. These are summarised in 2.3. By breaking down complex attacks into different stages, the Cyber

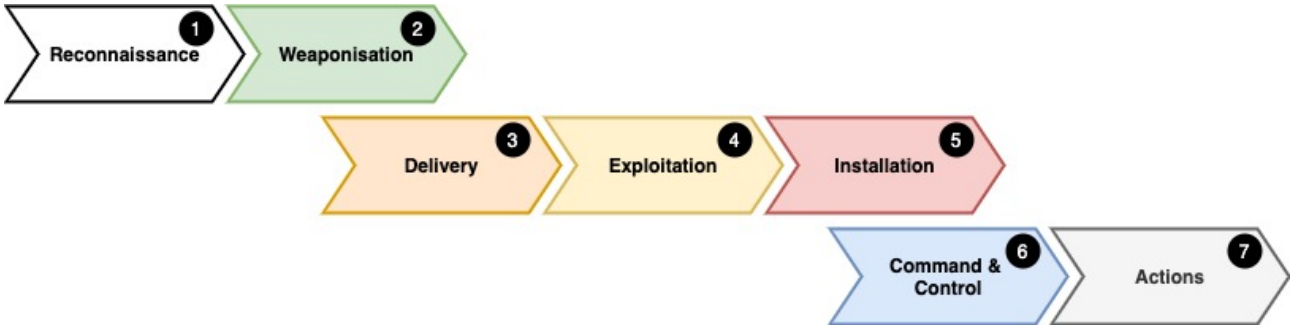


Figure 2.3: The cyber-kill chain

kill-chain aims to detect and disrupt attacks before they cause significant harm.

The **ATT&CK** framework developed by **MITRE** proposes a collection of tactics, techniques,

and sub-techniques derived from real-world observations of adversarial behaviours [ATT21]. The framework aims to provide a knowledge-base for the development of *more credible*, *advanced*, and *systematic* threat modelling tools. Currently, ATT&CK features three matrices; namely, enterprise, mobile, and industrial control system (ICS). In particular, the latter combines tactics, i.e., attack objectives, and techniques from real-world attacks targeting ICS. An additional feature is the link between ATT&CK tactics and the APT group(s) which have employed them. The absence of a recognised mapping between ATT&CK ICS and high-level threat models (e.g., STRIDE) hinders its integration with the analysis of attacks at a functional level. On the other hand, CAPEC patterns are increasingly being mapped to ATT&CK techniques, making it possible to combine them when modelling threats against architectural elements.

Attack description Attack trees (AT) and attack graphs are two modelling tools widely employed to describe attacks. Attack trees are directly adapted from fault trees with leaves representing basic security events, generally exploits. Formally, they are directed acyclic graphs whose visual representation resembles a tree. On the other hand, attack graphs represent relations between privileges and security events such as vulnerability exploits. Both formalisms are used to describe attacks. The main difference between the two is that attack trees are strictly acyclic and allow us to represent only one objective, the root of the tree. The two formalism and their differences are exhaustively described in [LDB20].

Many different variants of attack graphs can be found in the literature. These are mainly organised in two broad categories, *state based* and *exploit-dependency* attack graphs. Two exhaustive surveys on the different representations commonly adopted for describing attack graphs and their respective concerns are [LDB20, Zen22].

We will see how our approach adopts attack trees as a formalism to describe threat scenarios (Chapter 5). At the same time, we also adopt the two variations of *exploit-dependency* attack graphs to describe safety-critical attack paths (Chapter 6). In particular, we perform a qualitative analysis of logical attack graphs produced by MulVAL [OGA+05] and a quantitative

analysis through Bayesian attack graphs (BAG).

2.5 Integrated safety & security analysis of cyber-physical systems

In the previous section, we have introduced the methodologies traditionally employed to study safety in safety-critical systems as well as those employed to analyse the security of software-intensive systems.

In modern cyber-physical systems, where software elements and connectivity are integrated with the control loop, we observe several dependencies between safety and security [NTP07]. Conducting safety and security analysis *in parallel*, i.e., individually and (often) on two different models, carries significant limitations. In the first place, it hinders causality relationships between security threats, vulnerabilities, or weaknesses (depending on the underlying security model) and safety-critical events. In addition to this, conflicts can arise between safety requirements and security recommendations. A classic example is that safety guarantees traditionally rely on the availability of the information to hold, whereas security aims to restrict access to information. In particular, [PC10] identifies a few types of dependencies between safety requirements and security recommendations. Among these, the most relevant are *reinforcement*, i.e., security requirement increases safety, and *conflict* (or antagonism), i.e., security requirement violates a safety property.

In order to prevent accidents, and hence avoid harm, such dependencies must be considered within the scope of the safety analysis [KPCBH15]. In this sense, multidisciplinary and integrated approaches are required to identify threats that are key to preventing safety-critical attacks and to uncover vulnerabilities and weaknesses that enable them.

Several research works acknowledge the need for integrated approaches and have taken steps towards their realisation, although important gaps and challenges remain. These include improving current formalism and techniques to respectively better capture interactions between

Reference	Name	Formalism(s)	Safety model	Threat model	*Functional*	*Architectural*	*Behavioural*
[YP17]	STPA-Sec	STAMP	STPA	-	●	○	○
[NM15]	-	STAMP	CAST	Custom	●	○	○
[FMS+17]	STPA-SafeSec	STAMP Component diagram	STPA	Custom	●	●	○
[AR13, RA15]	SysML-Sec	Sys-ML Attack graph	-	-	●	●	●
[MSB+15a]	SAHARA	Component diagram	HARA	STRIDE	○	●	○
[BCEF18]	-	Custom	-	CVE	○	●	○
[SKG21]	-	Attack graph	-	CVE	○	●	○
[SMMY18, MSMY19]	-	STAMP	STPA	NIST SP-800	●	●	○
[DSCdMBH20]	-	STAMP	STPA	STRIDE	●	○	○
[KM21]	CyberSafety	STAMP	STPA	-	●	●	○
[HCS+21]	CVAnalyzer	FSM	*availability*	Custom	○	○	●
[MLS+21]	VERDICT (MBAAS)	AADL Attack graph	-	CAPEC	○	●	●
[MLS+21]	VERDICT (CRV)	AADL Attack graph	Custom	Custom	○	●	●
[PCB10a, PCB10b]	BDMP	Attack tree	-	-	○	●	●
[KNKS22]	AFMT	Attack tree	-	-	○	●	●
[LCCZ19]	-	Attack tree	-	Custom	○	●	●
[EVCC+22]	-	Closed control loops	-	Custom	●	●	○
[PGM21]	-	Goal based notation	Standard	Standard	●	○	○
[HTLC15]	-	AADL	-	Custom	○	●	●
[KBL19]	S-Cube	Custom	-	Custom	○	●	●
[HBCS19]	-	STAMP	STPA	Custom	●	○	●
[CL23] [CSP+23]	CASSANDRA*	STAMP Attack graph Network of timed automata	STPA	STRIDE CVE	●	●	●

(*) **Our work**

Table 2.1: Overview of related works

safety and security and ensure objectivity and reproducibility of results, as well as engineering aspects such as scalability and automation [SKN+21]. The following surveys present an overview of the main literature in the field and discuss similarities and differences in the approaches [RO13], [PCB13], [LŠČ18], [KPCBH15], [NPKS23], [PGM21]. International standards for the safety-critical industry are also mandating the need for integrated security and safety analysis across a variety of sectors from nuclear [PCQH13, PCQ15] to aviation [EUR18].

In particular, [KPCBH15] distinguishes between *generic* and *model-based* approaches to integrated safety and security analysis. The former are characterised by a macroscopic perspective and are further classified in *unification* or *integration*. Unification approaches aim to combine safety and security taxonomy in an integrated analysis, whereas integration approaches aim to intersect the result from safety and security analysis conducted in parallel. *Model-based* approaches, on the other hand, are classified depending on their stage of application as well as their input models, e.g., block diagrams, trees, graphs, etc. A similar classification is also adopted (and adapted) by [NPKS23].

We adopt the same type of classification while discussing approaches we believe are more relevant to our work further in this section. In particular, we distinguish between approaches based on systems engineering (Section 2.5.1), extensions of fault-trees and graphs (Section 2.5.2), and approaches that look more closely at CPS dynamics (Section 2.5.3). At the end of the section (Section 2.5.4), we discuss the main differences between state-of-the-art methodologies and our proposed approach.

In Table 2.1, we report an overview of the main approaches mentioned in this section. For each item, we have reported the formalisms employed, the underlying safety framework, and the threat model. The last three columns of the table highlight the type of analysis provided by the approach, including functional and architectural (deployment) aspects as well as their capability of dealing with system behaviour, i.e., performing dynamic analysis. We elaborate further on the main differences across the approaches at the end of the section (Table 2.2).

2.5.1 Approaches based on systems engineering

Systems Engineering (SE) involves a methodical approach to design and understand complex systems. A number of approaches in literature aim to perform integrated safety and security analyses on traditional SE models such as SysML [Wei11], Architecture Analysis & Design Language (AADL) [FGH06], or simpler block diagrams.

Security-Aware Hazard and Risk Analysis (SAHARA) combines STRIDE threat modelling with HARA safety analysis, to perform integrated safety and security analysis [MSB+15b]. The analysis is carried out manually on the high-level architecture of the system and is mainly driven by expert knowledge. The extent to which cascading effects are discovered depends on the knowledge of the system and the inference does not appear to be systematic. Security levels (SecLs) for individual threats against architectural elements are calculated based on the resources and the knowledge that the adversary would require to carry out the exploitation, and an assumed threat level. SAHARA uses a table to convert SecL scores into automotive safety integrity levels (ASILs) defined in HARA. Computing SecL scores is a highly subjective activity which depends on the experience of the analysts. Given the high subjectivity of threat scores and the expert-driven nature of the analysis, we believe that results obtained with SAHARA are not reproducible. While HARA is supported by tools in the automotive sector, we are not aware of specific support for SAHARA. Finally, being grounded in HARA to generate the safety goals, SAHARA has the potential to qualify as an acceptable means of compliance (AMC) for automotive, according to `ISO 26262` [iso18].

VERDICT is a framework developed within the CASE project sponsored by DARPA [DAR21]. VERDICT is an *annex* (extension) of AADL, which aims to facilitate integrated safety and security analysis on AADL models [MLS+21]. VERDICT consists of two major components, which allow analysts to perform static as well as dynamic analysis of CPS. The first component, Model-Based Architecture Analysis and Synthesis (MBAAS) looks at the propagation of the impact of attacks through the CPS. MBAAS uses the Assume Guarantee REasoning Environment (AGREE) [CGM+12] for AADL models to infer the propagation of the effects of threats in the CPS model. MBAAS requires system designers to manually specify a set of propagation

rules - hard-coded inside each component of the AADL model. This is a manual activity essentially driven by expert knowledge, thus error-prone and not reproducible. Results gathered by MBAAS do not consider dynamics intrinsic to the system. In this direction, the second component of VERDICT aims to perform a dynamic analysis of the CPS through a process of Cyber Resiliency Verification (CRV). CRV employs model checking to verify the reachability of threat scenarios directly from the behavioural model of system components, given a limited set of threats. While MBAAS uses a subset of CAPEC [MIT23a] as the source for threats, CRV leverages a much smaller and non-standard set of threats, which is fixed in the tool and not easily expandable. VERDICT returns high-level attack patterns and threats that impact safety. Tool support is available open-source and currently maintained.

SysML-Sec extends the OMG Systems Modelling Language [H$^+$06] with security related semantics. Among other things, it augments SysML with the capabilities of capturing security requirements as well as describing attacks through *attack graphs*. SysML-Sec is supported by TTool [Apv08] which provides an environment to work with SysML diagrams. TTool integrates with ProVerif [AB05] and UPPAAL [BDL$^+$06b] aiming to aid the design of safe and secure systems through formal verification and simulation. Being an extension of SysML, SysML-Sec does not aim to synthesise safety or security requirements. In this sense, it is agnostic to the safety analysis employed to derive safety requirements and to the threat model employed to derive security requirements.

Given a set of attacks - provided using SysML-Sec own semantics - TTool can check their reachability. Traditional SysML traceability between safety and security requirements can be used to signpost safety-critical attacks. Finally, SysML-Sec also integrates the capability of verifying the impact on performance and safety of security controls [SAJ22]. TTool support is available open-source and currently maintained.

Authors of [BCEF18] propose a model-based approach to security analysis for CPS and show its application to analyse the security of a generic Flight Control System (FCS). The work develops in three steps. First, they introduce and describe, in SysML, the primitives and a taxonomy needed to model the generic architecture of a CPS and its security attributes. Thus,

transformations are defined to transform the SysML diagram into a generic model, agnostic to any definition language. Finally, they match elements of the model with potential attack vectors from the Common Vulnerability and Exposures (CVE) database. The approach proposed in this work aims to study vulnerabilities in the system architecture and is not specifically oriented to safety. Further developments in the analysis of vulnerabilities and their visualisation are brought by [BSFE18, BSC+19]. On the other hand, the type of analysis provided is static and does not consider behavioural aspects of system components or the evolution of the state of the system. Therefore, these approaches alone are not able to identify cascading effects deriving from the exploitation of vulnerabilities.

S-Cube [KBL19] combines the analysis of the system architecture with a behavioural representation of system components to automatically generate paths leading to safety-critical scenarios. S-Cube enables the generation of non-malicious (e.g., as a result of faults), malicious, and hybrid (e.g., faults combined with deliberate malicious actions) paths to accidents. S-Cube relies on a knowledge base to generate said paths, which also encodes possible failure modes for system components, a collection of potential attack steps, and rules for the propagation of failure or attacks. Attack steps and patterns in the knowledge-based are specified with respect to components of the architecture, and their coverage is subjective to the experience of the analyst. In this sense, the approach does not explicitly consider any systematic threat modelling activity. At the same time, the model does not explicitly allow the encoding of safety requirements. The absence of links to industrial standards makes it not usable for the creation and evaluation of assurance cases. S-Cube employs two techniques for the generation of the attack scenarios; these are a path-based exploration of the state space and a simulation-based approach based on Monte Carlo.

Authors of [HTLC15] present a framework to study the impact of cyber attacks on CPS, which integrates a multi-layered representation of the system with an attack model based on the cyber kill-chain. In this work, the system is represented using a logical view of three layers: *architectural*, *control*, and *physical*. The architectural layer is modelled in AADL, whereas control and physical views are represented using a mathematical model based on control theory. By mapping elements of the cyber kill-chain to the control layer of the system, authors aim to

model the impact on the CPS at individual stages of the attacks. The proposed framework is used to develop (weaponisation) attacks targeted at specific architectures, and also to simulate the impact of these attacks. The use of the framework is demonstrated in a simulated use case featuring an unmanned aerial system (UAS). Overall, the framework provides the tool to analyse the potential impact of attacks against control properties of the system such as *controllability* and *observability*. In addition, the existence of a mapping to the AADL facilitates the integration between the control layer and the attack surface. On the other hand, the work is not specific to safety and does not provide means to represent safety requirements.

Recently, [PGM21] have presented an interesting study on the need for sound approaches able to capture key safety and security requirements in line with industrial standards. The first part of the study presents a survey of the state-of-the-art on safety and security. Thus, the authors elaborate on a new *goal-oriented framework* with the aim of aiding integrated analysis. One of the key aspects of their approach is the transposition and re-use of elements from existing (standard) frameworks without the need to introduce new notations. By doing so, the authors ensure that their approach is a potentially valid means for building assurance cases. At the same time, the type of analysis proposed is still characterised by a heavy manual component and remains driven by expert knowledge. The proposed framework is specified at the system level and enables quantitative reasoning through goal-based notation. The authors demonstrate its effectiveness through an application to a real-world use case featuring a water supply network.

System Theoretic Process Analysis for Security (STPA-Sec) STPA-Sec [YL13, YP17] extends STPA [Lev16] to the security domain by including *deliberate malicious actions* amongst potential causes for accidents. Stemming from STPA - an acceptable means of compliance (AMC) already - STPA-Sec is also a widely accepted method for integrated safety analysis, and it is applied to identify safety-critical threat scenarios across a number of different sectors [PCKDG22], including industrial control systems [KMM18], microgrids [BW19] and aerospace [SMMY18]. The EUROCAE `ED-203A` *Airworthiness Security Methods and Considerations* includes STPA-Sec as AMC for airworthiness security certification [EUR18]. The approach proposed in this thesis is grounded in STPA and STPA-Sec. Thus, given the importance of

STPA-Sec in the context of our approach, we provide a detailed overview of its structure and an example of application in the next section (Section 2.6). STPA and STPA-Sec provide a qualitative output, i.e., scenarios of potential accidents, where harm can be caused.

Several integrated approaches stem from STPA-Sec. STPA-SafeSec [FMS+17] extends STPA-Sec by proposing a correspondence between the STAMP model - which captures the information within the CPS at the functional layer - and the deployed architecture of the CPS. Authors leverage this relation to analyse the impact on the safety of a concrete (and custom) set of threats against system architecture. STPA-SafeSec provides the tool to understand the direct impact of these threats and manually investigates cascading effects on the STAMP model via causal analysis. The output of the process is a set of high-level threat scenarios showing threats that can be used to cause harm. STPA-SafeSec remains a manual process that is driven by expert knowledge and, as such, is error-prone. Moreover, produced results are characterised by a degree of subjectivity depending on the expertise of the analysts and, to this extent, results cannot be considered reproducible.

Authors of [KM21] propose *CyberSafety* methodology to analyse the impact of cyber-security on systems safety through the analysis of the STAMP model, thus at the functional level. The objective of the work is the refinement of STPA-Sec; this is achieved through a description of the steps that compose it, and a demonstration of its application on an electric distribution system. CyberSafety shares the same advantages and shortcomings of STPA and STPA-Sec (Section 2.6). Overall, it offers a systematic approach to the safety analysis, being independent of the system architecture, it is applicable starting from the early design stages. Moreover, it allows for a thorough - although manual - analysis of cascading effects within the CPS. On the other hand, it remains a manual approach and is driven by expert knowledge. To this extent, it is also error-prone and the completeness of results depends on the expertise of the analysts. For this reason, the results provided as an outcome of this approach cannot be considered reproducible in our opinion.

The research proposed in [DSCdMBH20] extends STPA with STRIDE. The objective of the work is to introduce a systematic approach to the identification of those *deliberate malicious*

actions which in STPA-Sec are potential root causes for accidents. On the other hand, the investigation of threat scenarios across the control structure is again carried out manually. Therefore, this approach shares the same issues of STPA-Sec and CyberSafety regarding objectivity and the reproducibility of results.

Authors of [SMMY18, SMGY18, MSMY19] propose a refinement of STPA-Sec which addresses security requirements as mandated by NIST SP-800-160 [RPG+21]. They show a thorough application of STPA-Sec on two complex systems, namely, a representative aerial refuelling system [SMMY18] and a notional autonomous space resupply vehicle [MSMY19]. The main objective of the works is to demonstrate the systematic application of STPA-Sec through the development life-cycle of safety-critical systems. Namely, they organised the application of STPA-Sec over three macro-phases: *concept*, *architecture*, and *design*. For each phase, several iterations are required. Overall, the work improves the application of STPA-Sec, making it more systematic. An additional improvement relates to the threat model which is now related to a standard approach (SP-800-160). As for traditional STPA-Sec, the proposed approach allows for a thorough, yet still manual, analysis of cascading effects deriving from threats. On the other hand, it remains a manual approach driven by expert knowledge.

Authors of [HBCS19] propose a methodology which extends STPA, aiming to perform integrated safety and security analysis. In particular, the objective of the authors is to identify *critical requirements* that mitigate hazards as well as security threats. Once identified, the requirements produced are then formally checked against a behavioural model of the system. The process of generation of critical requirements is manual and also driven by expert knowledge. In addition, the approach taken to model security threats does not appear systematic.

Pereira et al. propose an ontology to support systematic safety and security analysis with STPA and STPA-Sec in [PHNT19]. Finally, a STAMP-based approach was also applied by Nourian and Madnick to investigate the Stuxnet case [NM15]. Authors show the effectiveness of Causal Analysis based on STAMP (CAST) as a method to investigate cyber-incidents when *deliberate malicious actions* are included amongst potential causes for accidents. Their approach shares the same shortcomings as STPA-Sec and CyberSafety in terms of scalability and reproducibility

of results. Their output produces recommendations which highlight current vulnerabilities and can be used to improve the design and prevent further attacks.

2.5.2 Methods based on trees and graphs

The following selection of approaches proposes formalisms that extend the structure of attack trees or rely on a graph-based representation to perform integrated safety and security analysis.

Authors of [PCB10a, PCB10b] propose an approach based on Boolean Driven Markov Processes (BDMP) to perform integrated safety and security analysis. The formalism extends attack trees with the aim of performing dynamic analysis while, at the same time, also considering dependencies between safety and security events. In particular, BDMP associates a Markov process to each event (leaf) in the attack tree; it also introduces the notion of *trigger* to represent dynamic aspects of attacks, i.e., events enabling other events, or ordered sequences of events. The implementation of the formalism is discussed in [PCDB11], while a first example of application is shown in [PCB10b]. The approach is further used to assess the risk of safety-critical attacks in cyber-physical systems such as, among others, a pipeline control system [KBC+14]. The method is based on attack trees the construction of which is highly subjective and depends on analyst expert knowledge. It is worth highlighting that BDMP, and more in general, all approaches based on attack trees are not a replacement for wider safety analysis tools such as STPA, FMEA, etc. In fact, the latter aims to identify new modes of failure (or, in the case of STPA-Sec, new attacks). BDMP, on the other hand, offers the means to quantify the risk associated with found critical modes of failures. Another issue generally associated with stochastic approaches applied to security analysis is the limited knowledge that we have on the behaviour of an adversary. In this sense, the estimation of the likelihood of a security event, such as an exploitation of a vulnerability, is not trivial. Finally, Markov processes are memoryless, i.e., they rely on the assumption that events are random. While this is a valid assumption in the field of safety, the same cannot be said in security, where attacks are not random yet driven by an adversary. These limitations are widely discussed in [PCB10a, PCB10b].

Longari et al. [LCCZ19] propose a framework to perform a goal-oriented security analysis of

automotive systems. The threat model is based on that employed by EVITA [RWW+09] and uses a formalism based on attack trees. The authors assume that the adversary has (physical or remote) access to the vehicle and can read and write information on its network. On the other hand, they do not rely on a systematic tool (e.g., STRIDE) to identify individual threats against components. The aim of the work is twofold. It allows for quantifying the risk for found threats. It also enables the synthesis of new network topologies which minimise said risk. While safety properties can be among the goals, the analysis is not exclusive to safety and can also take into account other attributes including costs and performance. The authors identified an additional limitation to their approach, which is that synthesised architectures might not be applicable due to constraints not expressed by the model, e.g., real-time constraints. The threat model employed requires manually selecting the objectives for the analysis and cannot automatically discover complex attacks that rely on cascading effects, which we take into account in our approach. In fact, [RWW+09] defines attack trees manually, combining potential motivations behind attacks with system use cases. Furthermore, both works are specific to the automotive sector.

Authors of [KNKS22] introduce the formalism of the Attack Fault Maintenance Tree (AFMT) with the aim of identifying trade-offs across *safety*, *security* and *maintenance*. The analysis consists of stochastic model checking, which derives the parameters queried (metrics for security, safety and maintenance) through simulation. The latter makes strong assumptions about the attack steps, their duration, detectability, and success. AFMTs provided as input appear to be manually specified, as the authors do not explain how they are built nor how security or safety requirements are derived. The authors assume the methodology to be agnostic to the threat model as well as to its underlying safety framework. The manually defined attack-failure trees must identify all the attack steps and how they are connected. Separate trees must be provided for each scenario. The methodology is currently supported by a web application.

Authors of [EVCC+22] propose an approach based on the analysis of closed control loops (CCL) to identify critical attacks, i.e., with the potential to impair functionality, in industrial control systems. In their threat model, the adversary has knowledge of the information flow within the CPS. The work was validated in simulation and identified scenarios leading to the shutdown

of operations in a model of the Tennessee Eastman Plant. Differently from other approaches mentioned in this section the aim of which was to secure the system under analysis, the objective of [EVCC+22] is the identification of critical attack. We believe the approach is relevant to this section as critical attacks are discovered by analysing the system architecture in combination with the information flow (CCL).

Authors of [BH21] use an approach based on AND/OR dependency graphs to identify the *Most Likely Mission-Critical Component Set*, i.e., the set of critical components for which the probability of mission failure is maximised. The work is not specific to safety, but instead, it looks at a broader notion of mission. It assumes that a number of analyses, including safety analysis, are carried out before its application to identify the requirements objective of the analysis. In this sense, a study of the MLMCS cannot replace the current safety analysis. At the same time, as for the case of BDMP, it would provide interesting insights into threat scenarios discovered during the safety analysis. One limitation of the work relates to the construction of the AND/OR dependency graph, which we understand is a manual and expert-driven process. The authors show the application of the approach on a representative flight control system and discuss possible applications of their approach to improve the security posture of CPS and investigate the root causes of accidents. The proposed approach was implemented and is supported by a tool. In terms of performance, the analysis is stable and shows the ability to easily scale to up to thousands of nodes. Finally, being a stochastic method, the proposed methodology is also limited by the quality of input data, in particular with respect to the probability of security events.

Authors of [SKG21] introduce a recursive approach for building attack graphs for IoT systems based on both cyber and physical interactions. The proposed method extends existing methodologies by also considering physical interactions as part of the attack paths. Computed paths are quantitatively assessed to identify the associated risk. As for the quantitative methods introduced earlier, in this approach, too, the estimation of the likelihood of individual security events inside the attack paths constitutes a limitation. Authors refer to CVSS [NIS] to set the risk of part of said events. The proposed approach works on the architecture of the CPS and cannot identify design flaws or cascading effects stemming from the exploitation of threats. At

the same time, the proposed approach does not take into account aspects related to safety.

2.5.3 Expressing systems dynamics

The analysis of the internal dynamics of the cyber-physical system is critical to fully understand the impact that malicious actions have on system high-level properties, including safety. Though not directly causing harm, the exploitation of certain threats can still induce accidents through cascading effects, or render the system more susceptible to additional attacks.

Authors of [AHH+13] introduce the notion of *cyber-cycle* - rooted in cybernetics [Ash61] - to describe the interactions between the attacker and the system. They observed that during an attack, the adversary adapts its actions also based on the defender's response. Such interaction is defined as cyber-cycle. Breaking this cycle before it is too late would prevent damage to the CPS. In their research, authors of [RBD16] see two applications of cybernetics to security: to analyse supply, operations, and life-cycle of CPS, and to model the interaction between attacker and defender (cyber-cycle). Overall, the impact of cybernetics applications to security interests both prevention and mitigation aspects.

In this section, we present an overview of two main approaches employed in literature to study how cyber attacks can alter the internal state of a system. Both classes of approaches stem from cybernetics; one is rooted in *control theory* while the second in *finite state machines*.

Control theory In control theory, the dynamics of the physical process and control laws are described by a mathematical representation. Interactions between components, i.e., controllers and processes, are represented as signals in time, frequency, or Laplace domains. In this representation, the objective of the attacker is to *disturb* operations, eventually leading the system towards unsafe states [CAL+11, CAS08, MFL+13].

Teixeira et al. introduce a complete framework grounded in control theory to characterise attacks against CPS [TPSJ12]. Specifically, they propose a formalisation for disclosure, deception, and disruption attacks. In the same work, they also propose state observers as means

to detect anomalies. The design of state observers to detect and isolate attacks on networked control systems is further explored by [TSSJ15, USC17, CCQ17].

An interesting framework to mitigate attacks on discrete Linear Time Invariant (LTI) control systems was proposed by [KGM+18]. Authors exploit the notion of actuator saturation in physical actuators to limit the impact of malicious actions, preventing the system from reaching dangerous states where integrity and safety are not guaranteed. To do so, they reduce the working area of an actuator to an artificially crafted (safe) domain, smaller than the normal working domain of the real actuator.

Fang et al. propose a novel approach that exploits *dual encoding* techniques to achieve a secure networked control [FJS+19]. Their work develops in two steps. They prove, in Laplace space, the equivalence between their proposed model and the standard control models, thus they show that their model is robust to eavesdropping attacks. In particular, they show that the transfer function reconstructed from intercepted communication is different from the transfer function of the plant. Thus, leaked information alone cannot be used to gather insights on how to destabilise the physical process. Finally, in [TASJ18] a distributed control framework is leveraged to mitigate attacks against a networked system. Their approach is founded on the isolation of infected nodes, followed by a reconfiguration of the connection topology.

Overall, the main advantages brought by approaches to safety and security analysis based on *control theory* lie in their quantitative nature. Such characteristics make them suitable for the design of physics-based anomaly detectors and more robust controllers. At the same time, adversaries can use these approaches to compute how much disturbance is needed to cause damage or design signals undetectable to observers aiming to realise stealthy attacks. The main limitation of these is that the complexity of the models significantly increases in the presence of non-linearity and channels subjected to delays.

State machines The use of formal verification to discover flaws in complex systems and protocols dates back to 1998 (at least) when Mitchell et al. employed it to find multiple attack paths in SSL [MSS98]. Researchers at Amazon AWS have also applied model checking to

find complex sequences of actions leading to unexpected behaviour [NRZ+14]. Although this last work does not consider an adversarial environment, it is a sound example of a successful application of formal verification to discover complex errors in large distributed systems.

More recently, Hussain et al. combined model-checking with cryptographic protocol verification to identify vulnerabilities in Long Term Evolution (LTE) [HCMB18]. In particular, the authors represented message exchanges in the LTE protocol using state machines and constructed a behavioural model of an adversary targeting the integrity and confidentiality of the communication. They uncovered logical issues through the formal verification of properties against the *instrumented* model of the system. This is the parallel composition of the model of the protocol and that of the adversary. The main objective [HCMB18] is to use formal verification to discover vulnerabilities in a specific communication protocol. In contrast, Cassandra looks at the control structure of a cyber-physical system to identify safety-critical attacks. In doing so, our approach is generic and integrates with the safety and architectural model of the system. Moreover, Cassandra also introduces a systematic approach for deriving the behavioural model of the adversary.

In the scope of cyber-physical systems security, CVAnalyzer [HCS+21] combines traditional with probabilistic model checking to respectively, discover new vulnerabilities in inter-vehicle communication protocols and evaluate their risk. The approach is not directly aimed at the discovery of safety violations. Instead, authors look for more generic *availability* issues, which can result in disruptions as well as safety-critical events such as collisions. The input model verified by CVAnalyzer is a Finite State Machine (FSM) of the system, whereas the threat model assumes that the adversary can send malicious packets to other vehicles in the network through a compromised on-board unit (OBU) device. Issues with the communication channels, such as packet loss and latency, are not considered due to their complexity in model-checking. Although this assumption is not realistic, it plays in favour of the adversary which is placed in ideal conditions. At the same time, state space reduction is needed for model-checking operations to be tractable. Traditional model checking provides the tool to evaluate the impact of threats through cascading effects within the CPS and, at the same time, ensures that results are reproducible. On the other hand, considered threats along with the verified *availability* queries

are not systematically identified, which is understandable given the scope of the paper and does not diminish the quality of the work. Attacks discovered by CVAnalyzer are implemented to verify their feasibility.

Lanotte et al. [LMMV20] propose a framework to model and detect attacks against sensors and actuators in cyber-physical systems. The formalisation proposed by the authors primarily focuses on attacks targeting the integrity and availability of sensors and actuators and also considers timing aspects. With their semantics, authors are able to classify attacks based on the targeted components, starting time and duration. The authors used the proposed framework in combination with UPPAAL-SMC [DLL+15] with the aim of detecting attacks against a representative CPS. The main objective of [LMMV20] is the introduction of a new formalism to describe low-level attacks aiming to improve simulations and detection. In contrast, Cassandra looks at the entire control structure of the CPS to identify safety-critical attacks. In doing so, Cassandra integrates with the safety and architectural models of the system.

Alshalalfah et al. [AHM19] uses priced timed automata (PTA) in UPPAAL-SMC to verify the safety properties of an implantable medical device, specifically an insulin pump, against attacks in simulated adversarial scenarios. In particular, the work evaluates the safety of two types of controllers in an attack-free scenario and elaborates on the safety violations that verify when the system is device to replay attacks. Overall, the work shows that PTA is a valid tool to evaluate the behaviour of the safety-critical systems under attack. On the other hand, the study makes strong, unrealistic assumptions about the threat model, namely, they only consider replay attacks. Moreover, the simulation-based approach does not guarantee reproducibility of results.

Finally, Poorhadi et al. developed a framework based on Event-B to analyse the impact of cyber attacks on the safety of a communication-based train control (CBTC) system [PTD21]. As for previous cases, the focus of this work is the development of a strategy to model the different effects of the system behaviour and verify them in adversarial conditions. To this extent, the safety requirements analysed are not identified systematically. At the same time, the adopted threat model is restricted to the violation of confidentiality and integrity of only

one class of messages.

2.5.4 Discussion

Whilst, to the best of our knowledge, the approach to integrated safety and security analysis proposed with Cassandra is novel, we have seen how several studies in literature, have proposed steps in similar directions.

Approaches stemming from STPA-Sec share part of the safety analysis at the functional level with Cassandra. At the same time, they are manual, error-prone, and require an important degree of expert knowledge to be executed. These characteristics make their output human-dependent and, for this reason, not reproducible. Our proposed approach integrates systematic aspects from STPA-Sec with STRIDE threat modelling and formal verification to tackle these limitations. Provided a behavioural model of system components, we use model-checking to uncover a set of safety-critical scenarios that is sound and complete with respect to the threats and component behaviours and the model-checking bounds. In Cassandra, we also analyse the architectural layer of the cyber-physical system; in particular, we identify the privileges required for the threat scenarios and the potential attack paths leading to them.

The approach proposed by VERDICT [MLS+21] also shows a number of fundamental differences with our methodology. MBAAS requires the system designer to specify a set of propagation rules, hard-coded inside each component of the AADL model. In contrast to VERDICT (MBAAS), where system designers are required to specify a set of propagation rules hardcoded in each component, our approach can also rely directly on the behavioural model of system components to analyse the impact of the attack. This capability is also supported by VERDICT (CRV) but only for a limited fixed set of custom threats while we rely on STRIDE to systematically construct the attacker model. Moreover, VERDICT returns high-level threats that can potentially impact safety, whereas we analyse the attack graph of the system architecture and also determine the attack paths leading to those threats. Furthermore, Cassandra extends STPA-Sec through the automatic analysis of cascading effects whereas safety requirements are assumed already provided in VERDICT. Finally, another difference lies in the representation

adopted to represent safety and security requirements. In VERDICT, safety and security requirements are defined at the same level as to entail the equality of safety and security with respect to the mission. On the other hand, we believe in security guarantees as a means to achieve safety. The approach proposed by S-Cube [KBC+14] also shows a number of funda-

Reference	Name	Formalism	Safety model	Threat model	Output	Cascading effects	System state	Threat map	Reproducible results	Assurance	Tool support
[YL13]	STPA-Sec	STAMP	STPA	-	Qualitative	◐	◐	○	◐	●	●
[NM15]	-	STAMP	CAST	-	Qualitative	◐	◐	○	◐	●	○
[FMS+17]	STPA-SafeSec	STAMP Component diagram	STPA	Custom	Qualitative	◐	◐	◐	◐	●	○
[AR13, RA15]	SysML-Sec	Sys-ML	-	-	Quantitative	◐	●	◐	◐	○	●
[MSB+15a]	SAHARA	Component diagram	HARA	STRIDE	Quantitative	◐	○	◐	○	◐	○
[BCEF18]	-	Custom	-	-	Quantitative	○	○	●	●	○	●
[SMMY18, MSMY19]	-	STAMP	STPA	NIST SP 800 – 160	Qualitative	◐	◐	◐	◐	●	●
[DSCdMBH20]	-	STAMP	STPA	STRIDE	Qualitative	◐	◐	◐	◐	●	●
[KM21]	CyberSafety	STAMP	STPA	Custom	Qualitative	◐	◐	○	◐	●	●
[HCS+21]	CVAnalyzer	State machine	*availability*	Custom	Both	●	●	●	●	○	●
[EVCC+22]	-	Closed control loop	-	Custom	Qualitative	●	●	●	●	○	○
[PGM21]	-	Goal based notation	Standard	Standard	Quantitative	◐	◐	○	○	●	●
[SKG21]	-	Attack graph	-	CVE	Quantitative	○	○	●	●	○	●
[MLS+21]	VERDICT (MBAAS)	AADL	Custom	CAPEC	Qualitative	●	◐	●	●	○	●
[MLS+21]	VERDICT (CRV)	AADL	Custom	Custom	Qualitative	●	●	●	◐	○	●
[KBL19]	S-Cube	Custom	-	Custom	Both	●	◐	◐	●	○	●
[HBCS19]	-	STAMP	STPA	Custom	Qualitative	◐	◐	○	◐	●	○
[CL23] [CSP+23] [CL20, CHC+22]	CASSANDRA*	STAMP Network of timed automata Attack graph	STPA	STRIDE CVE	Both	●	●	●	●	●	●

(*) **Our work**

Table 2.2: Overview of related works

mental differences with our methodology. The first key difference is that, being grounded in STPA, Cassandra explicitly identifies attack paths leading to violations of safety properties, whereas S-Cube does not model safety requirements. Furthermore, in Cassandra, we leverage the systematic application of STRIDE on the STAMP model of the system to identify threats. This is not the case with S-Cube, where generic attack steps against system components are arbitrarily constructed, mainly relying on expert knowledge. Cassandra allows for two distinct representations of component behaviours (Chapter 4), both mapped to a safety model of the CPS. On the other hand, S-Cube relies on Figaro to describe the behaviour of system components and is not backed by a safety model. Another key difference lies in the process of identification of attacks that in Cassandra is done at the functional level (Chapter 5) through

formal verification, whereas S-Cube relies on path-based exploration on the system architecture and Monte Carlo simulation.

SysML-Sec [AR13] and Cassandra are characterised by different intents. The former offers an integrated framework to verify the impact of security measures on overall safety and performance or that, given an attack, enough security requirements are in place to protect against it. Unlike Cassandra, it relies on attack descriptions provided in input by the analyst. On the other hand, the aim of Cassandra is to discover new safety-critical scenarios in adversarial settings.

Similarly, the work proposed in [HBCS19] and Cassandra are also characterised by different intents. In fact, the former aims to generate *critical requirements* which, when satisfied, prevent security threats. Authors of [HBCS19] employ a formalisation of the system behaviour in Event-B to verify the requirements produced. In contrast, Cassandra uses formal verification on a Network of Timed Automata (NTA) to identify and enumerate threat scenarios. In doing so, we also define a formal correspondence between the STPA and the behavioural model of the CPS. Moreover, the security model adopted by [HBCS19] is not based on a systematic threat modelling framework and requires the control structure to be manually annotated with a set of *manipulation points*. Furthermore, [HBCS19] does not integrate with the system architecture, which limits the implementation of the *critical requirements* found. Lastly, authors of [HBCS19] define *adversarial actions* and *hazards* at the same level, which we believe is inaccurate. In contrast, Cassandra models adversarial actions as potential causes for hazards. In fact, in our approach, the adversary provokes hazards through the execution of one or more (i.e., sequences) of adversarial actions.

The work proposed by [HTLC15] and Cassandra are also characterised by different intents. The former offers a framework to map elements of the cyber kill-chain to the control layer of the system, whereas our methodology is aimed at the automatic identification of threat scenarios. Both approaches rely on a layered representation of the CPS; however, [HTLC15] leverages control theory to model system behaviours whereas our approach uses state machines. Finally, Cassandra also considers a functional view of the CPS as a further layer of the system model.

In Table 2.2 we show a summary of the methodologies discussed in this section. We have intentionally left out from the table works that are not aimed at the identification of safety-critical attacks and attack paths. This includes methodologies focused on risk assessments on attack trees. For each methodology, we report the input model(s), the underlying safety and security framework(s), and the type of output. The latter can be qualitative, including safety-critical attacks, or quantitative, i.e., in the form of a risk analysis. In addition, we also report the degree of granularity that proposed approaches employ to represent system states - if applicable - and their capability to deal with dynamic analysis of cascading effects, e.g., through automatic inference on systems behaviour. Other relevant aspects are the flexibility of the threat model (i.e., how extensible it is), their capability of producing results that are reproducible, their applicability as means of compliance, and, finally, the provided tool support.

We consider the following cases for the criteria outlined above

Evaluation of **cascading effects**: (○) not supported; (◐) supported but manual; (●) supported by automated analysis.

System state: (○) not supported; (◐) coarse grain (e.g., components internal state is hidden); (●) fine grain (e.g., components internal state is explicitly represented).

Type of threats: (○) threats not linked to architecture; (◐) generic threats but linked to architecture; (●) threats linked to specific vulnerabilities in architecture.

Reproducible results: (○) Analysis is subjective; (◐) Analysis is systematic but expert-driven; (●) Analysis is systematic and model-driven.

Assurance: (○) Safety framework is not an acceptable mean of compliance (AMC); (●) safety framework is AMC.

Tool support: (○) Tool support not available; (◐) tool support available but not currently maintained; (●) tool support available and currently maintained.

2.6 System Theoretic Accident Models and Processes (STAMP)

In this Section, we provide a description of STAMP and outline the off-the-shelf application of System Theoretic Process Analysis on STAMP, and its variant for security STPA-Sec [Lev16, LT18, YL13, YP17].

Systems Theoretic Accident Model and Process (STAMP) is a safety model grounded in systems theory [Lev16], which sees accidents as the result of flaws in the interactions between system components. In particular, STAMP can represent three aspects of a CPS that are relevant to its safe operation: safety constraints, control hierarchy and process models. Safety constraints are typically defined as a function of the state of components in the CPS and of the information they exchange with each other. Enforcing safety constraints is critical in preventing reaching hazardous scenarios (see also Section 2.1). The control hierarchy is a control block diagram that shows the information flow exchanged between components of the CPS. STAMP defines a Process Model for each controller in the control hierarchy. In fact, the process model provides a representation of the view that the controller has on the controlled processes. The authors of STAMP do not provide a formalisation of the process model or an approach to formalise a decision algorithm given a process model. While this design choice allows us to easily apply STAMP almost universally across a variety of different domains, it also makes the safety analysis process not systematic, and hugely dependent on subjective expert knowledge.

In the paragraphs below, we provide a more in-depth description of STAMP and use it to model the safety aspects of our Communication-Based Train Control system use case.

System level hazards and safety constraints The STAMP modelling process starts with the identification of system-level losses and accidents, and continues with the identification of the hazards. System-level safety constraints are designed following the identification of hazards and are iteratively refined during the safety analysis. Failing to enforce a safety constraint results in *hazard*, which can lead to an *accident* [How04].

We consider the following accidents; A_1 collision between two trains, and A_2 train derails (due to elevated speed) as these lead to the heaviest losses and endanger human life [CDP+17]. For completeness, we also consider an additional accident A_3 traffic disruption, as an example of a less important accident. In fact, although (A_3) is not life-threatening, it has the potential to provoke major economic damages to the company responsible for the transport. We identify a first set of three hazards: Train enters a not cleared zone, Over-speed and Train Halts and express them in the form *(Hazard, Description, Related Accidents)* in Table 2.3.

Table 2.3: Table of Hazards (refined) for the CBTC system.

#	**Hazard**	**Accidents**
H_1	Train enters a *not cleared* zone	A_1, A_3
H_2	Train *authorised* but does not enter a *cleared* zone	A_3
H_3	State of controlling authority is not consistent with the position of trains	A_1, A_2, A_3
H_4	Train travels below minimum speed	A_3
H_5	*Clear* zone wrongly marked as *not clear*	A_3
H_6	*Not clear* zone wrongly marked as *clear*	A_1
H_7	Train travels above maximum speed	A_1, A_2

In Table 2.4 we express related system-level safety constraints in the form *(Safety Constraint, Description, Linked Hazards)*.

Table 2.4: Safety Constraints - Communication Based Train Control

#	**Description**	**Mitigates Hazards**
SC_1	Train must not enter a zone that is not clear	$[H_1]$
SC_2	Train must not halt before a clear zone	$[H_2]$
SC_3	Train must not mark a zone as *clear* before having left it	$[H_3, H_6]$
SC_4	Train must not request entry to a zone without planning to enter it.	$[H_3, H_5]$
SC_5	Train speed must always be within constrained limits	$[H_4, H_7]$

Control hierarchy We introduce the formalism of *Safe Control Structure (SCS)* to describe control hierarchy and process model [CL20, CHC+22]. The SCS is defined as a tuple (C, D, K) where C is the set of components, D is the control flow and K the set of functional channels. Components in C can be *controllers, sensors, actuators* or *physical processes.* $D \subseteq C \times C$ is the set of control flows. We use it to represent the data that components exchange with each other. In particular, $D = CA \cup FBK$ with CA and FBK being, respectively, *control actions* and *feedback signals.* Functional channels in K represent the means over which the control flow

is exchanged and serve to logically group together control actions and feedback. At most, the number of functional channels reaches that of the control flow, in the particular case where a channel exists for each flow. Each controller is characterised by a *process model* and a *control algorithm*, which we outline below.

For simplicity, we assume that control actions are all provided instantaneously as discrete individual events or messages. The cyber-physical nature of the system, however, requires that we also model low-level control actions that must be provided for a continuous amount of time. For instance, the cruise controller of a car controls the throttle actuator providing a certain acceleration profile until the vehicle has reached the desired speed. Similarly, a low-level controller of a valve in a water distribution plant transmits a signal to the actuator until the valve has reached the expected position (e.g., open/close). To capture these aspects, we model control actions that need to be provided for a continuous, non-discrete, amount of time as two separate events (or messages) in the SCS. The two discrete events respectively sanction the *beginning* (start) and the *end* (stop) of the provision of the continuous control action.

Process models and control algorithm The controller's process model retains a representation of the controlled physical process. The state of the controlled process is memorised as a function of the process model variables. The control algorithm, on the other hand, determines the transformation to the output, given the current state and input. Thomas et al. show in [TI13] that *Process Model Variables* can be defined hierarchically, meaning that it is possible to have higher level abstract variables defined as a combination or function of lower level ones.

Process Model and *Control Algorithm* are used in STAMP to describe the view that the controller has of the physical process and the decision algorithm. In this thesis, we model the behaviour of the controller as a Labelled Transition System (LTS). States in the LTS are defined in functions of the process model variables whereas the transitions between states define the control algorithm. A formal description of our approaches to modelling the behaviour of controllers is provided later in this Section.

In Figure 2.4, we show the safe control structure of the system. Control actions (red) and

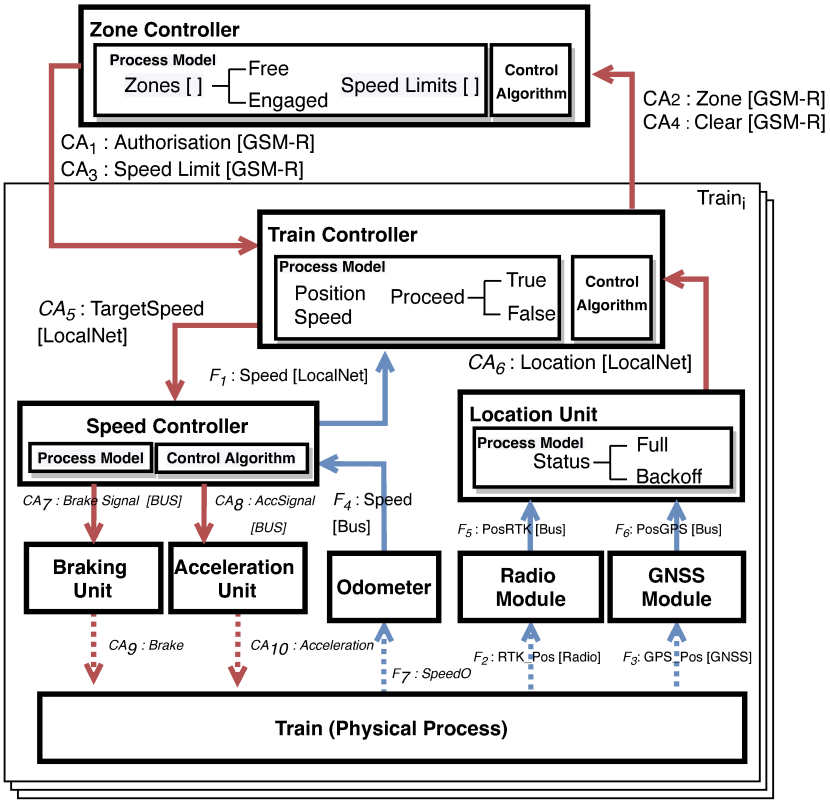


Figure 2.4: Safe Control Structure of Communication Based Train Control system.

feedback signals (blue) are exchanged over digital (dashed) and physical (dotted) channels. The system comprises *one* zone controller, and *three* controllers for each train: a *train controller*, a *location unit* and an *acceleration unit*. ZC is responsible for issuing control actions CA_1 (Authorisation) and CA_3 (Speed Limit) in a timely manner over a GSM-R channel. The process model of the ZC stores which zones are free and which are engaged, along with their speed limits. The status of a zone is a binary value indicating whether it is *free* or *engaged*, while the speed limit is an integer value. CA_1 carries the value *allow* or *deny* while CA_3 is a positive integer. Both control actions depend on the internal status of the controller (*Zones* and *Speed Limits* arrays, respectively) and the position of the recipient. The Train Controller holds, in its process model, the current position (zone) and speed as well as a binary value, which indicates whether the train has been cleared to proceed or not. TC issues CA_2 and CA_4 to respectively request access to and clear a specific zone. CA_2 and CA_4 carry the id of the issuer train ($tid \in N$) along with the id of the zone ($i \in M$) and also depend on the internal state of the controller for the current position. TC also sets the target speed CA_5 on the speed controller and receives the current speed from the speed controller (F_1). The Location Unit

is responsible for sending the current location CA_6 to TC. The value of CA_6 depends on the operating mode (*full, fallback*), which is saved in LU's process model as a binary variable. The LU receives its current position from the sensors Radio (F_5) and GNSS (F_6). The LU relies solely on GNSS (F_6) when it is operating in *fallback* mode. The Speed Controller receives the target speed from the TC and, given the current speed (F_4) computes an acceleration profile. The latter is used to command the brake (CA_7) and acceleration (CA_8) units. The odometer, radio sensor and GNSS module respectively receive feedback signals (F_7), (F_2) and (F_3).

2.7 System Theoretic Process Analysis (STPA)

System Theoretic Process Analysis (STPA) is a process for safety analysis introduced to study the dynamics leading to accidents (and their respective losses) on a STAMP model [Lev16]. To avoid ambiguities we rely on the definition of accidents and hazards proposed by Leveson (Section 2.1). In the literature, we find two distinct descriptions of the steps involved in STPA. In [Lev16], STPA is described as a two-step process to apply to the STAMP model. The same authors organise STPA in a $four$ steps process in [LT18]. As a matter of fact, the first two steps of [LT18] lay down the instruction to derive the STAMP model whereas [Lev16] assumes that the latter had been produced already. In fact, the last two steps of [LT18] match exactly the description of the accident analysis process as provided in [Lev16].

Cassandra relies upon System Theoretic Process Analysis Security (STPA-Sec) to analyse the safety of the CPS with respect to security threats and derive security constraints aimed to mitigate them. In the previous sections, we have mentioned that STPA-Sec is built on System Theoretic Process Analysis (STPA) and the fundamental difference between the two lies in the identification of possible causes for accidents, in the last step. In fact, while STPA looks at *faults* and *miscommunication* as possible root causes for accidents, whilst STPA-Sec finds them in *threats* [YP17].

In our use cases, we refer to the organisation in $four$ steps of STPA/STPA-Sec provided by [LT18] (Figure 2.5). The steps involved are represented in white rectangles whereas the

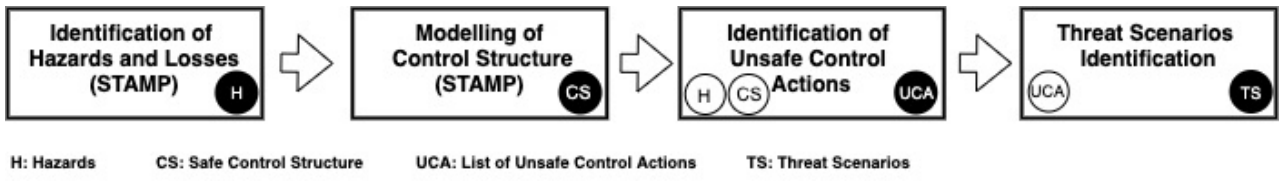


Figure 2.5: STPA-Sec Overview

input processed and the output produced at each step are shown in white and black circles, respectively. The acronyms used in the diagram are expanded in the key at the bottom of the Figure.

Steps 1 and 2 of STPA/STPA-Sec provide guidance towards the definition of the STAMP model for the system [Lev16]. In fact, the first step is the definition of system-level hazards, losses, and safety requirements, followed by the second step, which is the modelling of the safe control structure. We have elaborated on these two steps in the previous section.

The third step of STPA/STPA-Sec defines the *Unsafe Control Actions (UCA)*, those control actions (and feedback) that can cause hazards when applied in a specific *context*. The *context* of a UCA describes the combination of values of variables in the process model of the issuing controller for which the application (or missed application) of the control action is unsafe [TI13, AW15]. Formally, a UCA can be defined as a tuple $(ca, ctx, r, type)$ where $ca \in D$ is the control action, ctx the *context*, r is the value for which the hazard subsists and *type* the type of the UCA [AWL15]. STPA/STPA-Sec distinguishes among *four* types of unsafe control actions. In this sense, a control action can be *unsafe* if it is: *provided* when not required or *not provided* when required. In addition, a UCA in the case *provided* can be: simply *provided* but also *provided too early, too late or in the wrong order*, or *provided with wrong duration* [Lev16]. The *Context Table* of a controller has been introduced in [TI13] as a tool to systematically enumerate UCAs.

The fourth step of STPA/STPA-Sec is the identification, through the whole safe control structure, of possible scenarios of applications of UCAs. We recall that the fundamental difference between STPA and STPA-Sec lies in the domain of the causes identified at this stage (faults for STPA, and security threats for STPA-Sec). Traditionally, the $four$ steps of STPA are all performed by hand, mainly relying on expert knowledge. Although there have been proposals

to automate at least part of the analysis using techniques based on formal verification and model-checking, they do not consider security threats [TI13, AW15].

As an example, we apply STPA/STPA-Sec to our CBTC use case. We start with the identification of losses, accidents and system-level hazards, which we have carried out in the previous section while discussing STAMP (Table 2.3) (step 1). We have also derived the safe control structure (step 2), which is shown in Figure 2.4.

We proceed with the identification of the following Unsafe Control Actions (Step 3). To do so, we build the context table for the train and the zone controller in Table 2.5.

Table 2.5: Context Table of Zone Controller (ZC) and Train Controller (TC). r is the value for which CA is unsafe, v is the speed limit.

Controller	**CA**	*Zone*[i]	Limit	Position	Previous	Proceed	**Not Provided**	**Provided**	**PE**	**PL**
ZC	CA_1	Engaged	-	-	-	-	H_4	H_1 [$r = allow$]	-	H_4
ZC	CA_1	Free	-	-	-	-	H_4	H_2 [$r = deny$]	-	H_4
ZC	CA_2	-	-	$i \in M$	-	-	H_4	H_3, H_5, H_1 [$r \neq i$]	-	H_4
ZC	CA_3	-	v	-	-	-	H_4	H_7 [$r > v$]	H_7	H_4
ZC	CA_4	-	-	-	$i \in M$	-	H_3, H_5	H_3, H_5, H_6 [$r \neq i$]	-	H_6
TC	CA_5	-	-	-	-	False	-	H_1 [$r > 0$]	-	-
TC	CA_5	-	-	-	-	True	H_2	-	-	-

From the analysis of Table 2.5, we identify the following unsafe control actions for CA_1 in Table 2.6. We report the complete list of UCAs for control actions CA_1, CA_2, CA_4 and CA_5 in Chapter 7.

Table 2.6: Table of Unsafe Control Actions - CBTC

#	**Description**	**Hazard**
UCA_1	CA_1 *not provided* when *zone*[i] is *engaged*	H_4
UCA_2	CA_1 *provided* with *allow* when *zone*[i] is *engaged*	H_1
UCA_3	CA_1 *provided late* when *zone*[i] is *engaged*	H_4
UCA_4	CA_1 *not provided* when *zone*[i] is *free*	H_4
UCA_5	CA_1 *provided* with *deny* when *zone*[i] is *free*	H_2
UCA_6	CA_1 *provided* too late when *zone*[i] is *free*	H_4

Finally, the last step of STPA consists of the identification of *hazardous scenarios*, where accidents are possible and harm can be caused. In the case of STPA-Sec, we focus on the identification of *threat scenarios* instead; where an adversary provokes directly, or indirectly through cascading effects, the application of an unsafe control action. In both cases (STPA

and STPA-Sec), the operation is usually carried out manually, relying on expert knowledge. In our methodology, we use STRIDE to systematically identify threats against the safe control structure, which we use to synthesise an attacker model. Then, we employ model-checking to automatise the identification of threat scenarios (Chapter 5).

STPA does not aim to perform risk assessment or produce a quantitative output of any form mainly for two reasons. First, the objective of STPA is the identification of possible causes for accidents rather than their likelihood. Second, it is meant to be applied at the early design stage, where the likelihood of individual failure is unknown or, far worse [Lev15], highly uncertain [KLHP21]. The same argument is valid for STPA-Sec [KM21]. In addition, in the security realm, threats are *intentional* and not *accidental* anymore and reasoning about likelihood and probabilities becomes even more sensitive.

Integration with formal verification Although partially automated in [TI13], STPA essentially remains a manual process driven by expert knowledge. More specifically, identifying the root causes leading to the application of unsafe control actions (step 4) presents a number of challenges due to the inherent complexity of modern systems and the fact that components' behaviours might not be widely available or their formalisation might not be uniform across the entire system. In addition, STPA is often applied at the early stages of the system design process, and at this stage, some behaviours may not be available yet. Consequently, the results obtained by these analyses are highly subjective and lack reproducibility. Moreover, the methods employed lack a clear link to one another, thus limiting traceability between different analyses.

Several studies such as [AW15, AWL15] and [DV18] aim to build a correspondence between the STAMP model of the CPS and the behavioural model of its implementation. This mapping enables us to formulate hazardous control actions as Linear Temporal Logic (LTL) specification [Pnu77]. Safety specifications can then be formally verified to ensure that safety requirements are met. In particular, Abdulkhaleq et al. [AWL15] use this strategy to generate tests for safety-critical software. Similarly, system safety requirements (SSRs) are generated in [SQR+19].

Finally, Zhao et al. in [ZDLW21] construct specifications for an avionic system and verify them with the UPPAAL model checker[BDL+06b]. However, these studies do not consider security issues nor perform verification in the presence of an adversary.

2.8 Summary & Conclusions

In this chapter, we have provided an overview of the state of the art of methodologies for integrated safety and security analysis.

We have started by introducing the main terminology adopted in this thesis and the use-case that we are using as a run-through example in the rest of the thesis. We have discussed the relevant background in the field of safety analysis and threat modelling, respectively. Successively, we produced a review of the recent works at the intersection of security and safety. In particular, we have elaborated on their characteristics and their differences with each other.

Whilst our approach is novel, we have seen that there are works in literature that move steps in similar directions. We have explained the limitations of these works and have highlighted their main differences with the approach proposed in Cassandra.

Approaches stemming from STPA [YP17, FMS+17, DSCdMBH20, KM21] are characterised by an important manual component which makes them error-prone and produces results that depend on the expertise of the analysts and are not reproducible. With the exception of [FMS+17], scenarios produced by these approaches are defined at the functional level and do not directly map to the deployment context of the system. This is due to the fact that distinct systems views are generally analysed individually. Doing so hinders the identification of complex multistage attacks. With Cassandra , we also seek to identify complex attacks where malicious actions interleaved with system behaviours can cause harm.

Approaches proposed by [KBL19, MLS+21] also aim to identify safety-critical attacks but present limitations concerning the threat models, the safety formalism employed, and their capability of analysing cascading effects (Section 2.5.4).

The approach proposed by [AR13] is characterised by different intents from ours. In fact, the aim of [AR13] is to verify a design, including eventual security controls, against specific attacks, safety requirements and performances.

In this chapter, we have also provided a more in-depth description of Systems Theoretic Accident Model and Processes (STAMP), a system theoretic approach to modelling the safety of cyber-physical systems. Thus, we focused on System Theoretic Process Analysis (STPA), a process for safety analysis based on STAMP, and on STPA-Sec - an extension of STPA for integrated safety and security analysis. As an example, we have provided an off-the-shelf application of STPA/STPA-Sec to our CBTC use case.

In the next chapter, we present an overview of our approach and elaborate on the details of its structure.

Chapter 3

Cassandra: A Novel Methodology for Safety & Security Analysis

In the previous chapter, we have highlighted common issues associated with approaches traditionally employed for integrated safety and security analysis.

Approaches characterised by sound underlying safety models such as [YP17, FMS+17, KMM18, MSB+15a] are associated with a significant manual workload and are driven by expert knowledge. In this sense, they are time-consuming and require a significant amount of manual effort as well as highly specific knowledge across several teams of experts in safety and security (Chapter 2). As a result, their **outcomes** are **highly subjective** and **lack of reproducibility**.

Recent works such as [KBL19, MLS+21] have moved steps towards more automatic types of analysis. However, proposed solutions do not include sound safety models and work under relatively simple behavioural models and/or threat models (Chapter 2).

Another limitation of approaches traditionally adopted to perform integrated safety and security analysis is that they do not usually analyse distinct system representations (e.g., functional, behavioural, and architectural) in combination. This inevitably reflects in limited traceability across different types of analysis. For example, in [YP17, KMM18] threat modelling is carried on the STAMP model, but threats are not linked to their impact on components behaviour.

Such disconnection makes it exceptionally complex to identify exploitations of threats aimed at inducing cascading effects as well as safety-critical sequences of threats. Similarly, the lack of a clear connection between the functional layer of the CPS and its architecture, makes it difficult to separate attack vectors that are relevant to system safety from those that are not. The issue is tackled by [MSB+15a, FMS+17] which relate elements of safety models to components in the system architecture; however, they do not formally model the the impact of threats on system behaviour.

In this thesis, we propose a methodology, Cassandra, and a set of tools to systematically analyse the impact that attacks against safety-critical systems have on their high-level safety properties. The approach proposed is backed by STAMP and uses the STAMP control hierarchy to model functional interactions between system components. In particular, Cassandra looks at three distinct views - the functional, behavioural, and architectural representations - of the CPS to automatically identify threats leading to unsafe behaviours (threat scenarios) and the vulnerabilities that enable them. In particular, with Cassandra we propose the following combined analyses

On the **functional** representation of the CPS, we apply STRIDE to systematically model threats against interactions and components in the STAMP model of the cyber-physical system [DSCdMBH20]. While pure STPA-Sec does not provide guidelines on the nature of threats to consider during the analysis, employing STRIDE as a threat modelling tool ensures that considered threats are derived systematically starting from the control structure.

STAMP does not model the behaviour of the system components. Hence, to understand the impact of threats on emerging system properties, including safety, we build a set of relationships between the **STAMP** model of the CPS and a **behavioural representation** of its components (Chapter 4). The formal verification of safety properties allows Cassandra to systematically and automatically verify whether there exist combinations of STRIDE threats against the elements of the functional model that can lead to safety violations if they are applied when the CPS is in a particular state. The systematic

joint application of STPA-Sec, STRIDE and formal verification not only guarantees **reproducibility** of results but also allows us to introduce a degree of **automation** to enumerate threat scenarios (Chapter 5).

STRIDE threats are defined against the **functional view** of the CPS. Thus, we need to ground the analysis in the specific deployment context (**architectural** view) to map the output from the STRIDE analysis to the specific exploitation of vulnerabilities on system components. To address this gap, we relate threats to the exploitation of specific vulnerabilities in system components. This is done by identifying the privileges required to exploit threats. Finally, Cassandra relates these privileges to an attack graph representation to reason about the attack paths [JNO05, AJN12] leading to the exploitation of threats.

This chapter is organised as follows. We provide an overview of the processes involved in our approach in Section 3.1. In Section 3.2 we elaborate on the models employed to represent distinct views of the system and how they relate to each other. In Section 3.3, we discuss the main components designed for Cassandra to analyse system models. In Section 3.4, we show an overview of the applications of our approach featured in this thesis. Finally, Section 3.5 gives the conclusions of the chapter.

3.1 Overview of Cassandra

For a clearer presentation, we have structured Cassandra in three macro stages shown in Figure 3.1. The first stage (❶) is a process of discovery and enumeration of safety-critical threat



Figure 3.1: Cassandra - Main stages

scenarios, ordered sequences of threats leading to catastrophic consequences. The process

is grounded in STPA-Sec, which is an acceptable means of compliance for different classes of safety-critical systems - including aviation [EUR18] - and uses STRIDE to systematically derive threats against the STAMP model. Although highly effective in highlighting control dependencies among system components, STPA-Sec is a manual process and cannot perform the automatic analysis of the cascading effects leading to the application of unsafe control actions. This is because STAMP does not formalise the behaviour of system components and without it, we cannot infer the impact of malicious actions on the emerging behaviour of the CPS. We overcome this aspect by encoding the behavioural model of system components in a Network of Timed Automata (NTA) [AD90] (Chapter 4), and use model checking to automatically discover and enumerate threat scenarios (Chapter 5). The output of the process is the *threat scenarios tree*, a tree data structure that explains how high-level threats combined lead to hazards. Being based on STPA-Sec, STRIDE and formal verification, the discovery of threat scenarios is a **systematic** operation which we have successfully **automatised**. As such, the process is **objective** and **reproducible**, thus overcoming the first main limitation of traditional integrated approaches.

The threat discovery and enumeration process looks at the functional model of the CPS which does not include architectural details. In the second stage (❷) of Cassandra, we ground the analysis in the specific deployment context to map safety-critical threat scenarios to the specific exploitation of vulnerabilities on system components. We rely on requirements traceability, usually adopted during the development life-cycle of critical systems, to derive safety-critical attack paths, sequences of privileges that the adversary needs to obtain to execute safety-critical threat scenarios, hence cause harm (Chapter 6). These paths are represented using attack graphs. Cassandra supports different representations of attack graphs, which we adopt in different use cases according to the type of analyses that we intend to perform in later stages. In particular, we have employed logical representations of attack graphs [JNO05], the tripartite logical representation proposed by MulVal [OGA+05], and the Bayesian representation [LM05]. Overall, this stage is critical to understanding the relevance of individual vulnerabilities (e.g., a buffer overflow on a server program) with respect to more abstract system-level safety properties.

Safety-critical attack paths are then analysed in the third stage of Cassandra both qualitatively and quantitatively to understand the risk of safety-critical threats and to evaluate the effectiveness of security measures in reducing the likelihood of safety-critical attacks (❸). The qualitative analysis of the attack graph shows the feasibility of the threat scenarios found. It also provides insights into the weaknesses and vulnerabilities present in the system, which we use to prioritise the application of security controls. On the other hand, trading off usability and performance against security gains requires a quantitative analysis of the attack graphs produced. We do so by reasoning over the probabilities using Bayesian inference in the third step of Cassandra.

Table 3.1: Main formalism adopted in Cassandra

Stage	Methodologies	Underlying Model(s)
	System Theoretic Process Analysis Security (STPA-Sec) [LT18, YP17]	STAMP [Lev16]
Threat Scenarios Discovery and Enumeration ❶	STRIDE (Threat Model) [Sho14a]	STAMP [Lev16]
	Formal Verification (Model Checking)	Network of Timed Automata [AD90]
Derivation of safety-critical Attack Paths ❷	Formal Verification (SAT)	Logical Attack Graphs [AWK02, OGA+05] Bayesian Attack Graphs [LM05]
	Network Reachability	
Path Analysis ❸	SAT Graph Reachability Analysis Bayesian Inference	Logical Attack Graphs Logical Attack Graphs Bayesian Attack Graphs

In Table 3.1 we show a summary of the methodologies and formalism adopted at each stage of Cassandra.

3.2 An integrated multi-layer approach

We have observed how traditional approaches to safety and security analysis consider representation (e.g., functional, behavioural, architecture, etc.) individually and lack the capabilities of performing integrated analyses e.g., adoption of threats discovered during the threat modelling of the functional model to uncover unsafe behaviours or use threat scenarios as starting point

to uncover attack paths. In contrast, our methodology, integrates for the first time functional, behavioural, and architectural views in a comprehensive analysis which highlights the interplay among them.

We outline the *three* views of the cyber-physical system analysed by Cassandra in Figure 3.2. The functional model captures the main components of the system and the information flow (data) between them. Given its main focus on safety, Cassandra relies on the STAMP control hierarchy [Lev16] to model the functional aspects of the CPS. In fact, being grounded in control theory, STAMP is the optimal candidate to study the propagation of the impact of attacks through the system (Chapter 2). In particular, the STAMP control hierarchy allows to model the functional components such as controllers, sensors, actuators, and physical processes, as well as the information they exchange with one another.

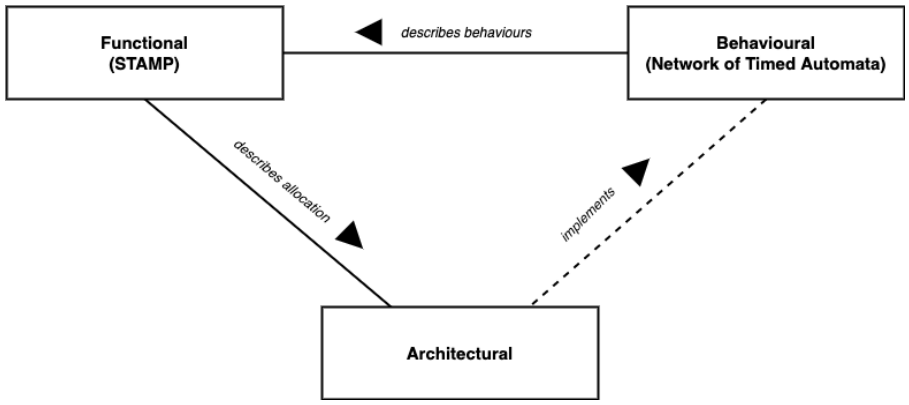


Figure 3.2: Representations of the cyber-physical system considered in Cassandra.

The integration of the STAMP model with a model of the behaviours of its components empowers Cassandra with the capability of analysing and formally verifying cascading effects and, in particular, whether certain malicious actions, caused by an adversary and interleaved with the system behaviour, ultimately lead to catastrophic consequences.

However, in order to use such formal verification, we need a) to build a formal correspondence between the STAMP control hierarchy and the model of components' behaviour. b) to translate safety properties (i.e., the properties that must be preserved during system operation for the system to operate safely) into their formal expressions that can be verified. We adopt Network of Timed Automata (NTA) [AD90] as a formalism to model the collective behaviour of system

components and their interactions (Chapter 2). At the same time, we translate STPA unsafe control actions - which we derive through the off-the-shelf application of the first three steps of STPA - into their formal representation (Chapter 4) using Timed-based Computation Tree Logic (TCTL).

With the term *architectural view*, we refer to the physical architecture which represents the specific deployment context of the cyber-physical system [BM16]. It covers the concrete components such as hosts, networking components, controllers, sensors, and actuators which together all contribute to the provisions of system functions. Elements of the functional model are *allocated* to concrete components of the CPS architecture (Chapter 6) the latter *produce*, *transport*, and *consume* the information flow.

Figure 3.3 shows how our methodology also relies on a behavioural representation of the attacker based on the formalism of Timed Automata [ACD90]. We have mentioned that we employ STRIDE [Sho14b] to model threats at the functional level. The result of the threat modelling is a set of adversarial actions, threats against elements of the STAMP model, that would enable the adversary to access, disrupt or alter them. By synthesising these adversarial actions into an aggregated model we compose the model of attacker behaviour. Cassandra verifies the reachability of safety properties in adversarial conditions, when the model of attacker behaviour interacts with the behaviour of the CPS. Every time that a violation is found reachable, a threat scenario is discovered (Chapter 5). We reiterate the model checking process multiple

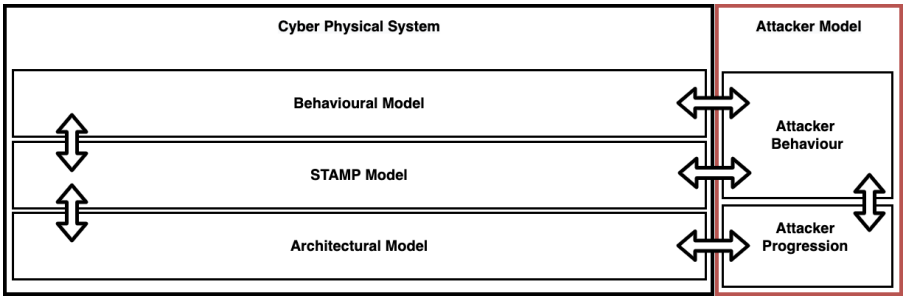


Figure 3.3: Integrated view of the cyber-physical system. Arrows show our mapping across models.

times to enumerate reachable threat scenarios, where sequences of threats can ultimately cause harm. The proposed integration between STAMP and behavioural models of the CPS allows

Cassandra to effectively automatically enumerate threat scenarios, thereby addressing the first shortcoming identified in the related works.

The STAMP control hierarchy of the cyber-physical system is a functional model and does not include deployment details such as network topology, hardware and software configurations, etc. To study the feasibility of threat scenarios, understand their associated risk, and identify appropriate security controls, we need to ground the analysis in the specific architecture context. To this extent, we map high-level threat scenarios identified through the application of model checking to the specific exploitation of vulnerabilities in system components. Such mapping allows the unveiling of attack paths that are critical to system safety and the study of their feasibility as well as the associated risk (Chapter 6). The attack paths are part of our attacker progression model (Figure 3.3). By using them, we are able to comprehend the importance of individual vulnerabilities and weaknesses with respect to the preservation of system-level safety properties and prioritise the application of security controls.

3.2.1 The functional model at the centre

We have highlighted how our approach relies on the integration of *three* models (or views) of the cyber-physical system to achieve its intended objectives. These are the functional, behavioural, and architectural views. In the literature, they are analysed separately, with no or minimal relationships established among them. In order to carry out an integrated analysis, we must, therefore, establish a comprehensive system domain model to be able to combine them. The system domain model [Fow12] behind our proposed methodology is shown in the class diagram in Figure 3.4. It gives an overview of the relations that we have established across the views (models) of a cyber-physical system. In the diagram, we have introduced two relationship stereotypes, namely *map* and *verify*. The first explains the existence of a mapping between two classes, whereas *verify* represents that, following the direction of the relationship, elements of the first class are formally verified on those of the second class, for example, a formal property (e.g., Timed Computational Tree Logic (TCTL) property) can be verified on a Network of Timed Automata. We will elaborate on how these mappings across models are established

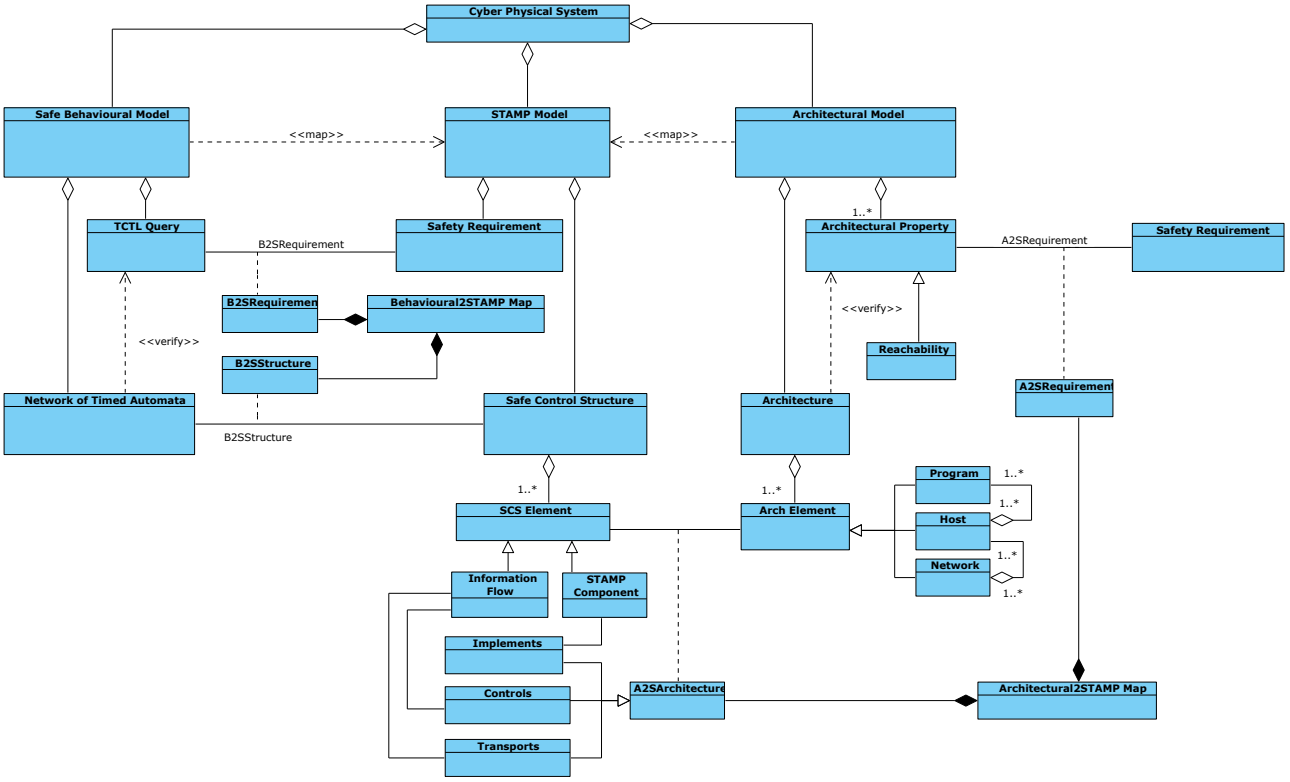


Figure 3.4: Integration of views of cyber-physical systems

(Chapter 4). In previous sections, we have described how STAMP allows us to derive the **safety requirements** of the system as well as the control hierarchy (**safe control structure**) (Section 2.6). In Section 4.1, we elaborate on our approaches to the safe behavioural model and establish two sets of relations; one maps the safe control structure to the behavioural model while the second allows us to translate the safety requirements into properties (TCTL Queries) that we can formally verify.

The architectural model represents the **architecture** and a set of **architectural properties**. We represent the **architecture** as a directed graph $\mathcal{G}(V, E)$ with vertex set V and edge set E. V is a collection of hosts such as workstations, servers, controllers, sensors and actuators. Let $v_i \in V$ and $v \in V$ be two host nodes, thus the edge $(v_i, v_j) \in E_H$ represents a network connection between two hosts. We introduce **architectural elements** to identify individual hosts (vertices) and connections between hosts (edge). These can also specify attributes that are related to an individual host (e.g., run software, permissions, etc.) or connection (e.g., protocol, etc.). On the other hand, we specify reachability properties, which represent global aspects of the architecture, such as reachability between two vertices in V.

Finally, Figure 3.4 also emphasises how STAMP plays a central role in the application of Cassandra. In fact, on one hand, we map the STAMP model of the cyber-physical system to its behavioural representation to analyse the propagation of cascading effects of attacks leading to hazardous scenarios. On the other hand, we map the same STAMP model to the physical architecture of the CPS. The latter correspondence is critical to identify the *attack paths* leading to hazardous scenarios as well as to model the application of security controls on safety.

3.2.2 Model of architecture

We map the output from the STRIDE analysis to the specific exploitation of vulnerabilities in system components. One of the points of strength of our methodology is that we keep the control model (STAMP) separate from the model of the architecture of the CPS, also referred to as *architectural model*. The control model is usually simpler than the architectural model and focuses on the control dependencies. The architectural model, on the other hand, shows how system components are connected with each other in computer networks. We consider that the latter also includes the IT infrastructure that, although not directly involved in the control of the system, can be an attack vector. The architectural model is used, in Chapter 5 to identify attack paths leading to hazards.

3.3 Structure of the methodology

Earlier in this chapter we mentioned how we have structured our proposed methodology in *three* distinct macro-processes; namely, *threat scenarios discovery and enumeration*, *derivation of safety-critical attack paths*, and *risk analysis and identification of security controls* (Figure 3.1).

Cassandra is designed to be modular, with each *process* being composed by one or more *sub-processes*. Although processes and sub-processes use each other's output, we have designed them as independent modules, each providing specific functionality for the integrated safety

and security analysis. In Figure 3.5 we use the UML class diagram notation to show the high-level overview of the structure of Cassandra. In particular, the methodology is defined as the

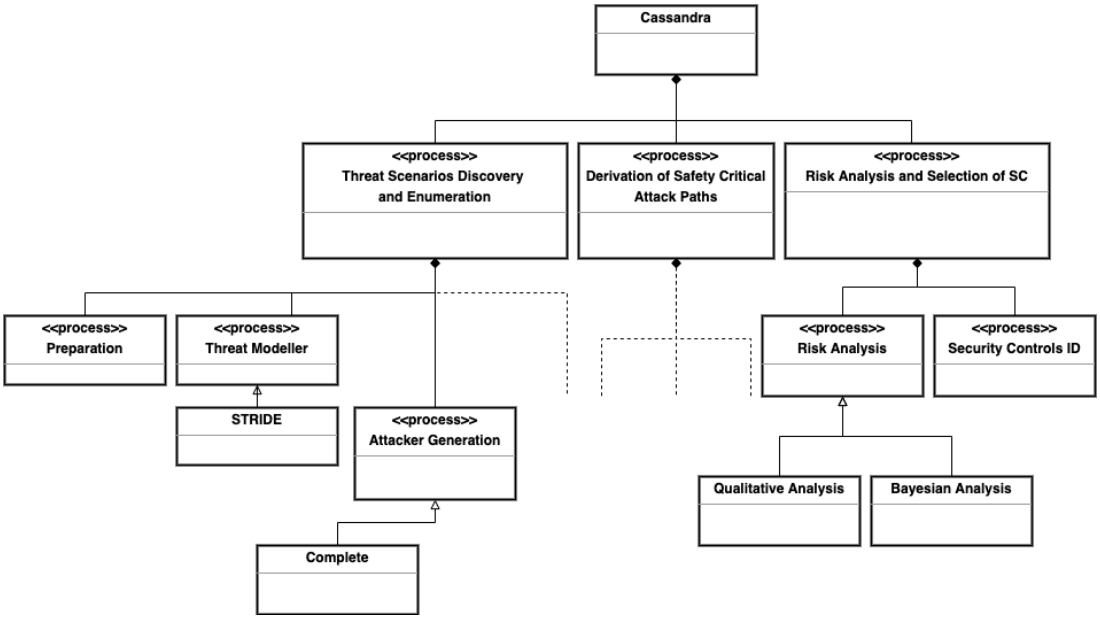


Figure 3.5: High Level Overview of the Structure of Cassandra.

aggregation of its *three* main *macro-processes*. These, in turn, are also defined as aggregations of the processes that compose them. The usage of inheritance relationships enables us to provide multiple distinct implementations for high-level functionalities. For example, for the risk analysis, we have implemented two processes: qualitative and quantitative analysis. Each implementation employs a different approach to the analysis of the AG. Similarly, processes can be aggregated and composed together to realise complex functionalities. Such modular structure allows for easily extending the scope of the methodology by adding additional processes, e.g., using a different technique for the identification of threat scenarios or risk analysis, or changing the internal logic of those that already exist without having to reconfigure the whole methodology. For simplicity, we decided to not include in Figure 3.5 details regarding the structure of all processes at this stage. Details about the specific structure of individual processes of the methodology follow in the next chapters.

We also propose a toolchain that implements our proposed methodology. In this thesis, we provide a programmatic implementation of only a subset of the processes while others are defined, but their execution is carried out in a non-programmatic fashion. For each process, we clearly state, when providing its description, if we have also worked on its programmatic

implementation in Cassandra.

It must be noted that the applicability of the first step of Cassandra (identification and enumeration of threat scenarios) requires that the *three* views of the cyber-physical system (Figure 3.3) are available for the analysis and their mappings have been defined. We acknowledge that this requirement might not be fulfilled in certain circumstances and, for this reason, we considered an additional *preparation* stage, to be executed before the application of Cassandra. In this stage, we perform a preliminary application of the first three steps of STPA (i.e., the definition of hazards, control hierarchy, and identification of unsafe control actions), and formally define the relationships between system's views where needed. In Figure 3.5, we represent the *preparation* as part of Cassandra's first step.

In the paragraphs below, we provide a summary description of the main processes in our proposed methodology.

Preparation We have anticipated how the application of Cassandra requires that an integrated view of the cyber-physical system combining safety (functional), architectural and behavioural models is available. This is because the first step of Cassandra (identification and enumeration of threat scenarios) relies on the following items:

P.1 The functional representation of the system is a STAMP model.

P.2 The model of behaviours of system components is a network of timed automata.

P.3 Part of the safety analysis has already been carried out using System Theoretic Process Analysis (STPA). We assume that the first three steps of STPA have been applied to the STAMP model of the system and that unsafe control actions are known.

P.4 There exists a formal map between STAMP and behavioural representations.

P.5 Threats against elements of the STAMP model have been modelled (threat model is available).

P.6 There exists a formal map between threats and privileges in the CPS architecture.

P.7 Vulnerabilities and weaknesses that affect the CPS architecture are known.

The *preparation* stage precedes the application of our methodology and aims to align the models and *prepare* an integrated view of the CPS by addressing the key points above.

We have mentioned that we use System Theoretic Accident Models and Processes (STAMP) to describe the safety aspects of the CPS ($P.1$). The process requires knowledge of the structure of the system to derive the control structure as well as knowledge of the operational domains to draft failure modes and hazards [Lev16]. Both activities demand a significant knowledge of system components as well as design capabilities and, for this reason, cannot be entirely automated (Chapter 2). In ($P.2$) we assume that a model of behaviours for system components is specified manually. Although there is related work such as [CMT18, LAVM18, MBMS21], which suggests such models could be derived automatically from the system implementation, how to do so is outside of the scope of Cassandra.

More in general, we start from the assumption that STAMP, *behavioural*, and *architectural* views of the CPS are known, with some degree of certainty when we start the preparation phase. In real applications, these models would be provided to security and safety engineers by the respective design teams throughout the development life-cycle of the CPS.

We perform the first three steps of off-the-shelf (OTS) STPA to derive unsafe control actions, therefore addressing $P.3$.

In order to be able to use model-checking to verify the reachability of unsafe control actions in the *identification of threat scenarios*, we need to construct a formal correspondence between STAMP and behavioural model (Chapter 4), hence ($P.4$). Such correspondence is the first main contribution of this thesis, as it enables to automate the safety verification with respect to security threats.

We employ STRIDE to model threats against the STAMP model of the CPS ($P.5$). Authors of STPA-Sec do not rely on any standard threat modelling techniques, hence we employ STRIDE [Sho14a] as it is systematic, and its use is widely accepted both within industry and academia.

Finally, we assume that the physical architecture of the cyber-physical system is known and available for analysis. We construct an additional mapping between the architectural view and STAMP control hierarchy to establish which elements of the architecture are responsible for functional elements (Chapter 6). We leverage this mapping to identify relations between privileges and threats (*P*.6). Vulnerabilities and weaknesses in the physical architecture are typically known. Vulnerability discovery can be achieved, e.g., through tools such as Nmap[Lyo08], Nessus [Ten], Openvas [Gre], or other procedures (*P*.7).

Threat scenarios discovery and enumeration The application of STRIDE on the STAMP control hierarchy (safe control structure) produces a list of instances of threats affecting the elements of the CPS, which we call the attack library. During the threat scenarios discovery and enumeration process, we use threats in the attack library along with our knowledge of system behaviours to build a timed automaton which provides a synthetic representation of the attacker behaviour (Figure 3.6). The TA is built through a *Attacker Generation* process

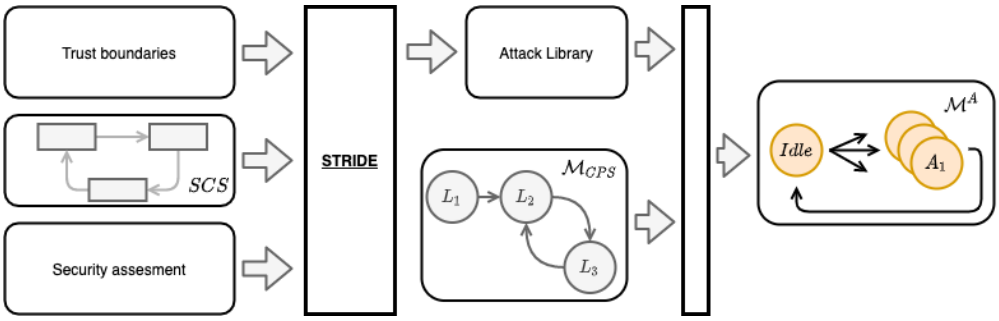


Figure 3.6: Derivation of the model of adversarial actions.

(Figure 3.5). We discuss a *complete* approach to the *attacker generation*, which uses all the actions in the attack library to build the attacker TA. This approach is complete as we assume that any of the attack actions can be performed at any time. Thus, this is a superset of all possible attacker behaviours identified through threat analysis. More *efficient* approaches can also be considered, which use the attack graph of the CPS and only consider threats in the attack library that are feasible in a fixed deployment context.

The *instrumented model* of the cyber-physical system then combines the model of the attacker, with that of components behaviours. It is produced through an instrumentation process (Figure

3.7). We define a systematic approach to the instrumentation process (Chapter 5). We leverage

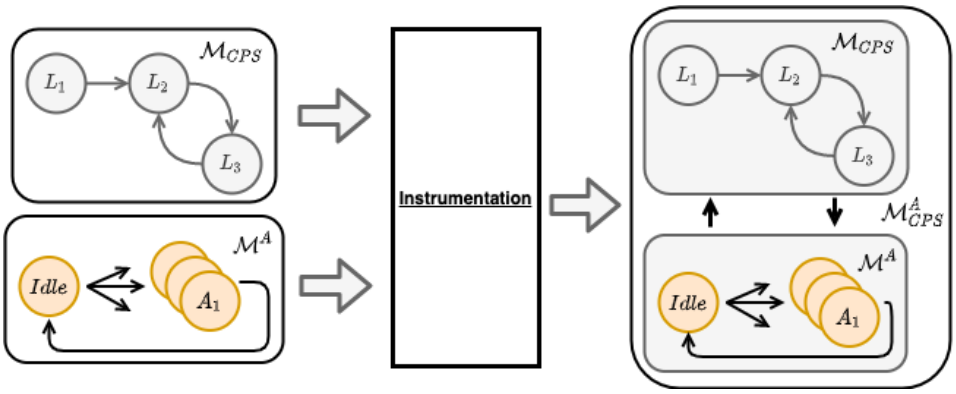


Figure 3.7: Process of adversarial instrumentation

symbolic model checking in conjunction with custom algorithms to enumerate threat scenarios affecting the cyber-physical system, given a model of the behaviour of the adversary.

Attack Graph Generation and Path Analysis The second key aspect of our proposed methodology is the integration with the architecture of the system to analyse the risk posed by adversarial threats under the current configuration of the CPS. The risk can be computed in this process using different approaches, including probabilistic model checking and attack graph analysis using both qualitative and quantitative (Bayesian) inference techniques (Chapter 5). This capability is critical in Cassandra not only to evaluate the effectiveness of suggested security measures in reducing the incidence of threat scenarios but also to build assurance cases for system designs and architectures. In fact, depending on the context where the CPS operates and on the nature of its functionality, certifying organisations will require that the overall risk with respect to certain hazards is lower than predefined thresholds. We elaborate on the possibility of using our proposed methodology for system assurance cases (Chapter 9). Although our methodology is sufficiently general to accommodate a variety of ways of calculating risk, in this thesis, we focus on two types of analyses:

Qualitative risk analysis: Given a threat scenario (TS), we perform a first assessment of its reachability on the respective attack graph.

Quantitative Risk Analysis using Bayesian Inference: Given a threat scenario, and

its related attack paths on the system architecture, our proposed methodology employs Bayesian inference to assess the unconditional probability that an attacker would be able to execute said safety-critical sequences of threats [MGSBL17].

Another key aspect of Cassandra is the capability of identifying security measures that should be prioritised to prevent threat scenarios and/or reduce the overall risk. We define and solve a Boolean satisfiability (SAT) problem to find the minimum set of privileges that should be protected to prevent the attacker from enacting the threat scenarios. In particular, given a threat scenario and its related attack graph, Cassandra derives a complete list of *minimal safety sets* (MSS), where a minimal safety set is the minimal set of privileges whose protection needs to be ensured to avert safety-critical attacks. We leverage the integration with the architecture of the CPS to construct, starting from the set of privileges, a system-wide attack graph. The resulting graph encompasses feasible attack paths that are relevant to system safety. Finally, we analyse the attack paths found and elaborate on mitigation strategies. More broadly, the attack paths found reveal the weight of individual vulnerabilities and weaknesses with respect to the preservation of system-level safety properties. Cassandra leverages an approach based on SAT to identify appropriate mitigation.

3.4 Practical Examples of Applications

In order to demonstrate the applicability of our methodology, we apply it in *three* different case studies, to *three* different CPS, and across *three* different industry sectors. We have chosen to do this because the use cases have different requirements, characteristics and contexts of application, thus showing the versatility of the approach and its usefulness across a broad range of applications.

Details regarding the analyses of the use cases follow in the paragraphs below. In Figure 3.8, we also outline the applications of the processes involved in Cassandra. Within each process, we use colour-coded labels to indicate the scenarios where we have used them. ERTMS and EPIC labels, respectively, indicate the control-based train control of the electric power system,

while FMS refers to the flight management system. In Table 3.2, we provide an overview of the types of analyses that we have carried out with our methodology for each analysed scenario.

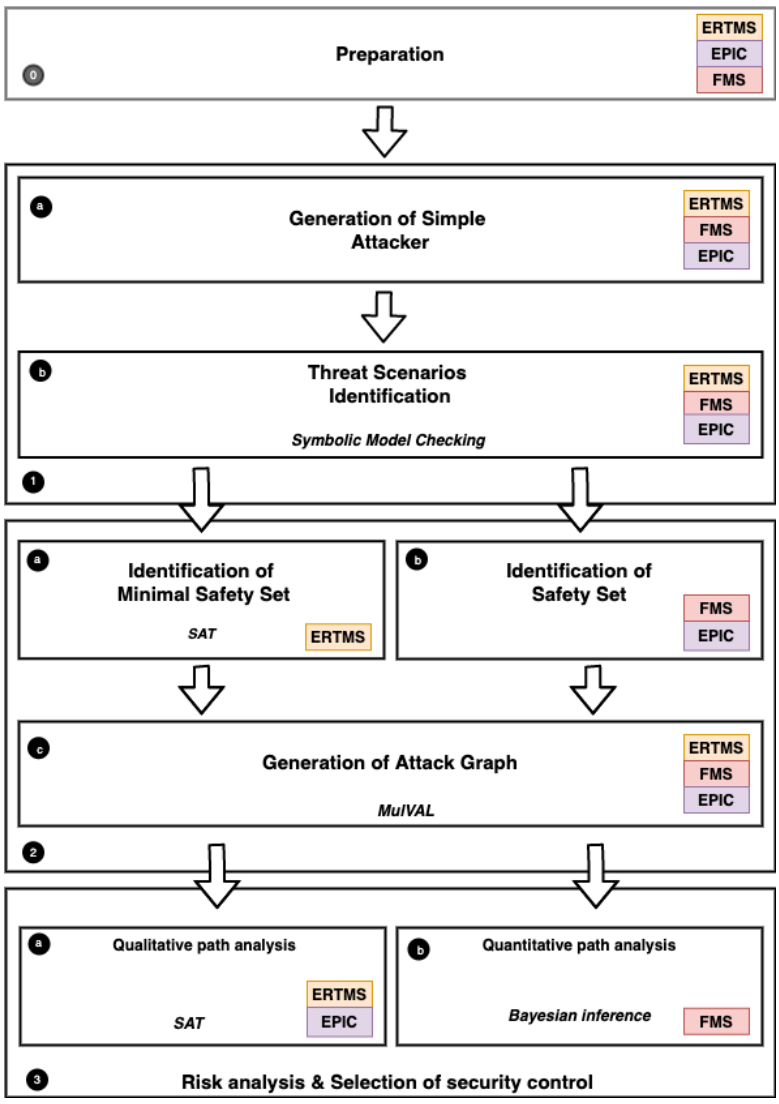


Figure 3.8: Processes involved in the methodology

Communication Based Train Control Our first use case consists of a Communication-Based Train Control System (CBTC), which we have designed following the specifications of the European Railway Traffic Management System (ERTMS) made available from the European Union [Uni]. In Chapter 2, we demonstrated the application of the first three steps of STPA in this use case. We employ the CBTC system as a run-through example throughout the thesis to demonstrate the applications of individual parts of Cassandra. In Chapter 7, we recap and expand this use case in a single and coherent analysis.

In Figure 3.8, we show the Cassandra processes that we employed in this use case. Following the preparation phase, we generate a *synthetic attacker* (1.*a*) for the system in order to perform *threat scenarios identification and enumeration* (1.*b*). The operation leverages symbolic model checking to enumerate threat scenarios. Thus, we identify privileges in the minimal safety set (MSS) (2.*a*) and generate the attack graph leading to privileges in the MSS (2.*c*). We perform a first *qualitative* evaluation of the risk (reachability). Thus, we identify security measures to apply to prevent adversaries from causing a hazard (3.*a*), i.e., effective in preventing the reachability of threat scenarios. Finally, we discuss the limitations of this approach owing to the complexity of the scenario (Section 7.3).

Electric Power Grid Although designed on the basis of ERTMS specifications, the CBTC use case reflects a fictional scenario which might not fully capture the complexity of a real deployment. To this extent, our second use case scenario (Chapter 8) is an electric power grid system, whose design reflects the architecture of the Electric Power and Intelligent Control (EPIC), an Electric Management System (EMS) testbed deployed in Singapore [AKM18]. Through the analysis of this use case, we show that only Cassandra is applicable to a realistic system and that it can find the expected threat scenarios. We start with the generation of a *synthetic attacker* (1.*a*) for the system in order to perform *threat scenarios identification and enumeration* (1.*b*). The operation leverages symbolic model checking to enumerate threat scenarios. We elaborate on the limitation of this approach due to the complexity of the scenario. We identify the privileges linked to the execution of all safety-critical threats (2.*b*), thus starting from the threat scenarios previously generated (1.*b*), we leverage the integration with the system architecture to generate the attack graph encompassing safety-critical attack paths (2.*c*). Thus, we analyse the attack paths to identify weak points in the OT network of EPIC (Section 8.4). Our findings align with [AKM18].

Flight Management System As part of the EU H2020 project Concordia, we have developed an additional use case in collaboration with Airbus. The aim of the initiative is to demonstrate the applicability of Cassandra to support assurance cases. In particular, we

analyse the safe operations of a Flight Management System, when an adversary tries to gain unauthorised access to aircraft avionics. With the help of Cassandra, we are able to identify security controls that, once applied, would bring the likelihood of a threat scenario below acceptable thresholds. The case is outlined in Chapter 9. Once again, we start with the generation a *complete attacker* (1.*a*) for the system in order to perform *threat scenarios identification and enumeration* (1.*b*) Section 9.2. The operation leverages symbolic model checking to enumerate the threat scenarios.

Starting from the threat scenarios generated, we leverage the integration with the system architecture to generate the attack graph showing the safety-critical attack paths (Section 9.3). We elaborate on our algorithm to derive a Bayesian network from the generated attack graph and apply Bayesian inference to identify the unconditional probability of an attacker causing a hazard. This risk analysis constitutes a baseline for the later application and evaluation of security measures. Finally, we identify security controls to apply in order to reduce the unconditional probability of an attacker causing a hazard. We repeat the risk analysis (3) in different scenarios to evaluate the assurance case (Section 9.3).

Table 3.2: Processes of Cassandra involved in the analysis of safety-critical scenarios.

Process	Use Case		
	ERTMS	EPIC	FMS
Identification of Threat Scenarios - Complete (Symbolic MC)	✓	✓	✓
Risk Analysis (Qualitative)	✓	✓	✓
Risk Analysis (Bayesian Inference)			✓
Identification of Safety Set	✓	✓	✓
Identification Minimal Safety Set	✓		
Attack graph to privileges in Safety Set	✓	✓	✓
Attack graph to privileges in Minimal Safety Set	✓		
Evaluation of Security Controls (quantitative)			✓

3.5 Summary & Conclusions

In this chapter, we have elaborated on the high level structure of our proposed methodology. It consists of *three* macro-processes that we apply, in sequence, to carry out the integrated analysis of the safety-critical system. First, we start with the joint analysis of *functional*

and *behavioural* view of the CPS to identify and enumerate threat scenarios, i.e., sequences of applications of threats that adversaries can leverage to cause harm. Then, we perform the joint study of *functional* and *physical architectural* layers to identify the attack paths leading malicious actors to the exploitation of found threat scenarios. Finally, we perform a qualitative and quantitative risk analysis of the attack paths to estimate the likelihood of threat scenarios and identify applicable security controls. Such integrated analyses are supported by a comprehensive system domain model, which we have also introduced in this chapter.

In the next chapter, we outline a set of novel formal relationships that we have introduced between the functional and behavioural views of the cyber-physical system to be able to apply formal verification in the first stage of Cassandra.

Chapter 4

Cascading Effects

In Chapter 3, we have given a high-level overview of Cassandra, our proposed methodology for integrated safety and security analysis. In particular, we have highlighted the key processes composing it and their internal structure. In this chapter, we provide a detailed description of the model used by Cassandra to represent the behavioural aspects of the cyber-physical system and its relations to the STAMP control hierarchy and STPA unsafe control actions.

One of the main objectives of Cassandra is to perform a systematic and semi-automatic analysis of the impact that attacks have on the *safety properties* of the CPS. In Section 2.6, we have provided an outline of System Theoretic Accident Models and Processes (STAMP) which Cassandra employs to model the safety aspects of the cyber-physical system. By performing System Theoretic Process Analysis (STPA) and its variant for security, STPA-Sec, on the STAMP model, we are able to reason about the dynamics of safety-critical attacks. Although highly effective in highlighting control dependencies within the CPS, STPA/STPA-Sec alone cannot *systematically* determine whether the actions performed by an adversary would lead to hazardous scenarios. The reason for this limitation lies in the fact that the STAMP model does not formalise the behaviour of system components.

In this chapter, we discuss the possibility of integrating STPA with model checking to overcome this shortcoming and achieve an *automatic* evaluation of cascading effects. In particular, we elaborate on two distinct approaches that we have developed. The first approach leverages a

complete description of the behaviour of system components to evaluate how perturbations on their input affect their output. The second approach, on the other hand, leverages a lighter propagation model which only describes functional input/output relationships expressed in the STAMP model.

This chapter is organised as follows. We introduce the notion of a safe behavioural model that maps to STAMP in Section 4.1. In Sections 4.2 and 4.3), we introduce our two approaches to model system behaviours. Finally, Section 4.4 features the conclusions of the chapter.

4.1 Safe behavioural model

In conventional STPA (and STPA-Sec), the identification of threat scenarios is typically done manually relying on expert knowledge. In particular, the *expert* needs to know how changes to the input affect the output for each component and trace these changes across the entire system. This is an exceptionally complex operation which requires a significant amount of effort and highly specific knowledge across several teams of experts in safety and security. The manual nature of this process constitutes a bottleneck in the application of STPA/STPA-Sec, and the results obtained by these analyses are highly subjective and lack reproducibility.

Previous studies have successfully integrated STPA with formal verification and model checking to verify the reachability of hazardous scenarios [ALS13, AW15, DV18] in non-adversarial conditions. In particular, [AW15] uses STPA output as a source to formalise safety specifications in Linear Temporal Logic (LTL) and employs model checking to verify the safe behaviour of the CPS. This approach is refined further in [AW16], which uses model-checking to generate test cases. Authors of [AW16] introduce a *safe behavioural model* (SBM), which models the aggregated behaviour of CPS components. However, the SBM in [AW16] does not consider the presence of an adversary and adversarial actions that may be interleaved with the correct operation of the system. In contrast, Cassandra also synthesises a formal model of the attacker behaviour derived from the threat analysis and uses model-checking to evaluate its impact on the system safety and verify the preservation of the safety properties. In fact, the

SBM introduced with Cassandra allows for the integration with a behavioural representation of an adversary who can access and modify information in the CPS. We call the result of such integration the *instrumented model* of the cyber-physical system [HCMB18]. The verification of safety properties against the instrumented model allows us to automatically find (and enumerate) threat scenarios. Moreover, we employ an approach based on Time-based Computational Tree Logic (TCTL), instead of linear logic, to verify the reachability of unsafe states. While the relative benefits of branching logic and linear logic have been debated in the literature [Var01], our choice of using TCTL is linked to the tool used for verification (Chapter 5).

4.1.1 Formalisation

We define the safe behavioural model (SBM) of the CPS as a network of timed automata (NTA). Semantically, an NTA is a labelled transition system (LTS) $\mathcal{N} = (T, Ch, B)$ where T is the set of timed automata, B a set of data buffers and Ch a set of synchronisation channels [BY03]. Automata in T synchronise with each other over synchronisation channels in Ch and exchange data through global access variables in B. The semantics of the NTA and of the timed automata used in this thesis are derived from those defined in [ACD90] and [BY03, BDL06a].

A timed automaton (TA) is a tuple $(\Sigma, L, l_0, \mathcal{V}, \mathcal{X}, E, I)$ where:

- $\Sigma = \Sigma_i \cup \Sigma_o$, with Σ_i and Σ_o being the input and output alphabet, respectively.
- L is the set of locations and l_0 is the initial location.
- $\mathcal{X}$ is the set of local clock variables.
- $\mathcal{V}$ is the set of local variables.
- E is the set of edges. The edge $e = (l, l', \sigma, o, \lambda, \delta) \in E$ connects two locations l, l' in the timed automaton. The transition $(l \rightarrow l')$ is taken on the input word $\sigma \in \Sigma_i$, if the current location of the automaton is l and the guard condition δ is met. Contextually, λ represents the updates that are applied to local (and global) variables during the transition. The automaton produces an output word $o \in \Sigma_o$ on the transition.

- I is the set of invariants, which restrict the number of locations that can be visited in function of clock values.

Authors of [BDL06a] show that a timed automaton behaves as a LTS $(S, s_0, \rightarrow)$ with set of states S, initial state s_0 and transition set $(\rightarrow)$, where the state of the LTS is defined as $s = (l, x)$, where with l and x being location and clock of the TA.

In our model, we allow the following types of synchronisations and synchronisation channels defined in [BDL06a, DLL+15]. Let a NTA be composed by the automata T_0 and T_1 with edge sets, respectively, E_0 and E_1. Let $c \in Ch$ be a synchronisation channel in the NTA. The edges $e_0 \in E_0$ and $e_1 \in E_1$ are synchronised on c if and only if, when T_0 completes the transition described by e_0, then T_1 enters the transition described by e_1. In this case, we say that e_0 is synchronised on *send* on synchronisation channel c and describe such synchronisation as $c!$. Similarly, the transition defined by $e_1 \in E_1$ is said synchronised on *receive* and labelled with $c?$ [DLL+15]. The NTA formalism that we employ conceives two types of synchronisation channels: *normal* and *broadcast* which are described in [BDL+06b]. Briefly, if $e_0 \in E_0$ and $e_1 \in E_1$ are synchronised over a *normal* channel, T_0 can undertake the transition described by e_0 if and only if T_1 is in a state where the transition described by e_1 is allowed. It is not possible to have more than one transition synchronised on *receive* on a *normal* synchronisation channel. On the other hand, if $e_0 \in E_0$ and $e_1 \in E_1$ are synchronised over a *broadcast* channel, T_0 is allowed to undertake the transition described by e_0 even if T_1 cannot follow with the transition described by e_1. In this case, the latter will not be executed. It is possible to have more than one transition synchronised on *receive* on a *broadcast* synchronisation channel.

4.1.2 Approaches to modelling cascading effects

We elaborate on two different approaches to modelling the propagation of cascading effects through the CPS. The first approach aims to explicitly model the dynamics of controllers and physical processes that compose the cyber-physical system. Here, the correctness as well as the completeness of the safe behavioural model are limited to the information known regarding the

design of the CPS when Cassandra is applied. Nevertheless, if Cassandra is applied as part of the design process, this model can be refined as further details are defined. In this approach, also referred to as *full-knowledge*, the internal structure of controllers in the SBM (e.g., number of states, process model variables, inputs, outputs, etc.) needs to be defined explicitly, starting from the control algorithm and process models previously defined during the STAMP analysis. In fact, in this approach, we need to design an SBM that has a precise correspondence with the STAMP model. This is needed in order to later proceed with the formal verification of the reachability of unsafe states. Such full-knowledge approach to the SBM presents two important advantages:

- It enables to verify the reachability of unsafe control actions while also taking into account the context where the system is operating. This is particularly important as the safety of a control action depends on the context of its application in the overwhelming majority of the cases (e.g., it is safe to open the door of a train *when it has reached the platform and it is not moving*, and it is not safe to open the door of a train *while it is travelling*).
- It does not require the safety/security analyst to manually specify, for each component, how alterations on the input propagate to its output. By including the dynamics of controllers and physical processes, in this approach, the effects of alterations induced by adversaries are intrinsically propagated across components.

On the other hand, the prime shortcoming of a full-knowledge approach to the safe behavioural model is that it requires knowledge regarding the behaviour of system controllers and the evolution of the physical process that might not always be available.

In Section 3.3, we have highlighted that the automatic derivation of a behavioural model of the CPS is outside of the scope of our methodology. In fact, deriving the state chart model of a running system is not a trivial task which can be achieved in different ways from the analysis of controller programs [CMT18] to *black-box* inference based on fuzzing [MBMS21].

We assume that Cassandra is applied at the early design stage of the CPS and that a description of the expected high-level behaviour is provided by design teams. We believe that this

assumption is reasonable in the context of the design and development of complex safety-critical systems. An additional critical issue in this approach lies in the complexity of the produced SBM. In fact, while a complete and detailed SBM is preferred as it leads to more precise results, a complete verification of the reachability of unsafe control actions is not always feasible if the SBM is too large.

We also defined a second approach to the description of the safe behavioural model to analyse the propagation of cascading effects. In this second approach, also referred to as *partial-knowledge*, we abstract from the specific implementation of the controllers and only define the dependencies between the input and the output of the components. This is different from the full-knowledge SBM, where we model the concrete behaviour of the system. Instead, in a propagation model, we express input-output dependencies through a set of functional relationships. With this representation, we are able to capture, for each component, which output is affected following a perturbation of its input. This approach is broadly based on the propagation model proposed in [MLS+21], and extends it by allowing us to specify different *modes* for each controller. This approach addresses the following shortcomings identified for the first approach, in fact,

- To implement a safe behavioural model based on a propagation approach, we do not need extensive knowledge regarding the specific behaviour of each component of the CPS. In fact, for each component, we only need to express high-level input-output relationships.
- By not requiring extensive knowledge of the system behaviour, a propagation approach can be applied starting from the early stages of system design.
- Similarly, it can also be built at later stages when the concrete behaviour of a component is partially unknown or not available.
- By not implementing complex behaviour a propagation approach allows to swiftly verify the reachability of unsafe control actions.

On the other hand, these benefits come at the expense of modelling details. The partial-knowledge approach, in fact, does not generally allow us to verify reachability queries that

take into account the dynamics (evolution) of the physical process, e.g., *do the variables of a physical process assume unsafe values?*. Similarly, since it relies on fixed rules to achieve the propagation of cascading effects, it cannot discover threat scenarios depending on the specific implementation of the components, e.g., *a logical flaw in the implementation of the control algorithm, a sensor uses a specific type to represent data, etc.* An additional limitation stems from the fact that rules for propagating cascading effects need to be manually specified for complex components.

In Chapters 7 and 8, we use a full-knowledge approach to describe the behavioural model of two of the three use cases that we have analysed during our study. In Chapter 9, we leverage an approach based on partial-knowledge to study the safety of a flight management system. In the sections below we provide a more formal description of the two approaches.

4.1.3 Verification

In Section 2.6, we have defined a UCA as a tuple $(ca, ctx, r, type)$, where ca is the control action, ctx the context of unsafe application, and r eventual values for which ca is unsafe in ctx. The latter is only valid if ca is of the type *provided*. Hence, to verify the reachability of unsafe control actions in the SBM, it is essential to establish a map that allows us to translate the notions of ca, ctx and r across the two models (SCS and SBM).

4.2 Full-Knowledge approach

We first describe the formalisation of the approach based on *full knowledge* of the controller behaviour. The aim of this section is to describe the proposed full-knowledge approach to the analysis of cascading effects in cyber-physical systems.

As mentioned previously, we use the tuple (C, F, K) to refer to the safe control structure of a CPS with component set C, information flow set F, and K set of functional channels. We

denote with N_c the cardinality of the component set C, with $N_p \leq N_c$ the number of physical processes, and with N_f and N_k the cardinality of F and K, respectively.

To reduce the complexity of the resulting safe behavioural model, we explicitly model the behaviour of only a subset of elements in the original safe control structure. In doing this, we aim to find a trade off between how well the SBM reflects the behaviour of the CPS and how complete are the results obtained from the formal verification process. In particular, a component is said to be modelled *explicitly* if its behaviour is captured by a dedicated timed automaton in the safe behavioural model. On the other hand, a component *is not* modelled *explicitly* when we assume that it behaves safely (e.g., deviations from nominal behaviour are deemed not possible or do not impact the safety of the dynamics of the CPS). A *non explicitly modelled* component is not included in the safe behavioural model and control actions and feedback need to be replaced/created to preserve the communication in the CPS. The last point will be discussed in more detail in the later paragraphs. From an adversarial standpoint, attacks against non-explicitly modelled components cannot be included in the threat model. For example, in the use case in Chapter 7, some of the actuators and sensors are not included and direct communication between the physical process and controller is assumed in the safe behavioural model. In doing so, we have assumed that the threat model does not include *physical* attacks, e.g., tampering/spoofing a sensor. Similarly, a functional channel is said to be modelled *explicitly* if its behaviour is captured by a dedicated timed automaton in the safe behavioural model. By explicitly modelling functional channels we are able to consider message loss/manipulation against control actions and feedback exchanged on said functional channel as part of our threat model. In contrast, a functional channel is said *not to be* modelled *explicitly* if its behaviour is not captured by a dedicated timed automaton in the safe behavioural model.

We use $C' \subseteq C$ to denote the subsets of components that are modelled explicitly in the safe behavioural model. Let K^* and F^* be the set of functional channels and control actions and feedback between components in C', we use $K' \subseteq K^* \subseteq K$ to denote functional channels that we choose to model explicitly in the SBM, and $F' \subseteq F^* \subseteq F$ to denote control actions and feedback exchanged over functional channels in K'. We also define $N_f^* \leq N_f$ and $N_k^* \leq N_k$, the number of control actions and feedback, and functional channels in F^* and K^*, respectively;

and $N'_c \leq N_c$, $N'_f \leq N^*_f$ and $N'_k \leq N^*_k$, the number of components, control actions and feedback, and functional channels in C', F', and K', respectively.

Let $\mathcal{N} = (\mathcal{T}, \mathcal{C}, \mathcal{B})$ be the safe behavioural model of the CPS, and (C, F, K) its safe control structure, we define the following mapping between $\mathcal{N}$ and (C, F, K)

- The set of timed automata is $\mathcal{T} = \mathcal{T}_{C'} \cup \mathcal{T}_M \cup \mathcal{T}_{K'}$. We use the notation $|\mathcal{T}|$, $|\mathcal{B}|$ and $|\mathcal{C}|$ to represent the size of respectively $\mathcal{T}$, $\mathcal{B}$ and $\mathcal{C}$.
- Timed automata in $\mathcal{T}_{C'}$ model the behaviours of the components in C' and $|\mathcal{T}_{C'}| = N'_C$.
- Timed automata in $\mathcal{T}_{K'}$ model the behaviours of functional channels in K' and $|\mathcal{T}_{K'}| = N'_K$.
- Timed automata in $\mathcal{T}$ are synchronised over $|\mathcal{C}| = 2 * N'_f + (N^*_f - N'_f) = N'_f + N^*_f$ synchronisation channels. Let, $\mathcal{B}_v \subseteq \mathcal{B}$ be a set of buffer variables, automata in $\mathcal{T}$ exchange data using $|\mathcal{B}|_v = N^*_f$ buffer variables.
- Control actions and feedback in F' are modelled using a pair of synchronisation variables, and a buffer variable. The latter is only used if data is exchanged.
- Control actions and feedback in $(F^* - F')$ are modelled using a single synchronisation variable, and a buffer variable. The latter is only used if data is exchanged.
- Timed automata in $\mathcal{T}_M$ are instantiated to *observe* the unsafe applications of control actions of the type *not provided*, *provided with wrong duration* and *provided with wrong timing*.
- $N_M = |\mathbf{T}_M|$ is the number of observer automata.

In Figure 4.1, we use a class diagram to provide an overview of the mapping that we have defined for a plain, full-knowledge approach, to the analysis of cascading effects in the cyber-physical system. The aim of this paragraph is to provide a high level overview of the proposed mapping. A more formal definition follows in the paragraphs below. For improved readability, some of the classes have been duplicated in the class diagram in Figure 4.1. This allows us to represent relationships between elements while avoiding complicated line jumps, thus reducing

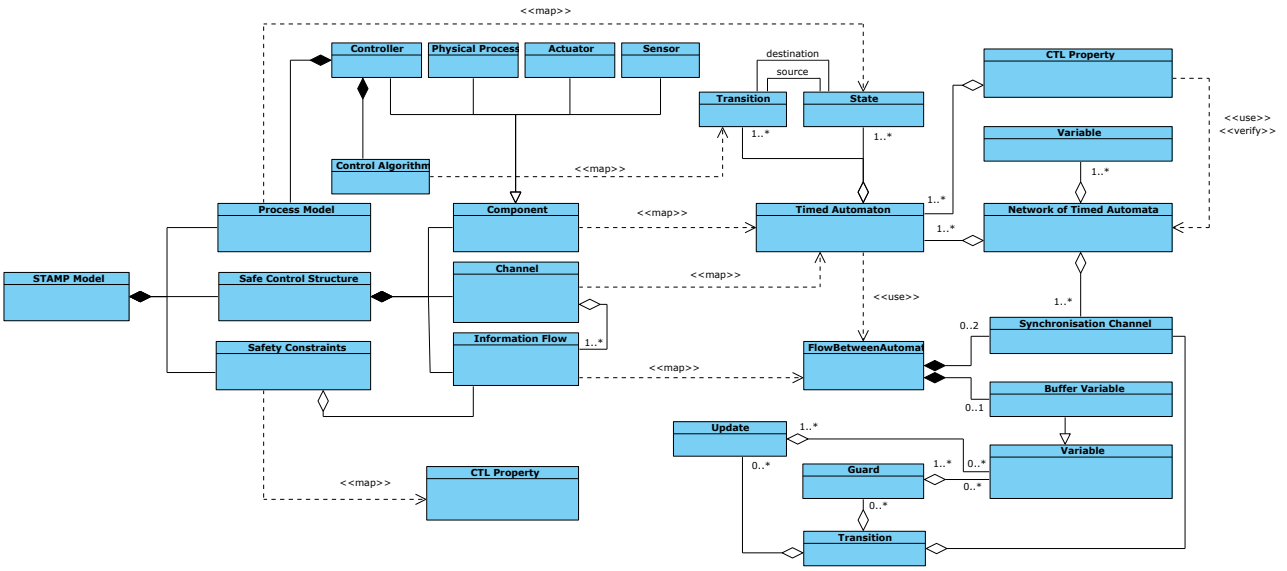


Figure 4.1: Full-knowledge mapping

confusion. Duplicated entries refer to the same entity. The multiplicity of association and aggregation relationships in the diagram is 1, unless otherwise specified.

In the diagram, we have introduced two relationship stereotypes, namely *map* and *verify*. The first explains the existence of a mapping between two classes, whereas *verify* represents that, following the direction of the relationship, elements of the first class are formally verified on those of the second class, e.g., a TCTL property can be verified on the network of timed automata.

In the following paragraphs we detail the formalisation of the mapping between timed automata, synchronisation channels and global variables in the NTA and components, functional channels and control actions and feedback in the safe control structure.

Components As mentioned earlier in this section, we employ timed automata to model components in $C' \subseteq C$. In this type of approach, the definition of locations, invariants, edges and I/O alphabet of each automaton largely depends on the provided design of the respective component, as well as on its role in the context of the whole CPS.

In this paragraph, we propose a mapping between the safe control structure and the given model of behaviours of the CPS. We use these definitions to construct the safe behavioural

model presented in our use case scenarios in Chapters 7 and 8.

Let $T_i(\Sigma^i, L^i, l_0^i, \mathcal{X}^i, \mathcal{V}^i, E^i) \in \mathcal{T}_{\mathcal{C}'}$ be the timed automaton that encodes the behaviour of the component $c_i \in C' \subseteq C$, we define T_i as follows

- $\Sigma = \Sigma_i^i \cup \Sigma_o^i$. Σ_i^i directly depends on the set of control actions (feedback) that are provided as input to c_i in the SCS. Similarly, the composition of Σ_o^i depends on the control actions (feedback) output of c_i. While discussing control actions (feedback), and functional channels, we will provide more details on the composition of Σ for automata that model components. In fact, we will see that the composition of Σ changes depending on the type of modelling adopted for the functional channel (explicit or not) and on whether the component source (destination) of the input (output) belongs to C'.

- L is the set of locations and l_0 is the initial location. Locations are defined as functions of local variables and depend on the provided behaviour of the individual components. However, if the component is a controller, for each control action that is issued, we define a location *provided*. The TA enters this location when issuing the respective control action. We will see that this structure enables the verification of unsafe control actions of the type *provided*.

- $\mathcal{V}$ is the set of local variables. We aim to build a one-to-one mapping between the process model variables of c_i and the state variables of T_i.

- The definition of each edge $(l, l', \sigma, o, \lambda, \delta)$ in the edge set E varies depending on the type of component presented and its behaviour.

- $\mathcal{X}$ is the set of local clock variables.

- I is the set of invariants, which restrict the number of locations that can be visited in function of clock values.

For controllers the set of locations, edges, local variables and invariants are built starting from the description of the control algorithm and process model; at the same time, for physical

processes, they depend on their physical dynamics. The description of components of types of sensors and actuators depends on the complexity of the operations that they carry out.

The existence of a correspondence between STAMP *process model variables* and *automata local variables* allows us to unambiguously translate the *context* of a UCA. In order to reduce the complexity of the resulting safe behavioural model, not all the components in the SCS have to be modelled in the SBM and we did not model, in the SBM, the behaviour of components $(C - C')$. Generally, in the use cases studied in Chapters 7 and 8, we assume that sensors and actuators do not implement complex algorithms and have not included them in the behavioural model. Later in this section, we will discuss the implications of control actions and feedback resulting from the exclusion of a component from the safe behavioural model. We will also elaborate on possible strategies to address them.

Functional channels Control actions (feedback) are exchanged over functional channels in the safe control structure. We adopt two different paradigms to model a functional channel in a full-knowledge safe behavioural model: *explicit* and *not explicit*. We choose which paradigm to use during the creation of the safe behavioural model, depending on whether the functional channel is part of our threat model or not. In fact, an explicit model of the functional channel allows us to analyse the impact of availability, and integrity attacks on the control actions (feedback) transported on it, whereas a non-explicit model will not. On the other hand, to explicitly model a functional channel, we require a dedicated timed automaton, augmenting the overall complexity of the safe behavioural model.

Explicit model: Let $T_k(\Sigma^k, L^k, l_0^k, \mathcal{X}^k, \mathcal{V}^k, E^k) \in \mathcal{T}_{K'}$ be a TA that represents the functional channel $k \in K'$, let $F'(k) \subseteq F'$ the control actions (feedback) that are sent on $k \in K'$ in the SCS, and let N_{fk} be the cardinality of $F'(k)$, the automaton is structured as follows.

Let $ca_{kj} \in F'$ be a control action (feedback) exchanged over the functional channel $k \in K'$, with $j = 1, \ldots, N_{fk}$, and N_{fk} the number of control actions (feedback) exchanged over k. In the safe behavioural model, we model ca using a pair of synchronisation channels $(send\_ca_{kj}, rcv\_ca_{kj})$ and the buffer variable $v_{ca_{kj}} \in \mathcal{B}$, with $send\_ca_{kj}$ and rcv_ca_{kj} in $\mathcal{C}$.

- $\Sigma^k = \Sigma_i^k \cup \Sigma_o^k$ is such that

$$\Sigma_i^k = \{send\_ca_{k1}, send\_ca_{k2}, \ldots, send\_ca_{kN_{fk}}\}$$
$$\Sigma_o^k = \{rcv\_ca_{k1}, rcv\_ca_{k2}, \ldots, rcv\_ca_{kN_{fk}}\}$$

- L^k is the set of locations and l_0^k is the initial location. The size of L^k is $|L^k| = N_{fk} + 1$ locations. In fact, to the N_{fk} locations - one for each control action (feedback) - we add a *Idle* location which servers as the starting location of the automaton, such that

$$l_0^k = \{Idle\}$$
$$L^k = \{ca_{k1}, ca_{k2}, \ldots, ca_{kN_{fk}}\}$$

- $\mathcal{V}$ is the set of local variables that is empty for this automaton.

- E is the set of edges contains $2N_{fk}$ edges. From *Idle*, we add N_{fk} transitions, one towards each of the N_{fk} locations. The automaton undertakes the j-th transition on the input word $send\_ca_{kj}$ and, from there, it returns to *Idle*, while also firing a rcv event for the control action (feedback) $f_i \in F'(k)$. Formally, $E = \{e_1, e_2, \ldots, e_{N_{fk}}, e_{N_{fk}+1}, e_{N_{fk}+2}, \ldots, e_{2N_{fk}}\}$ with

$$e_1 = (Idle, ca_{k1}, send\_ca_{k1}, -, -, -)$$
$$e_2 = (Idle, ca_{k2}, send\_ca_{k2}, -, -, -)$$
$$\ldots$$
$$e_{N_{fk}} = (Idle, ca_{kN_{fk}}, send\_ca_{kN_{fk}}, -, -, -)$$
$$e_{N_{fk}+1} = (ca_{k1}, Idle, -, rcv\_ca_{k1}, -, -)$$
$$e_{N_{fk}+2} = (ca_{k2}, Idle, -, rcv\_ca_{k2}, -, -)$$
$$\ldots$$
$$e_{2N_{fk}} = (ca_{kN_{fk}}, Idle, -, rcv\_ca_{kN_{fk}}, -, -)$$

- $\mathcal{X}$ is the set of local clock variables, which can be employed to reason about delays in the propagation of control actions/feedback.

- I is the set of invariants, which along with clock variables, can be employed to reason

about delays in the propagation of control actions/feedback.

A correct implementation of the channel is essential to decouple the senders and receivers in the NTA. In fact, without such decoupling, *send* and *receive* would be executed atomically in the NTA. An example of interactions between two components through an explicitly modelled functional channel is shown in Figure 4.2. The source component writes the values on v_{ca} and *locks* it so that it cannot be changed until it is read. Then the source also issues an event on the synchronisation channel *send_ca*. On the other hand, when the destination component detects a synchronisation event on *rcv_ca*), it reads the transmitted value. We will show that

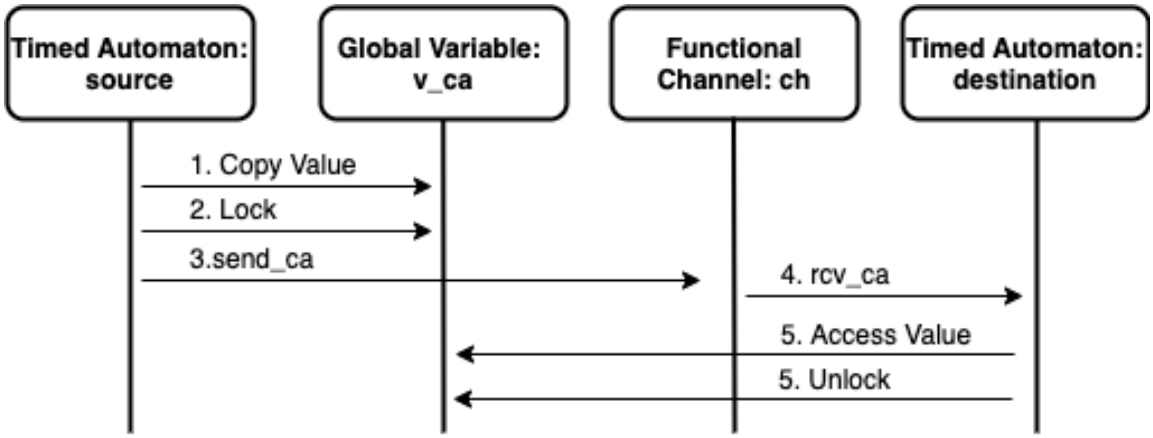


Figure 4.2: Control action (Feedback) - Full Knowledge (Explicit)

the decoupling introduced allows us to model denial-of-service and man-in-the-middle attacks on exchanged messages (Chapter 5).

When two components need to talk to each other, e.g., exchange a control action (feedback), the sender copies the message value into a shared buffer (e.g., global variable) and then synchronises with the channel over a *send*. In normal conditions, the functional channel acts as a relay and sends a synchronisation signal to the receiver, prompting the latter to copy the value from the shared buffer. In adversarial conditions, the attacker can *disrupt* the message (e.g., the channel does not notify the receiver) as well as *change the value* of the message before the receiver is notified.

Finally, let T_{src} and T_{dst} be the timed automata that describe the behaviour of source and destination components of control action ca_{kj} $\forall j = 1, \ldots, N_{fk}$, and $k \in K' \subseteq K$. Then

$send\_ca_{kj} \in \mathcal{C}$ and the buffer variable $v_{ca} \in \mathcal{B}$ are in Σ_o^{src} whereas $send\_ca_{kj} \in \mathcal{C}$ and $v_{ca} \in \mathcal{B}$ are in Σ_i^{dst}.

Non-explicit model: Let k be a functional channel such that $k \in (K^* - K')$ and let $F(k) \subseteq F$ be the control actions (feedback) sent over k, we call $ca_{kj} \in F(k)$ the control actions (feedback) exchanged over k, with $j = 1, \ldots, N_{fk}$, with N_{fk} the cardinality of $F(k)$. We model each ca_{kj} using a single synchronisation channel ($chca_{kj}$) and a buffer variable $v_{ca_{kj}} \in \mathcal{B}$, with $chca_{kj}$ in $\mathcal{C}$. In Figure 4.3, we represent the interactions between source and destination components over a channel that was not modelled explicitly in the safe behavioural model. This case does not need the synchronisation control through the use of *locks* since in the semantics of *NTA* synchronisation events are executed as an atomic sequence.

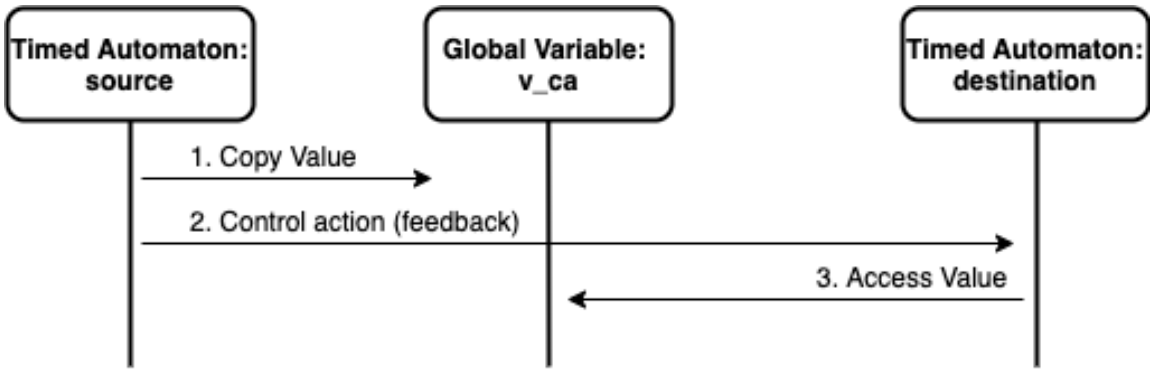


Figure 4.3: Control action (Feedback) - Plain Mapping Case 2

Let T_{src} and T_{dst} be the timed automata that describe the behaviour of source and destination components of ca_{kj} $\forall j = 1, \ldots, N_{fk}$ in the safe behavioural model. Then $chca_{kj} \in \mathcal{C}$ and the buffer variable $v_{ca} \in \mathcal{B}$ are in Σ_o^{src} and Σ_i^{dst}.

Control actions and feedback While mapping the information flow (control actions and feedback) from the safe control structure to the safe behavioural model, we distinguish $four$ different cases depending on whether source and destination components are modelled explicitly or not in the SBM.

1. Source and destination component **are** in C'.

   (a) (Functional channel **is** in K') This is the very same case we gave as an example

in the previous paragraph while describing the behavioural model of a functional channel which is modelled explicitly. Let $ca \in F'$ be a control action (feedback), we represent it through the pair of synchronisation channels $(send\_ca, rcv\_ca)$ and the buffer variable $v_{ca} \in \mathcal{B}$, with $send\_ca$ and rcv_ca in $\mathcal{C}$. Furthermore, let T_{src} and T_{dst} be the timed automata that describe the behaviour of source and destination components in the safe behavioural model. Then $send\_ca \in \mathcal{C}$ and the buffer variable $v_{ca} \in \mathcal{B}$ are in Σ_o^{src}. On the other hand, $rcv\_ca \in \mathcal{C}$ and the buffer variable $v_{ca} \in \mathcal{B}$ are in Σ_i^{dst}. The interaction is shown in Figure (4.2). The source component writes the values on v_{ca} and *locks* it, so that it cannot be changed until it is read. Then the source also issues an event on the synchronisation channel $send\_ca$. On the other hand, when the destination component detects a synchronisation event on rcv_ca), it reads the transmitted value.

(b) (Functional channel **is** in $(K^* - K')$) This is the very same case we gave as an example in the previous paragraph while describing the interaction over a functional channel that has a non-explicit model. Let ca be a control action (feedback) exchanged over the functional channel $k \in (K^* - K')$, we model ca using a single synchronisation channel $(synChannel\_ca)$ and a buffer variable $v_{ca} \in \mathcal{B}$, with $synChannel\_ca$ in $\mathcal{C}$. Moreover, let T_{src} and T_{dst} be the timed automata that describe the behaviour of source and destination components of the control action ca. Then $chca_{kj} \in \mathcal{C}$ and the buffer variable $v_{ca} \in \mathcal{B}$ are in Σ_o^{src} and Σ_i^{dst}. An example of interaction under this paradigm is shown in Figure (4.3). In this case, the sender engages a transition that fires an event on the synchronisation that correspond to the control action and the receiver directly synchronises.

2. Source component c_{src} **is in** C' $(c_{src} \in C')$, while the destination component c_{dst} **is not** $(c_{dst} \in (C - C'))$. Let $\mathcal{G}(V, E)$ be the directed graph associated to the safe control structure, with vertex set $V \equiv C$ and edge set $E \equiv F$. If there exists a third component $c^* \in C'$ such that

- There exists a directed path p in $\mathcal{G}(V, E)$ between c_{src} and c^* such that p is shortest path.
- There is no other path q in $\mathcal{G}(V, E)$ between c_{src} and c^*, such that p and q are of the same length.
- The component (node) $c_{dst} \in p$

We model control actions (feedback) $f_1, \ldots, f_{N_p} \in p$ as one control action (feedback) $\bar{f} \in F'$ between c_{src} and c^* in the safe behavioural model. Moreover, let $k_1, \ldots, k_{M_p} \in K$, the functional channels in p, $\bar{f}$ is exchanged over $\bar{k}$, with $\bar{k} \in K' \subseteq K$ iff at least one between $k_1, \ldots, k_{M_p}$ is in $\in K'$. Otherwise, if $k_1, \ldots, k_{M_p}$ are all in $(K^* - K')$ then $\bar{k} \in (K^* - K')$. Finally, we map the new structure to the safe behavioural model following case 1.a if $k \in K'$ or 1.b otherwise.

In other words, if a component that is modelled explicitly in the safe behavioural model (e.g., $c^* \in C'$) lists, among its input, a control action (feedback) that is output of a component that is not in the safe behavioural model (e.g., c_{dst}) but, at the same time, c_{dst} computes it using information received directly from c_{src} (and c_{src} is in the SBM), we model this case by explicitly modelling a control action (feedback) $\bar{f} \in F'$ between c_{src} and c^*. As an example, let's consider the three components a, b and c in the safe control structure such that $a, c \in C' \subseteq C$, $b \in C$, and $b \notin C'$. Control actions (feedback) $ca_1, ca_2 \in F$ are such that $ca_1 = (a, b)$, $ca_2 = (b, c)$, and $ca_1, ca_2 \notin F'$. In this case, before mapping to the safe behavioural model, we create a control action ca such that $ca = (a, c)$ and $ca \in F' \subseteq F$ as shown in Figure 4.4.

In other words, we bypass the component that is not in C' by artificially introducing the information ca. Then, we refer to case 1 above to model the exchange over a functional channel between two components in C'. This case is shown in Figure 4.7, where we bypass *Speed Controller* and *Acceleration Unit* and merge control actions CA_5, CA_8, and CA_{10} through the creation of a direct interaction between train controller and physical process.

3. Source component **is not** in C' ($c_{src} \in (C - C')$), but the destination **is** in C' ($c_{dst} \in C'$). Let $\mathcal{G}(V, E)$ be the directed graph associated to the safe control structure, with vertex

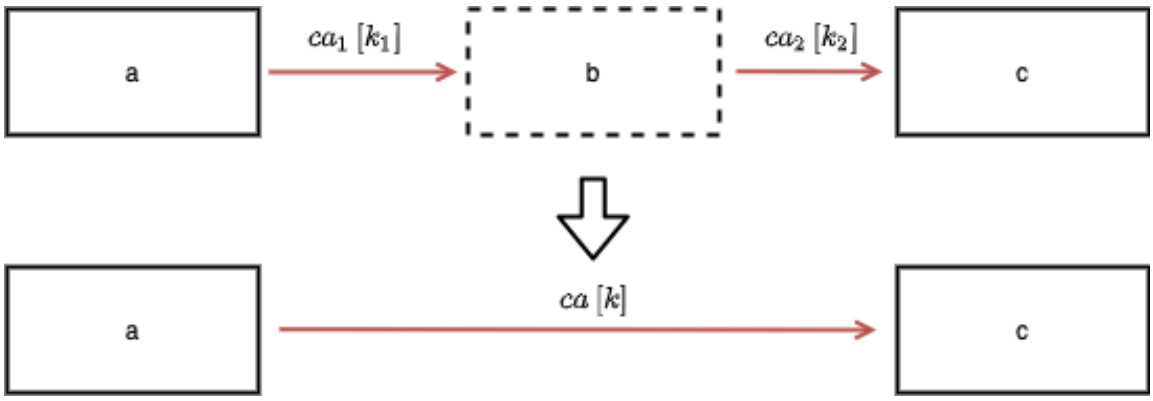


Figure 4.4: Merging control actions (feedback) when component when $a, c \in C'$ and $b \notin C'$

set $V \equiv C$ and edge set $E \equiv F$. If there exists a third component $c^* \in C'$ such that

- There exists a directed path p in $\mathcal{G}(V, E)$ between c^* and c_{dst} such that p is shortest path.
- There is no other path q in $\mathcal{G}(V, E)$ between c^* and c_{dst}, such that p and q are of the same length.
- The component (node) $c_{src} \in p$

We model control actions (feedback) $f_1, \ldots, f_{N_p} \in p$ as one control action (feedback) $\bar{f} \in F'$ between c^* and c_{dst} in the safe behavioural model. Moreover, let $k_1, \ldots, k_{M_p} \in K$, the functional channels in p, $\bar{f}$ is exchanged over $\bar{k}$, with $\bar{k} \in K' \subseteq K$ iff at least one between $k_1, \ldots, k_{M_p}$ is in $\in K'$. Otherwise, if $k_1, \ldots, k_{M_p}$ are all in $(K^* - K')$ then $\bar{k} \in (K^* - K')$. Finally, we map the new structure to the safe behavioural model following case 1.a if $k \in K'$ or 1.b otherwise.

In other words, if a component that is modelled explicitly in the safe behavioural model (e.g., $c_{dst} \in C'$) lists, among its input, a control action (feedback) that is an output of a component that is not in the safe behavioural model (e.g., c_{src}) but, at the same time, c_{src} computes it using information received directly from c^* (and c^* is in the SBM), we model this case by explicitly modelling a control action (feedback) $\bar{f} \in F'$ between c^* and c_{dst}.

An instance of this case is shown in Figure 4.7, where we bypass the F_5 and F_2 or F_3 and F_6. Similarly to the previous case, this case does not need the synchronised access control

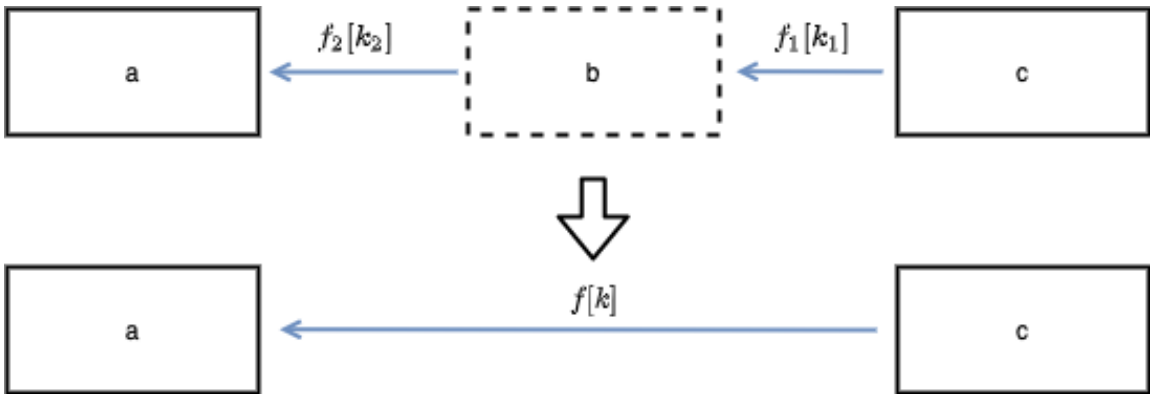


Figure 4.5: Merging control actions (feedback) when component when $a, c \in C'$ and $b \notin C'$

to variables through the use of *locks* since in the semantics of *NTA* synchronisation events are executed as an atomic sequence.

4. One or both among source and destination components **are not** in C' and the configuration of the safe control structure is such that it does not fall within cases 3 or 4 above. In this case, we cannot establish a mapping between the safe behavioural model and the safe control structure of the STAMP model. As a result, unsafe control actions directly or indirectly deriving by control actions (feedback) falling under these cases will not be considered as safety-critical traces in the model-verification process.

Observers In the SBM, we account for many observer automata as there are control actions whose unsafe applications we need to monitor.

Let $(\mathcal{T}, \mathcal{B}, \mathcal{C})$ be a network of timed automata, with $\mathcal{T}$ the set of timed automata, $\mathcal{C}$ set of synchronisation channel and $\mathcal{B}$ set of global variables. We call $T_i = (\Sigma^i, L^i, l_0^i, V^i, E^i, \mathcal{X}^i, I^i) \in \mathcal{T}_M \subset \mathcal{T}$ the timed automaton that we instantiate to monitor the unsafe control action $UCA_i = (ca_i, t, ctx_i, v_i)$, with $ca_i \in C' \subseteq C$. We assume that the context ctx_i and the value v_i map to automata variables, global variables and synchronisation channels in NTA.

The timed automaton monitors the values of global variables in the context ctx_i and the events on synchronisation channels in ctx_i. Observers all share the same structure: they start from an *Idle* location and move towards a *wait/update* location as soon as a specific context is detected

in the NTA. If an application of ca_i is detected, the TA transitions towards a *provided* location. Similarly, if ca_i is not detected within x_{max} time units, then the TA enters a timeout location (control action *not provided/provided too late*). By verifying the reachability of scenarios where the TA enters the *provided* while the clock variable $x \in \mathcal{X}$ is not within the safe bounds $x_{min} \leq x \leq x_{max}$, we are able to detect the unsafe applications of control actions of the type: *not provided/provided late* and *provided early*.

Finally, in Section 2.6 we discussed that continuous control actions (e.g. those that are applied for a non-discrete, continuous amount of time) are modelled as two discrete events (e.g., start and stop) in the safe control structure. These control actions are subject to unsafe applications of the type *provided for too long* and *provided for too short*. In this case, we add an additional clock variable $y \in \mathcal{X}$ as well as an additional location (e.g., *Stop Providing*) to $TA_i \in \mathcal{T}_M$. The automaton enters the *provided/Start Providing* location when the start event associated with the control action is detected. Then, it transits towards *Stop Providing* when the stop event is detected. By verifying the reachability of scenarios where the TA enters the*Stop Providing* location while the clock variable is not within the safe bounds $y_{min} \leq y \leq y_{max}$, we are able to verify unsafe control actions of the type *provided for too long* and *provided for too short*.

Formally, the $TA_i \in \mathcal{T}_{\mathcal{M}}$ has the following structure:

- The input alphabet of T_i is composed of the union of those synchronisation channels and global variables of the NTA that map to the contexts of unsafe application of ca_i previously identified in STPA.

  Given the context ctx_i, $\{v_1, \ldots, v_{N_i}\}$ are the global variables in ctx_i, and $ch_j \in \mathcal{C}$ is the synchronisation channel that is updated when the global variable $v_j \in \mathcal{B}$ is updated, $\forall j = 1, \ldots, N_i$. The input alphabet of T_i is

$$\Sigma_i^i = \{v_1, \ldots, v_{N_i}\} \cup \{ch_1, \ldots, ch_{N_i}\}$$

On the other hand, an observer automaton has no output, hence

$$\Sigma_o^i = \{\emptyset\}$$

- The observer automaton T_i has location set $L = \{l_0, l_1, l_2, l_3\}$, with l_0 being the initial location.

$$l_0 = \{Idle\}$$
$$l_1 = \{Wait\}$$
$$l_2 = \{Provided\}$$
$$l_3 = \{Not \quad provided\}$$

if the monitored control action is applied for a continuous time, then $L = \{l_0, l_1, l_2, l_3, l_4\}$ with

$$l_0 = \{Idle\}$$
$$l_1 = \{Wait\}$$
$$l_2 = \{Start \quad Providing\}$$
$$l_3 = \{Not \quad provided\}$$
$$l_4 = \{Stop \quad Providing\}$$

- $\mathcal{V}$ is the set of local variables that is empty in the general case.

- The edge set $E = \{e_1, ...e_4\}$ such that

$$e_0 = (l_0, l_1, \sigma_{ctx_i}, -, (x = 0, y = 0), -)$$
$$e_1 = (l_1, l_2, \sigma_{ca_i}, -, -, x < x_{max})$$
$$e_2 = (l_1, l_3, -, -, -, x \geq x_{max})$$
$$e_3 = (l_2, l_0, -, -, -, -)$$
$$e_4 = (l_3, l_0, -, -, -, -)$$

with $x, y \in \mathcal{X}$ clock variables, σ_{ctx_i} the word that maps to the context ctx_i and σ_{ca_i} that

which maps to the control action ca_i. The word σ_{ctx_i} and σ_{ca_i} depend on how the channel was modelled. In the case of an explicitly modelled channel, we map the word to the *send* event.

In the case of a monitored control action that is applied for a continuous time, then the edge set becomes $E = \{e_1, \ldots, e_5\}$

$$
\begin{aligned}
e_0 &= (l_0, l_1, \sigma_{ctx_i}, -, (x = 0, y = 0), -) \\
e_1 &= (l_1, l_2, \sigma_{start\_ca_i}, -, -, x < x_{max}) \\
e_2 &= (l_1, l_3, -, -, -, x \geq x_{max}) \\
e_3 &= (l_2, l_4, \sigma_{stop\_ca_i}, -, -, -) \\
e_4 &= (l_3, l_0, -, -, -, -) \\
e_5 &= (l_4, l_0, -, -, -, -)
\end{aligned}
$$

- $\mathcal{X} = \{x, y\}$ is the set of clock variables.
- I is the set of invariants, which in this case is empty.

We introduce the concept of *transaction* to provide a more generic view of the observer automata. Let $a!$ and $b?$ be two synchronisation events on two different channels, we define a *transaction* as any pair $< a!, b? >$. Observer automata are employed to verify the time elapsed between the two events that define the transaction and flag any violation of time-sensitive safety requirements.

Verification We express STPA UCAs in timed computational tree logic (TCTL) [ACD90] and verify them against the SBM. The existence of clocks in TCTL allows us to explicitly handle and verify time-constrained hazards. Safety specifications are expressed in TCTL as reachability statements of the form: $E <> \phi_{UCA}$, i.e., *there is at least one computation where* ϕ_{UCA} *is true.* UCAs of the type *provided when not required* or *provided with wrong parameters* are expressed as:

$$\phi_{UCA} := ctx \wedge T_i.l_{CA}\,[arg]$$

where ctx denotes the context, $T_i.l_{CA}$ denotes the location of the controller automaton that issues the control action and arg the (optional) parameters. The verification of UCAs of the type *not provided* (NP), *provided with wrong timing* and *provided with the wrong duration* is carried out through *observers*, e.g., the observer automaton of a control action enters a *wait* state when a particular context is true. From there we can tell if the UCA is not provided within an amount of time $\bar{t}$ or whether it lasts for too long or too short. We use the notation

$$\phi_{UCA} := (CAMonitor).NP \wedge (CAMonitor).clock \leq \bar{t}$$

where $T_i.clock$ denotes the observer's clock. Similarly, UCAs of the type *stopped too early (too late)* are formulated as:

$$\phi_{UCA} := (CAMonitor).Stop \wedge (CAMonitor).clock \leq (\geq)\bar{t}$$

Finally, we combine observers with logical clocks [Lam05] to identify whether a control action is provided too early or too late. Given a reachability statement, the model checker verifies whether the hazardous location is reachable under the current context and returns a *witness trace* π of the sequence of events leading to the hazard. This can be expressed as

$$\phi_{UCA} := ctx \wedge T_i.l_{CA}$$

The strategy of using existential constructs is also employed by Zhao et al. in [ZDLW21]. The translation of UCA into TCTL specifications is a systematic process that leverages the mapping between the SCS and the SBM.

Applied example (model) We have designed, implemented and tested the safe behavioural model of the CBTC subsystem responsible for the zone management. This includes aspects related to the exchange of the train position between the zone controller and multiple trains as well as coordination, permissions and clearances. In this Section, we only report the models of a subset of components in the CBTC system as an illustrative example. A more exhaustive

description of the CBTC system is provided in Chapter 7.

In particular, we have used UPPAAL [BDL+06b, BDL+12, DLL+15] to implement the safe behavioural model of the system and verify safety-critical safety properties. Due to implementation constraints imposed by UPPAAL, such as the order of execution of instruction on edges and only single updates being admitted on each edge, some artefacts have been introduced in the implementation shown in the Figures below. We will comment on these where relevant.

The behavioural model is a network of timed automata (NTA) and consists of $N_p+N_c+N_M+N_k$ timed automata synchronised over $N_{Ch} = 2*N_f$ synchronisation channels, with $|B| = N_f$ buffer variables.

Controllers: Our implementation features $N_c = 2N+1$, where N is the number of trains in the model. In particular, we have two controllers for each train, the train controller (TC) and location unit (LU), and the zone controller (ZC).

Physical process: Our implementation features $N_p = 1$ number of physical processes. We use one TA to summarise the movements of all the trains across the zones.

Observers: Also, for each train, $N_M = 2$ observers monitor unsafe control actions of the type *Not provided* and *provided too Late* on CA_1, CA_2 and CA_4. To reduce verification times, we use the same automaton to observe CA_2 and CA_4.

Logic channels: Our model uses one automaton ($N_k = 1$) to model the behaviour of the GSM-R channel for communications between ZC and trains. For simplicity, we do not include in the NTA the automata that defines the behaviour of the train's local networks and buses. Instead, we assume that the communication on these channels is reliable and instantaneous. As a result of this simplification, our behavioural model does not consider the impact of threats against feedback and control actions exchanged inside each train.

Synchronisation: *Three* pairs of synchronisation channels and *three* buffer variables for each train, allow to orchestrate the exchange of control actions CA_1, CA_2 and CA_4 ($N_f = 3$). Two additional synchronisation channels are used, together with dedicated buffer

variables to enable communication between location units and train controllers (CA_6) and to signal to trigger the evolution of the physical process when the train controllers *decide* to enter a zone.

In Figure 4.6 we show the template of the timed automaton implementing the behaviour of the zone controller (ZC). The automaton starts from *Ready*. When a *zone* message is received

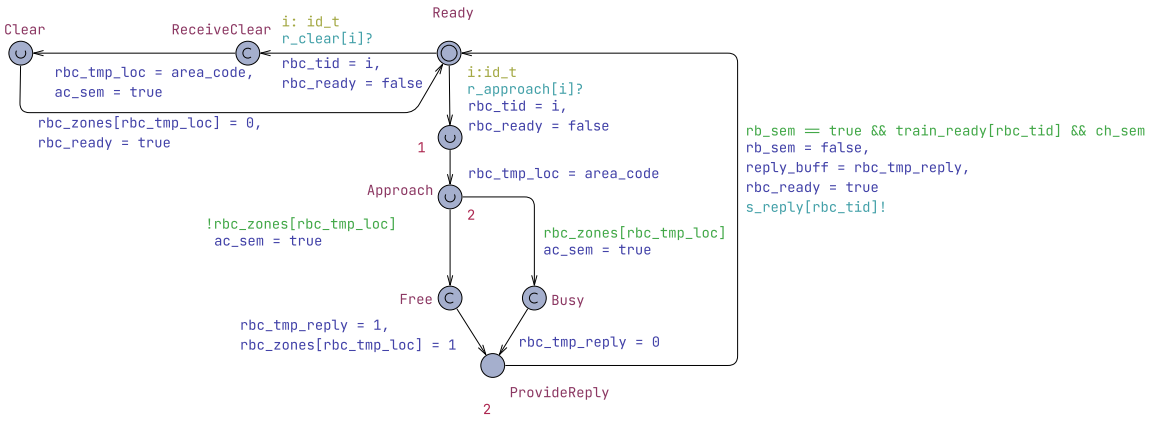


Figure 4.6: Zone controller

(CA_2), ZC saves the *id* of the sender, copies the value from the shared buffer to a local variable and moves to an *Approach* state. ZC evaluates if the requested zone is busy or free and replies to the sender by issuing a *authorisation* message (CA_1). Then, it returns in the *Ready* state, waiting for the next message. From the *Ready* state, when *clear* message (CA_4) is received from train i, the zone controller copies the data from the shared buffer to a local variable and marks as *clear* the respective zone in memory. Then, it returns in the *Ready* state, waiting for the next message. A set of semaphores is used to enforce synchronisation between the zone controller and trains.

In Figure 4.7, we show the template of the timed automaton implementing the behaviour of the train controllers. In our use case, we instantiate 2 automata using this template, one for each train, respectively, with IDs 0 and 1. Each train starts in a *Safe* state. We define a train to be in a *Safe* state when it is travelling within its assigned movement authority (MA). When the train senses, through a change on the variable *lu_loc[train_id]*, that it is approaching the end of the current MA, the timed automaton sends a *zone* message (CA_2) to the zone controller and moves to the *Approach* state while it waits for the reply. When an *authorisation*

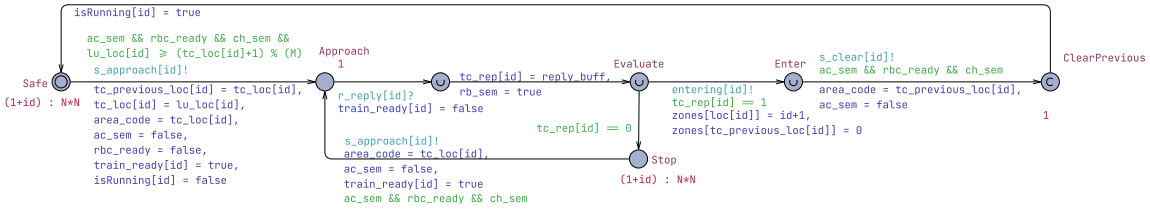


Figure 4.7: Train controller

message (CA_1) is received, the train evaluates its content and, if positive, proceeds towards the next movement authority. If ZC denies access, the train enters a *Stop* state and sends a new authorisation request. When the train enters the next zone, it sends a *clear* message (CA_4) to the zone controller to signal that it has left the previously occupied zone.

In Figure 4.8 we show the template of the timed automaton implementing the behaviour of the physical process. This automaton provides a discrete simulation of the evolution of the state of the mutual exclusive access zones. The state machine stores an array that represents

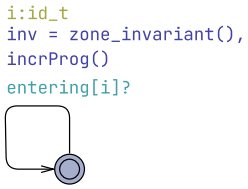


Figure 4.8: Physical process

the current state (free or engaged) of the M zones. Every time that a train enters a zone, the respective zone is marked as *engaged*. Similarly, every time that a train leaves a zone, it is marked as *free*.

In Figure 4.9 we show the template of the timed automaton that models the transmission of a message on the zc-trains channel. The channel starts in a *Wait* state. When a component sends a *approach* (CA_2), *clear* (CA_4) or *reply* control action (CA_1), it sends a synchronisation message on the respective synchronisation channel (*s_approach*, *s_clear* or *s_reply*). The channel automaton synchronises with the sender and makes a transition towards an intermediate state. While the channel automaton is in the intermediate state, the message has been sent but has not been received yet. This allows an adversary to drop the message or change its payload before it is received. If the message is not dropped as a consequence of an adversarial action, the channel synchronises with the receiver on the respective synchronisation channel (*r_approach*,

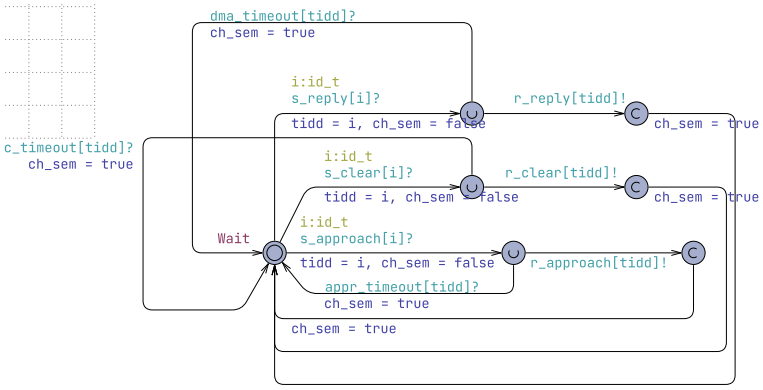


Figure 4.9: Communication Channel

r_clear or *r_reply*). On this synchronisation, the receiver copies the value of the payload from the shared buffer and evaluates it. Modelling functional channels through dedicated automata enables us to split the send and receive phases of the communication between two automata in two subsequent transitions. By doing so we can verify safety properties related to attacks on the integrity and the availability of the information flow, such as altered, missing or delayed communications.

In Figure 4.10 we show the observer for control actions CA_2 and CA_4. For simplicity, we have used one observer to monitor both control actions. The automaton in Figure 4.10 starts in

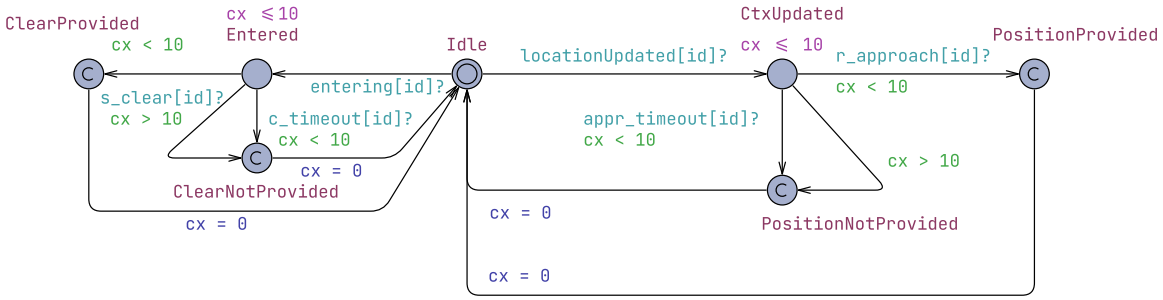


Figure 4.10: Observer 1

a *Idle* state and, when the system enters a context where the control action is expected it moves to *Entered* (if CA_4 is expected) or *CtxUpdated* (if CA_2 is expected). If the adversary *disrupts* the control action or if the control action is not issued within 10 time units (i.e., the adversary disrupted the process model, the controller or other elements in the control loop), the automaton enters a *Notprovided* state.

Applied example (verification) Using the UCAs in Table 2.6 of the CBTC, the hazards in Table 2.3, we define the following TCTL specifications that verify UCAs:

UCA_1, UCA_4 - $\phi_1 := E <> CA1Monitor.Notprovided\ and\ clk < \bar{t}$

Explanation: The ZC must always reply to CA_2. A violation occurs if it is not provided within $\bar{t}$ from CA_2.

UCA_2 - $\phi_2 := E <> zones[i] == j$ and $t\_id \neq j$ and $loc[j-1] == loc[zones[i]-1]$ and $ZC.ProvidedReply$ and $reply == allow$

Explanation: Find an example of trace where, if i-th zone is engaged by train j the ZC replies *allow* to an access request for the same zone incoming from a train with an *id* that is different from the one in the zone i.

(UCA_3, UCA_6) - $\phi_3 := E <> CA1Monitor.Timeout$

Explanation: The automaton that monitors the time delay between CA_2 and CA_1 reaches timeout.

UCA_5 - $\phi_4 := E <> zone_i == 0$ and $t\_id = k$ and $ZC.Busy$

Explanation: Find examples where a controller infers that a zone is "busy" when the zone is not occupied by any train.

We report a more exhaustive list of safety properties when discussing the CBTC use case in Chapter 7.

These safety properties are satisfied by the safe behavioural model of the CPS in the absence of an adversary. To this extent, we have verified that our model checker does not find any traces that violate the above TCTL specifications on the initial model. In the next chapters, we will investigate what happens in adversarial conditions.

4.3 Partial-Knowledge approach

Two are the main limitations deriving from the use of a full-knowledge approach to identify cascading effects leading to hazardous scenarios. First, the completeness, i.e., in the sense of having uncovered all possible scenarios, depends on how accurately the behavioural model of individual components reflects their actual behaviour. Second, the complexity arising from a thoroughly refined safe behavioural model requires the imposition of bounds in order to complete model-checking operations in a sensible amount of time. To overcome these limitations, we propose a second approach to modelling cascading effects across the STAMP model, which aims to model how the impact of an event (e.g., attack) propagates through the system rather than the exact behaviour of system components.

Given the safe control structure (C, F, K) of a CPS with component set C, information flow set F, and K set of functional channels, we denote with N_c the cardinality of component set C, with $N_p \leq N_c$ the number of physical processes, and with N_f and N_k the cardinality of F and K, respectively. Similarly to the previous approach, we use $C' \subseteq C$ to denote the subset of components that are modelled explicitly in the safe behavioural model. At the same time, we denote with F' the set of control actions and feedback that we choose to model explicitly in the SBM. $N'_c \leq N_c$ and $N'_f \leq N_f$ connote the number of components and control actions and feedback in C' and F', respectively.

Let $\mathcal{N} = (\mathcal{T}, \mathcal{C}, \mathcal{B})$ be the safe behavioural model of the CPS, and (C, F, K) its safe control structure, we define the following mapping between $\mathcal{N}$ and (C, F, K)

- The set of timed automata is $\mathcal{T} = \mathcal{T}_{C'} \cup \mathcal{T}_{F'}$.
- As for the previous approach, we use the notation $|\mathcal{T}|$, $|\mathcal{B}|$ and $|\mathcal{C}|$ to represent the size of respectively $\mathcal{T}$, $\mathcal{B}$ and $\mathcal{C}$.
- Timed automata in $\mathcal{T}_{F'}$ model the state of the control action (information flow). We will see that each piece of information flow can be *safe*, *not reliable*, or *not available*. To this extent, $|\mathcal{T}_{F'}| = N'_C$.

- Timed automata in $\mathcal{T}_{C'}$ model how changes to the input of each component propagate to their output. To this extent, $|\mathcal{T}_{C'}| = N'_C$.

  - For *sensors* and *actuators* components, we will assume that modifications and disturbances of the input directly affect their output.
  - For each *controller* in C', we define a set of propagation rules, given alterations to part of their input. These are manually specified, given information on the specific behaviour of the controller.
  - We assume that the *physical process* behaves safely and is secured. However, we include a timed automaton to represent the physical process when there is the need to analyse the propagation of effects between actuators and sensors through a low-level feedback loop.

- Timed automata in $\mathcal{T}$ are synchronised over $|\mathcal{C}| = (2+2) \times N'_f = 4 \times N'_f$. Also, $\mathcal{B}_v \subseteq \mathcal{B}$ is the set of buffer variables which are used by automata to access common data when needed.

- $\mathcal{B}_{ctx} \subseteq \mathcal{B}$ is a set of global variables that we use to represent the *context* cyber-physical system.

In Figure 4.11, we use a class diagram to provide an overview of the mapping that we have defined for the propagation approach to the analysis of cascading effects in the cyber-physical system. The aim of this paragraph is to provide a high-level overview of the proposed mapping. A more formal definition follows in the paragraphs below. As for the previous UML class diagram, some of the classes have been duplicated in the class diagram in Figure 4.11 for improved readability. Such choice allows to picture relationships between elements while avoiding complicated line jumps, thus reducing confusion. Duplicated entries refer to the same entity. The multiplicity of association and aggregation relationships in the diagram is 1 unless otherwise specified. In this diagram, we re-use the relationships stereotypes *map* and *verify* that we have introduced earlier in this chapter.

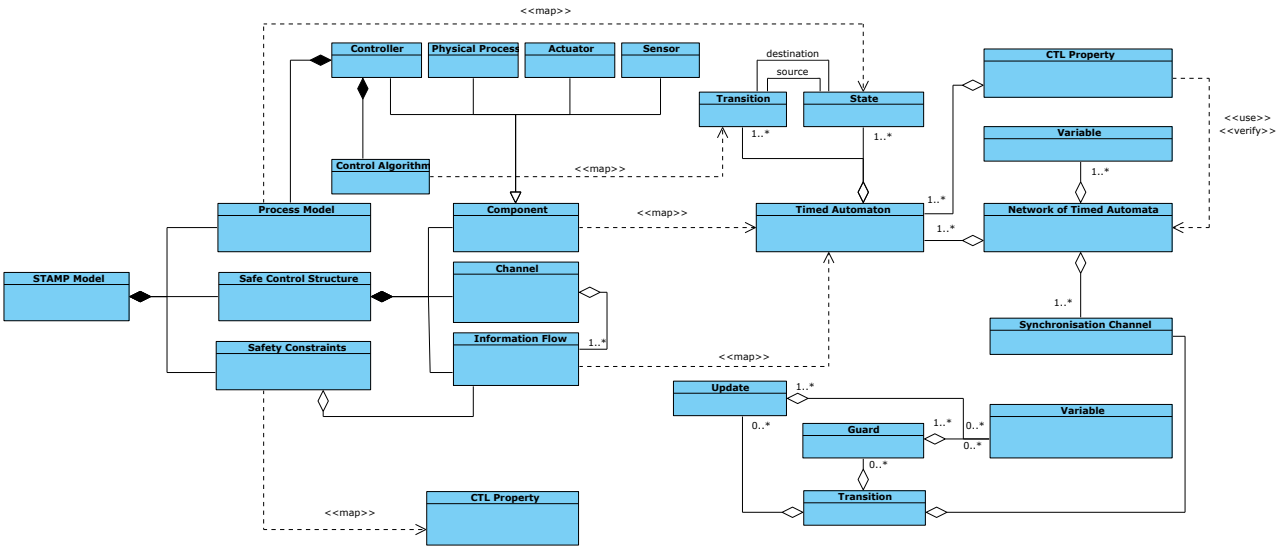


Figure 4.11: Partial-Knowledge approach mappings

In the following paragraphs, we provide the details about the formalisation of the mapping between timed automata in the NTA and components and components, control actions and feedback in the safe control structure.

Feedback We model each *feedback* in the safe behavioural model using a timed automaton $T_i(\Sigma^i, L^i, l_0^i, \mathcal{X}^i, \mathcal{V}^i, E^i) \in \mathcal{T}_{F'_{feedback}} \subseteq \mathcal{T}_{F'}$.

- The automaton alphabet is $\Sigma = \Sigma_i^i \cup \Sigma_o^i$. In non-adversarial conditions, the input alphabet Σ_i^i contains two words; one signals the unavailability of the feedback, while the second signals its unreliability. Likewise, the output alphabet also contains two words two propagate the unavailability and unreliability of the feedback, respectively.

- L is the set of locations and l_0 is the initial location. The automaton counts 3 locations, one for each possible state *Safe*, *Not Available*, and *Not Reliable*. The timed automaton starts from the *Safe* (initial location), where we assume the feedback is available and sound. The automaton transitions towards the state *Not Available* when the feedback becomes unavailable. Similarly, the automaton enters a *Not Reliable* state when the feedback becomes unreliable.

- $\mathcal{V}$ is the set of local variables, which is empty in this case.

- The edge set is $E = \{e_1, e_2, e_3\}$, where each edge e is defined as $e = (l, l', \sigma, o, \lambda, \delta)$

$$e_1 = \{Safe, Not \quad Available, \sigma_{i,1}, \sigma_{o,1}, -, -\}$$
$$e_2 = \{Safe, Not \quad Reliable, \sigma_{i,2}, \sigma_{o,2}, -, -\}$$
$$e_3 = \{Not \quad Reliable, NotAvailable, \sigma_{i,1}, \sigma_{o,1}, -, -\}$$

  In non-adversarial conditions, the feedback becomes unavailable as a result of the propagation, e.g., the issuing sensor/controller became unavailable and issue $\sigma_{i,1}$. When discussing the threat model in the next chapter, we will observe that, in our scenarios, unavailability can also arise as a result of a direct attack (e.g., the attacker directly disrupts feedback). Similarly, from a *Safe* state, the feedback becomes unreliable as a result of a propagation, e.g., the integrity of the issuing sensor/controller was compromised, and $\sigma_{i,2}$ is fired. We will discuss that unreliability can also come from an attack on the feedback. In case of *unavailability* or *unreliability*, the feedback automaton fires respectively $\sigma_{o,1}$ and $\sigma_{o,2}$ to propagate the issue.

- $\mathcal{X}$ is the set of local clock variables.

- I is the set of invariants, which restrict the number of locations that can be visited in function of clock values.

Control actions We model control actions using the same structure of timed automaton that we have employed for the modelling feedback. In fact, control actions are part of the information flow of the safe control structure along with feedback. In order to verify unsafe control actions identified during the STPA analysis, we also need to provide a mapping between *Not Available* and *Not Reliable* states in control action timed automata and types of unsafe control actions in STPA. This mapping is outlined later when we discuss the structure of verification queries in *verification paragraph* below.

Components (sensors and actuators) Similarly to what we have done for the previous approach we assume that sensors and actuators do not incorporate a complex logic. To this

extent, from a high-level perspective, the role of these components is to provide an interface to the physical world by providing measurement (sensor) and executing control actions (actuators). For the purposes of analysing the propagation of cascading effects, we model sensors and actuators as timed automata employing a formalism that is very similar to the one employed to model control actions and feedback earlier. Sensors and actuators that do not respect these assumptions, e.g., they have a complex behaviour and/or with multiple modes of functioning, are modelled as controllers, i.e., using the same structure that we have outlined controller components.

Let $T_i(\Sigma^i, L^i, l_0^i, \mathcal{X}^i, \mathcal{V}^i, E^i) \in (\mathcal{T}_{sensors} \cup \mathcal{T}_{actuators}) \subseteq \mathcal{T}_C$ be the timed automaton that models the behaviour of a sensor (actuator), it shows the following structure

- The automaton alphabet is $\Sigma = \Sigma_i^i \cup \Sigma_o^i$. In non-adversarial conditions, the input alphabet Σ_i^i contains two words for each of the inputs accepted by the component in the safe control structure; one signals the unavailability of the component, and the second signals its unreliability. Likewise, the output alphabet also contains two words to propagate the unavailability and unreliability of the component.

- L is the set of locations and l_0 is the initial location. The automaton counts 3 locations, one for each possible state *Safe*, *Not Available*, and *Not Reliable*. The timed automaton starts from the *Safe* (initial location), where we assume the component is available and sound. The automaton transitions towards the state *Not Available* when the feedback becomes unavailable. Similarly, the automaton enters a *Not Reliable* state when the component becomes unreliable.

- $\mathcal{V}$ is the set of local variables, which is empty in this case.

- The edge set is $E = \{e_1, e_2, e_3\}$, where each edge e is defined as $e = (l, l', \sigma, o, \lambda, \delta)$

$$
\begin{aligned}
e_1 &= \{Safe, Not \quad Available, \sigma_{i,1}, \sigma_{o,1}, -, -\} \\
e_2 &= \{Safe, Not \quad Reliable, \sigma_{i,2}, \sigma_{o,2}, -, -\} \\
e_3 &= \{Not \quad Reliable, NotAvailable, \sigma_{i,1}, \sigma_{o,1}, -, -\}
\end{aligned}
$$

In non-adversarial conditions, the feedback becomes unavailable in two circumstances; as a result of the propagation, e.g., the input control action (feedback) becomes unavailable and issues $\sigma_{i,1}$, or the actuator (sensor) itself becomes unavailable. When discussing the threat model in the next chapter, we will observe that, in our scenarios, the unavailability of the input as well as that of the sensor (actuator) can arise as a result of a direct attack (e.g., the attacker directly disrupts input or component). Similarly, from a *Safe* state, the actuator (sensor) becomes unreliable as a result of the propagation, e.g., the integrity of the input had been compromised and $\sigma_{i,2}$ was fired, or the actuator (sensor) itself becomes unreliable. In our threat model, we will discuss that the unreliability of the input, as well as that of the component itself, can be the result of an attack. In case of *unavailability* or *unreliability*, the feedback automaton fires respectively $\sigma_{o,1}$ and $\sigma_{o,2}$ to propagate the issue.

- $\mathcal{X}$ is the set of local clock variables.
- I is the set of invariants, which restrict the number of locations that can be visited in function of clock values.

Controllers Each controller in the safe control structure is modelled using a timed automaton in the safe behavioural model. Let $T_i(\Sigma^i, L^i, l_0^i, \mathcal{X}^i, \mathcal{V}^i, E^i) \in (\mathcal{T}_{controllers}) \subseteq \mathcal{T}_C$ be the timed automaton that models the behaviour of a sensor (actuator), it shows the following structure

- The automaton alphabet is $\Sigma = \Sigma_i^i \cup \Sigma_o^i$. In non-adversarial conditions, the input alphabet Σ_i^i contains two words for each of the inputs accepted by the component in the safe control structure; one signals the unavailability of the input, and the second its unreliability. At the same time, the output alphabet also contains two words which propagate the unavailability and unreliability of the output, respectively.
- L is the set of locations and l_0 is the initial location. The simplest controller has location set $L = \{Safe, Not \quad Available, Not \quad Reliable\}$ which is equivalent to the one that we proposed for actuators and sensors components. Without an explicit specification of the

process model and control algorithm in the STAMP model, we assume that the entirety of controller output depends on its input. To this extent, if one of the inputs becomes unavailable or unreliable, the output will change accordingly.

Our approach enables us to also analyse the propagation of effects also in the cases of controllers that use complex input/output relationships and with more than just one mode of functioning. In case of multiple modes, alongside the three locations *Safe* (main mode), *Not Available* (output entirely unavailable), and *Not Reliable* (output entirely unreliable), we have N_{AM} additional locations, one for each additional alternate mode of functioning of the controller. Each of these locations is associated with none, partial, or complete unavailability (unreliability) of controller output when part of the input becomes unavailable (unreliable). Associations between the loss of availability (reliability) of the input and the respective alternate mode of functioning are done manually. Likewise, alternate modes of functioning also need to be manually associated with the loss of availability (reliability) they cause to the output. In other words, the propagation of the loss of unavailability (unreliability) on the input of each component needs to be defined manually.

In both cases, the controller automaton starts in a location $l_0 = \{Safe\}$.

- $\mathcal{V}$ is the set of local variables, which is empty in this case.

- The composition of the edge set E of the controller varies depending on the complexity of the control algorithm and on the number of alternate modes of functioning of the controller. In the case of a simple controller (no alternate mode of functioning), the edge set of the controller completely matches the edge set of non-controller components (sensors and actuators). However, if the controller allows for alternate modes, transitions between normal mode (*Safe*) to other locations and between locations entirely depend on the description of the controller and need to be manually set from expert knowledge.

- $\mathcal{X}$ is the set of local clock variables.

- I is the set of invariants, which restrict the number of locations that can be visited in function of clock values.

Verification Again, we rely on existential queries in TCTL to verify the reachability of scenarios of applications of unsafe control actions [ZDLW21]. This strategy is analogous to the one that we have adopted to verify the reachability of unsafe scenarios in the full-knowledge approach to the safe behavioural model.

The verification of unsafe control actions in the previous approach relied on the safe behavioural model being aligned with the assumed behaviour of system components. To this extent, we could verify whether the network of timed automata reached a state where a specific control action was *provided* in an unsafe context. Using the same principle, we use *observers* to verify whether control actions were *not provided*, or provided with *wrong timing* or with *wrong duration* under specific contexts.

Unfortunately, this is not possible in a propagation approach. In fact, in this case, the safe behavioural model does not describe the evolution of the state of CPS. Instead, it captures how certain conditions such as *alternate modes of functioning*, but also loss of *availability* or *reliability* affect control actions and feedback and how these effects propagate through the control structure. In this paragraph, we build a correspondence between states of the automata in $\mathcal{T}'_F$ - which represents control actions - and types of unsafe control actions. In building such correspondence we assume that a *loss of availability* of a control action can lead to:

- An unsafe control action of the type **non provided**.
- An unsafe control action of the type **provided too late**, if the loss of availability is temporary.
- An unsafe control action of the type **provided for too short**, if the loss of availability ends the application of the control action before time.

Similarly, we assume that the *loss of reliability* of a control action leads to

- An unsafe control action of the type **non provided**. If the control action is not applied due to an unreliable information flow.

- An unsafe control action of the type **provided**. If the control action is provided when it should not due to an unreliable information flow.

- An unsafe control action of the type **provided too early** or **too late**, if the loss of reliability affects the timing of the control action.

- An unsafe control action of the type **provided for too short** or provided for **too long**, if the loss of reliability respectively ends the application of the control action before time, or when it is too late.

Safety specifications are expressed in TCTL as reachability statements of the form: $E <> \phi_{UCA}$, i.e., *there is at least one computation where* ϕ_{UCA} *is true.*

We verify unsafe control actions of the types *provided when not required*, *provided too early*, and *provided for too long*, by verifying that a *loss of reliability* of the related control action $ca_i \in F'$ is not reachable in the NTA. This is done by model-checking the following TCTL property

$$\phi_{UCA} := T_i.NotReliable \wedge [ctx]$$

where $T_i \in \mathcal{T}'_F$ denotes the timed automaton that represents ca_i, and ctx denotes the context which is optional.

Similarly, we verify unsafe control actions of the types *provided when not required*, *not provided when required*, *provided with wrong timing*, or *provided with wrong duration*, by verifying that both *loss of availability* and *loss of reliability* are not reachable in the automaton that represents the related control action $ca_i \in F'$. We achieve this by model-checking the following TCTL properties

$$\phi_{UCA} := T_i.NotReliable \wedge [ctx]$$
$$\phi_{UCA} := T_i.NotAvailable \wedge [ctx]$$

where $T_i \in \mathcal{T}'_F$ denotes the timed automaton that represents ca_i, and ctx denotes the context which is optional.

Applied example (model) We have designed, implemented and tested the propagation model for the CBTC subsystem responsible for the zone management, with $N_{trains} = 2$ trains. Also, in this case, we have only modelled aspects related to the exchange of the train position between the zone controller and multiple trains and omitted those related to speed control. We have used UPPAAL [BDL+06b, BDL+12, DLL+15] to implement the safe behavioural model of the system and verify safety-critical safety properties.

The behavioural model is a network of timed automata (NTA) and consists of $1 + N_{trains}(N_C + N_F - 1)$ timed automata synchronised over $N_{Ch} = 4 \times N_{trains} \times N_f$ synchronisation channels.

Controllers: Our implementation features $N_c = 2 \times N_{trains} + 1$: a train controller (TC) and a zone controller (ZC) for each train, and a centralised zone controller (ZC).

Control action: Our implementation features $4 \times N$ timed automata to model the status of control actions. Namely, we have modelled the following feedback: Authorisation (CA_1), Zone Access Request (CA_2), Clear Request (CA_4), and Target Speed (CA_5) (Figure 2.4).

Feedback signals: Our implementation features $3 \times N$ timed automata to model the status of feedback signal in the system. Namely, we have modelled the following feedback: GNSS Position (F_2), Radio Position (F_3), and Location (CA_6) (Figure 2.4).

Physical process: Since we are interested in verifying the propagation of unsafe behaviour in the loop train(s) controller - zone controller, we did not model the physical process in the safe behavioural model.

In Figure 4.12, we show the propagation model of the Zone controller. The model comprises three locations, each representing a state of the controller: *safe*, *not available*, or *not reliable*. To these *three*, we included *two* more anonymous (committed) locations between *safe* and *not available* and *safe* and *not reliable*, respectively. This artefact has emerged during the implementation of the model since UPPAL does not allow for both *receiving* and *issuing* synchronisation events on the same edge. Through the use of a *committed* location, we are able to receive and issue events in sequence.

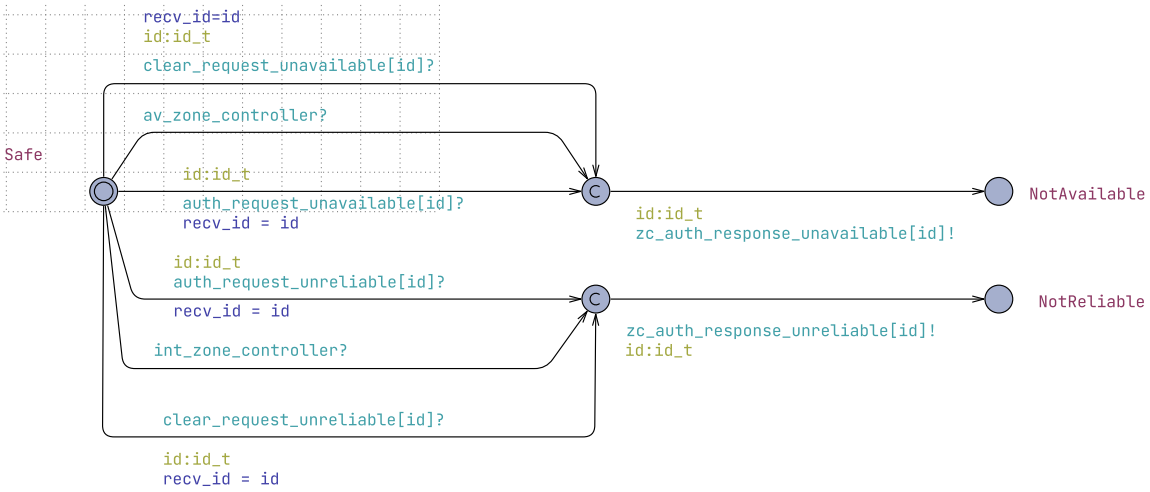


Figure 4.12: Zone controller - Propagation model

A disruption (loss of availability) on the input Zone Access Request (CA_2), propagates to the sole output (we did not model aspects related to speed), rendering the authorisation response (CA_1), output of the zone controller, unavailable. The same effect is produced by a propagated disruption on Clear Request (CA_4). The two propagation rules are based on the assumption in the absence of its input, the zone controller cannot produce an output. Similarly, the output is unavailable also in the case of the zone controller being rendered unavailable itself. On the other hand, any alteration (loss of reliability) affecting Zone Access Request (CA_2) or Clear Request (CA_4), propagates to the output by making the authorisation response (CA_1) unreliable. In fact, an output computed on an unreliable input cannot be relied upon. Similarly, if the zone controller becomes unreliable as a result of a fault or a targeted attack, we consider the authorisation response (CA_1) untrustworthy too.

In Figure 4.13, we show the propagation model of the train controller of the $i-th$ train. As in the case of the zone controller, it includes three locations, each representing a state of the controller: *safe*, *not available*, or *not reliable*. The model also comprehends *three* anonymous (committed) locations between, needed for the implementation to be verified with UPPAAL. A disruption (loss of availability) on the input Authorisation response (CA_1) impacts only the availability of Target Speed (CA_5) output. On the other hand, a disruption of the position (Location (CA_6)) impacts the availability of all the outputs of the controller. In fact, we assume that, in the absence of information about the current position of the train, the train controller

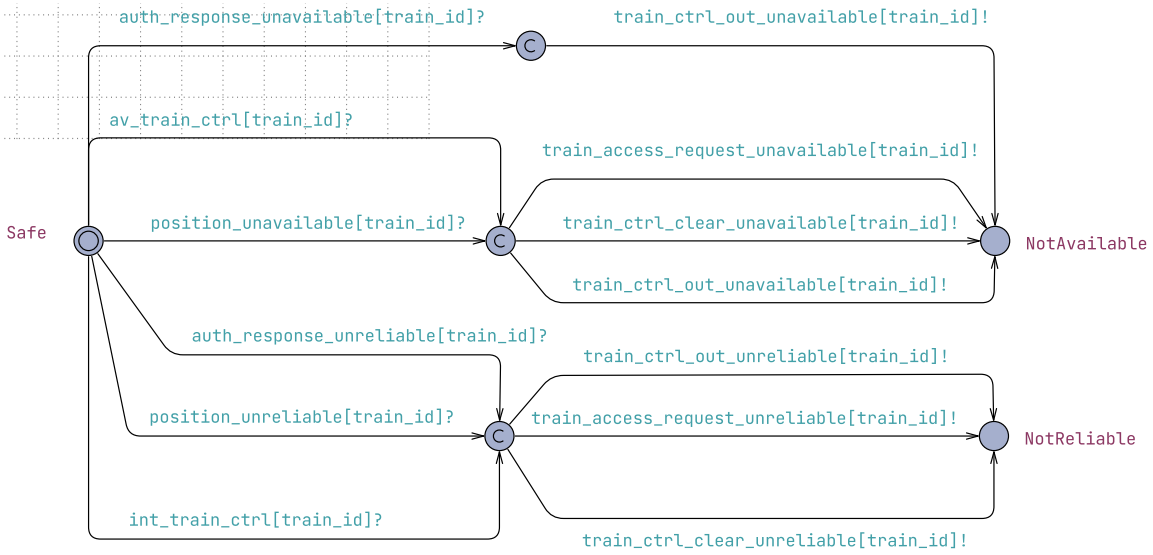


Figure 4.13: Train controller - Propagation model

cannot set a Target Speed (CA_5) nor send Zone Access (CA_2) and Clear Requests (CA_4). The same behaviours are exhibited should the train controller itself become unavailable. On the other hand, any alteration (loss of reliability) affecting Authorisation Response (CA_1) or Location (CA_6), propagates to all the outputs. In fact, any Target Speed (CA_5), Zone Access (CA_2), or Clear Requests (CA_4) would not be reliable if it has been computed on the basis of untrustworthy data.

In Figure 4.14, we show the propagation model of the train controller of the $i-th$ train. As in the case of the zone controller, it includes $five$ locations, each representing a state of the controller: *safe (normal mode)*, *radio only (alt mode* 1*)*, *GNSS only (alt mode* 2*)*, *not available*, or *not reliable*. The model also comprehends *two* anonymous (committed) locations between, needed for the implementation to be verified with UPPAAL. The Location unit has two inputs

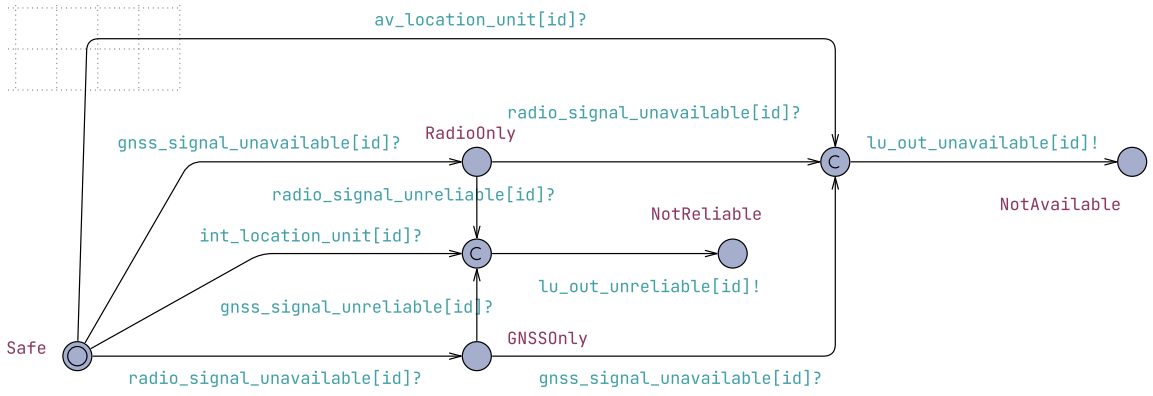
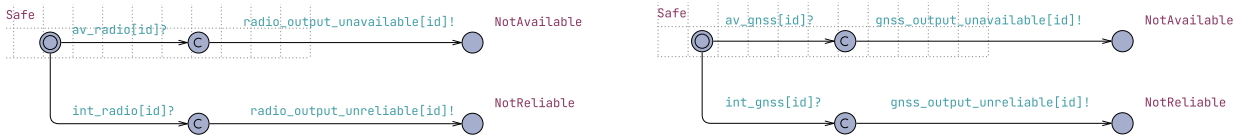


Figure 4.14: Location unit - Propagation model

GNSS Position (F_2) and Radio Position (F_3), and one output and Location (CA_6). It has three alternate modes of functioning: Safe (normal mode), GNSS Only, and Radio Only. The last two are also safe but with lower accuracy. In case of loss of availability of the radio position (F_3), the component switches to GNSS only. On the other hand, if the availability of the GNSS signal is compromised while in normal mode, the component switches to radio only. Finally, in case of loss of availability of both inputs, the availability of the output Location (CA_6) is lost too. We assume that while in normal mode, loss of reliability on one of the inputs does not impact the reliability of the output. At the same time, if the reliability of one of the inputs is loss while the other is unavailable, then also the output Location (CA_6) would not be reliable as it was computed on the basis of untrustworthy data.

In Figure 4.15, we show the propagation model of the Radio and GNSS components of the $i-th$ train. We have modelled these two components as sensors which receive information regarding the current position of the physical process within the system. Both components include *three* locations: *safe*, *not available*, and *not reliable*. They also comprise *two* anonymous (committed) locations between, needed for the implementation to be verified with UPPAAL. Since we chose



(a) Radio Component **(b)** GNSS Component

Figure 4.15: Radio and GNSS Components - Propagation model

to not model the physical process in this instance of SBM, both components transition from *safe* to *not available* only when the component itself becomes unavailable (e.g., as a result of an attack). Similarly, they transition from *safe* to *not reliable* when the component becomes unreliable.

We have used the same approach to model the propagation of cascading effects through the speed controller of the $i-th$ train component in Figure 4.16. In this case, the controller also includes *three* locations: *safe*, *not available*, and *not reliable*. Since we chose to not model the physical process in this instance of SBM, the speed actuator does not propagate the effects further; hence, we do not need anonymous (committed) locations. The component transits from

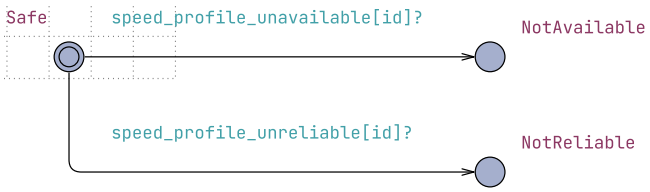
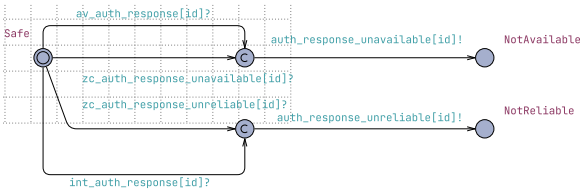


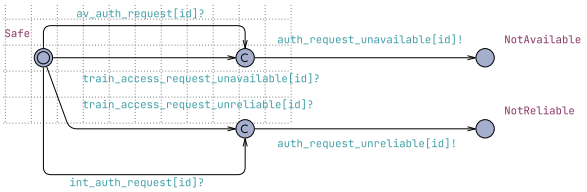
Figure 4.16: Speed controller (actuator) - Propagation model

safe to not available if the input becomes unavailable or when the availability of the component is compromised. On the same lines, the component transitions from *safe* to *not reliable* when the input is not trusted or the component itself becomes unreliable.

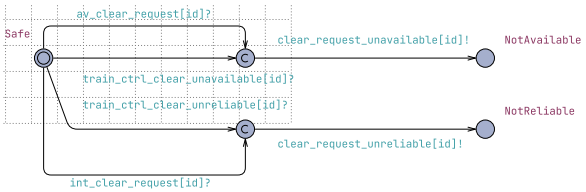
In Figure 4.17, we show the timed automata that we have used to model control actions Authorisation (CA_1), Zone Access Request (CA_2), Clear Request (CA_4), and Target Speed (CA_5), within the $i-th$ train. The models share all the same structure made of *three* locations (*safe*,



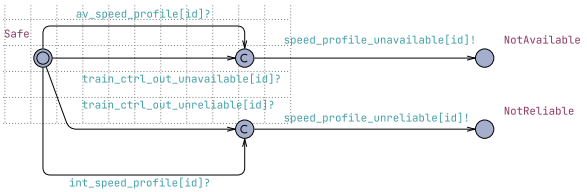
(a) Authorisation (CA_1)



(b) Zone Access Request (CA_2)



(c) Clear Request (CA_4)

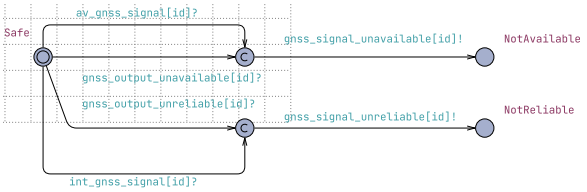


(d) Target Speed (CA_5)

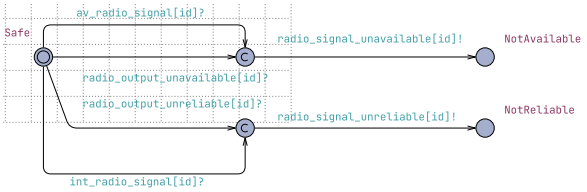
Figure 4.17: control actions Model - Propagation model

not available, and *not reliable*). To each timed automaton we add *two* additional committed locations, to enable the propagation of effects in the UPPAAL implementation. In modelling these, we assume that a control action becomes unavailable (unreliable) as a result of an event, e.g., loss of availability of the information flow, or propagation from the issuing component, e.g., the controller is untrustworthy.

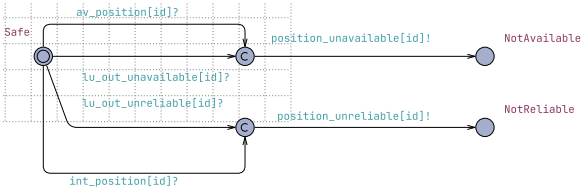
Finally, in Figure 4.18, we show the timed automata that we have used to model GNSS Position (F_2), Radio Position (F_3), and Location (CA_6), within the $i-th$ train. The models share all the same structure made of *three* locations (*safe*, *not available*, and *not reliable*). To each timed



(a) GNSS Position (F_2)



(b) Radio Position (F_3)



(c) Position/Location (CA_6)

Figure 4.18: Control actions model - Propagation model

automaton we add *two* additional committed locations, to enable the propagation of effects in the UPPAAL implementation. In modelling these, we assume that the feedback becomes unavailable (unreliable) as a result of an event, e.g., loss of availability of the information flow, or propagation from the issuing component, e.g., sensor/controller is untrustworthy.

Applied example (verification) We start from the unsafe control actions defined in Table 2.6 to define the TCTL specifications in Table 4.1.

#	TCTL Query	UCA
ϕ_1	$E <> Authorisation.NotAvailable$	UCA_1 : Authorisation Response not provided (zone engaged) UCA_3 : Authorisation Response provided late UCA_4 : Authorisation Response not provided (zone free)
ϕ_2	$E <> Authorisation.NotReliable$	UCA_1 : Authorisation Response not provided (zone engaged) UCA_2 : Authorisation Response provided (allow, zone engaged) UCA_3 : Authorisation provided late (deny, zone engaged) UCA_4 : Authorisation Response not provided (zone free) UCA_5 : Authorisation Response provided (deny, zone free) UCA_6 : Authorisation Response provided too late (allow, zone free)

Table 4.1: Verification UCA - Propagation model

We assume that the safe behavioural model of the cyber-physical system is safe in the absence of an adversary. To this extent, in Figure 4.19, we show that our model checker does not find any traces that violate the above TCTL specifications on the initial model. In the next chapters, we will investigate what happens in adversarial conditions. In Figure 4.19, the automaton *auth_responses(1)* models the behaviour of the control action *Authorisation* between

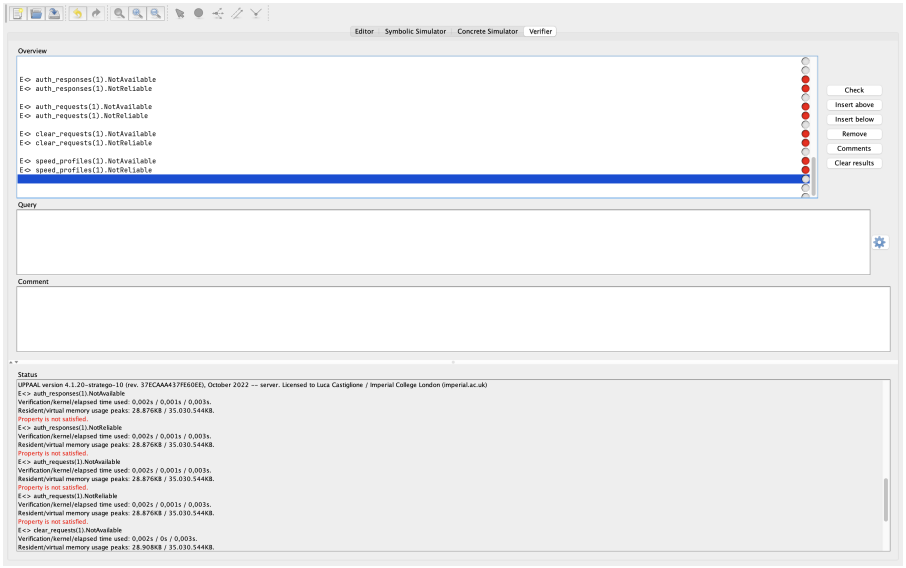


Figure 4.19: Non reachability of TCTL queries under non adversarial conditions

zone controller and the train with index 1. Similarly, *auth_requests(1)*, *clear_requests(1)*, and *speed_profiles(1)* model control actions Zone Access Requests, Clear Requests and Target Speed, respectively.

4.4 Summary & Conclusions

In this chapter, we have elaborated on two novel approaches that we introduce to analyse the cascading effects through the STAMP model of the cyber-physical system. The full-knowledge approach requires the behaviour of each component in the safe control structure to be explicitly modelled. With this strategy, we are able to verify the reachability of unsafe control actions while also considering the state of the system (context). In addition, by including components' internal dynamics, we can easily see how alterations on the input propagate to the output, without the need of manually specifying additional information. On the other hand, such knowledge of component dynamics might not be always available and, even if it is available, the strategy does not easily scale with an increasing number of system components. The *partial-knowledge* or *propagation based* approach, aims to address the shortcoming of its full-knowledge counterpart. This strategy does not require extensive knowledge of the internal dynamics of system components and, for this reason, can be applied starting from the early design

stages. It also scales better with an increasing number of system components. These benefits come at the expense of modelling details. This approach, in fact, does not allow us to verify complex reachability queries that take into account the dynamics (evolution) of the physical process. Likewise, since propagation is achieved through fixed rules, a *partial-knowledge* model would not discover threat scenarios that depend on the specific implementation of components. An additional limitation of the *partial-knowledge* concerns the fact that it requires manual intervention to define rules for the propagation of cascading effects. We have demonstrated the application of both approaches to model a Communication-Based Train Control system (CBTC).

In the next chapter, we dive into the first macro process of Cassandra, our proposed methodology for integrated safety and security analysis. In particular, we show how it uses model checking to identify and enumerate safety-critical threat scenarios.

Chapter 5

Automatic Identification of Threat Scenarios

The malicious use of legitimate system functionality creates cascading effects capable of propagating across the entire cyber-physical system. By judiciously combining malicious actions with legitimate system behaviour it is possible to create significant effects with physical consequences. In safety-critical systems, these include the violation of safety requirements leading to hazards, including damage to equipment, economic loss and even loss of life. Identifying threats or sequences of threats that, when applied, lead to catastrophic effects requires going beyond the simple modelling of threats that affect the control hierarchy. In fact, we also need to consider how these threats can alter the behaviour of individual system components and how the altered behaviour propagates through the rest of the control hierarchy. In this sense, the combination of threat modelling with the analysis of system behaviours allows us to study the effects of actions such as: spoofing sensor measurements or control actions in different system states, maliciously applying control actions at the wrong time or for the wrong duration. Combining these effects with the safety analysis is key in determining when such actions violate the safety properties. This is the case of incidents such as [SS11] where the attacker forces the system towards a backup state and then relies on its adapted behaviour to execute the attack.

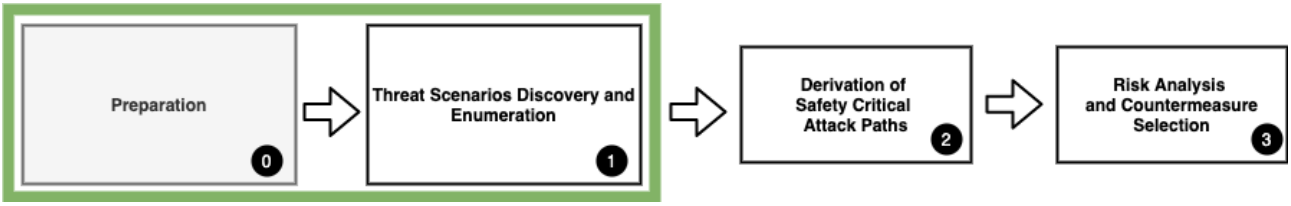


Figure 5.1: Cassandra - Overview

In this chapter, we present the first stage of our proposed methodology. This is the process of *threat scenarios discovery and enumeration*. We also recap the preparation step which we apply early in Cassandra (Chapter 3). We summarise the processes in Cassandra in Figure 5.1 and highlight, in a green box those treated in this Chapter. During the preparation stage (❶), we use System Theoretic Accident Models and Processes (STAMP) to model safety aspects of the CPS and perform the use STPA to derive unsafe control actions. We then model threats against the STAMP model of the CPS and move towards the automatic identification of threat scenarios (❶). Here we enumerate threat scenarios, i.e., malicious actions that, interleaved with system behaviour, can result in catastrophic consequences.

The rest of the Chapter is organised as follows. We present an overview of stages ❶ and ❶ in Section 5.1, and we introduce the threat model in Section 5.2. We formalise the process of discovery of threat scenarios in 5.3 and show its application in 5.4. Finally, 5.5 present the conclusions of this chapter.

5.1 Process overview

In Chapter 3, we have elaborated on the broad steps in Cassandra and highlighted the need for additional *preparation* steps. A detailed overview of the preparatory and first stages of Cassandra is shown in Figure 5.2. Preparatory stages *P*.1 to P.6 precede the application of the stage of *threat scenarios identification*, and are highlighted with a lighter background in the figure. The input consumed at each stage and the output produced are indicated in white and black circles, respectively. A key at the bottom of Figure 5.2 expands the acronyms used for input/output description 5.2. During the preparation, we perform the conventional application

of the first three steps of STPA/STPA-Sec. In particular, (P.1) we identify Losses, Accidents and Hazards, (P.2) we model the Safe Control Structure, and (P.3) we determine the Unsafe Control Actions (Chapter 2). Model checking enables us to automatise the fourth step of

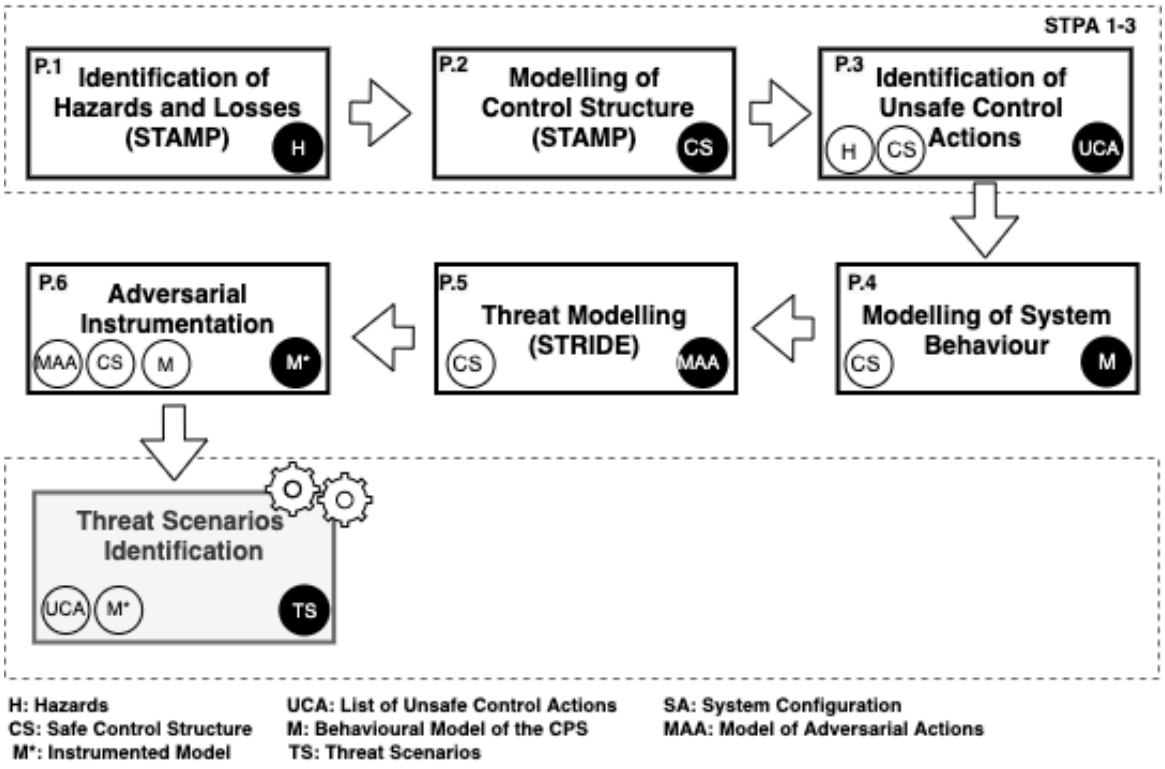


Figure 5.2: Overview of the proposed methodology.

STPA-Sec and automatically identify threat scenarios in the first stage of Cassandra. For it to be applicable we need to model (P.4) the safe behavioural model (SBM) of the CPS and build a map to the safety model (Chapter 4). In this step, we assume that the high-level behaviour of system components is already known from system design and provided as a labelled transition system (LTS). We also propose and apply a set of rules to map a subset of the states of the LTS to the SCS (Chapter 4). These rules allow us to formally verify UCAs on the behavioural model of the system. Then, in (P.5) we apply STRIDE to identify potential threats to elements of the SCS. As the final stage of the preparation, we construct (P.6) the parallel composition between the SBM and the MAA. As STPA and STPA-Sec are accepted for the creation of assurance cases for safety-critical systems (e.g., [EUR18]) we assume that stages P.1, P.2, P.3, and P.5, are carried out already in the design life-cycle of cyber-physical systems. At the same time, we assume that the high-level behaviour of system components is known at design time; therefore

it is possible to design the SBM (P.4). Although the preparation phase is essential to apply our methodology, we leave the development of tools that support it for further work. A detailed discussion of P.5 and P.6 follows in the next section (Section 5.2).

The *identification of threat scenarios* made in Cassandra corresponds with the fourth step of STPA-Sec, where this operation is performed manually. In contrast, we rely on model checking to formally verify the reachability of UCAs in the presence of an omniscient attacker.

5.2 Threat Model

STPA-Sec does not provide any guidelines on the nature of threats to consider while performing its fourth step (identification of threat scenarios). We employ STRIDE [Sho14b] to systematically identify potential threats to elements of the safe control structure. We find that STRIDE provides a more structured approach to the analysis of potential threats compared to traditional STPA-Sec. STRIDE allows us to identify the *adversarial actions (AAs)*, where each action represents the use of a STRIDE threat against an element of the SCS. *By aggregating the adversarial actions into an aggregated model of adversarial actions (MAA) and formally verifying the specifications derived from STPA against the composition of the MAA with the CPS behavioural model, we identify the threat scenarios as sequences of AAs that, combined with the system behaviour, lead to hazards (Section 5.2.1).* To the best of our knowledge, this approach is novel. It allows us to identify the sequences of adversarial actions that, if performed, can violate the safety properties and thus lead to accidents and hazards.

5.2.1 Model of adversarial actions

We employ STRIDE on the STAMP model (Step P.5, Figure 5.2), to build the model of adversarial actions. The MAA is a synthetic aggregated model of the possible behaviour of an adversary. This synthetic model is an overapproximation of an adversary's behaviour, i.e., it includes threat sequences that may not be feasible, but it serves our purpose well. In particular,

we use STRIDE-per-interaction [Sho14a] on control actions, feedback and process models to build the MAA in a semi-automated, systematic, staged process shown in Figure 3.6.

From the systematic application of STRIDE, we derive an *attack library* (AL), i.e., a list of adversarial actions an attacker can apply against the SCS. Formally, the attack library is an alphabet of adversarial actions $AL^* = \{a_1, a_2, \ldots, a_K\}$ of size K. In particular, for each *interaction* outside the defined trust boundary, we consider STRIDE threats against the exchanged information as well as against source and destination components.

Definition 5.1 (Adversarial action) *An adversarial action a is the specialisation of a STRIDE threat to an element of the SCS. It is defined as a tuple* $(type, obj, values, privileges)$.

Where *type* is the type of threat (e.g. spoofing, tampering, etc.), *obj* is the target object in the safe control structure, *values* are the values of the action's parameters (e.g., values that are injected or spoofed), and *privileges* is the set of privileges required to perform the action in the system. The parameter *values* can be null (e.g. Denial of Service), a value or a range of values and is determined during the safety analysis. Because we are applying symbolic model checking, we are considering discrete values (or a range of values), e.g., the spoofing of a control action with value $\bar{x}$. Different adversarial actions may be defined for multiple values in a range, e.g., the spoofing of a control action with value $\bar{x}_1$ and the spoofing of the same control action with value $\bar{x}_2$. In general, determining values for data spoofing/tampering in conjunction with a model-checking approach remains a topic for further research. To our knowledge, no related studies have attempted this.

Privileges are obtained through the exploitation of vulnerabilities (Chapter 6). The map between STRIDE threats and required privileges needs to be carried out manually, as part of the preparation stage (P.5). The output of P.5 is the model of adversarial actions (MAA). The process is shown in (Figure 5.3).

The MAA is a timed automaton and its location set includes the actions in the attack library and a set of buffer locations, which we use to synchronise the MAA with the behavioural model of the CPS. For example, tampering with a message requires that the MAA is synchronised

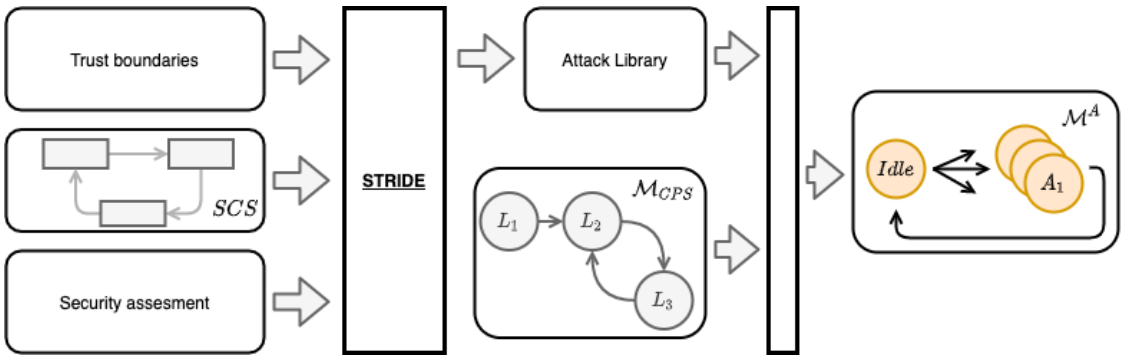


Figure 5.3: Derivation of the model of adversarial actions.

on the communication between a source and a destination component. Specifically, we define one buffer location in the MAA for each control action (and feedback) that the adversary can tamper with in the safe control structure.

In Cassandra, we systematically derive the MAA using the following strategy:

- Each adversarial action in the AL is a state in the MAA.
- The MAA starts from a *Idle* location (initial location)
- From *Idle*, the MAA synchronises with the behavioural model when a control action (or feedback) is exchanged between two components. For this reason, the SBM is also an input in the process of generation of MAA Figure 5.3). From there, the MAA non-deterministically chooses whether to execute the threat, e.g., inject a value, or not. In one case, it performs the malicious action and then returns to *Idle*; in the other, it returns to *Idle* without performing the action.
- From *Idle*, the attacker chooses the next malicious action non-deterministically. After each action, it returns to the initial location.
- We assume that all adversarial actions are independent of each other. However, we allow dependencies between them to be set manually (e.g. when the outcome of an action depends on the success of the preceding action). For example, a *tamper* action can only be reached following an *information disclosure*.

The resulting automaton models the adversary performing any of the adversarial actions identified in the STRIDE model at any point or following the dependencies included in the model. This represents a superset of the possible attacks. An example of an MAA for the CBTC use case is shown in Figure 5.8.

In Step P.6 (Figure 5.2), we then integrate the MAA with the safe behavioural model following a process of adversarial instrumentation (Figure 5.4). The output of the instrumentation is $\hat{\mathcal{M}} = \mathcal{M}_{CPS}|\mathcal{M}_A$ where $\mathcal{M}_{CPS}$ is the safe behavioural model of the CPS and $\mathcal{M}_A$ the MAA.

In general, it is possible to generate more than one MAA and analyse them separately e.g., the MMA can include threats representing the Tactics, Techniques, and Procedures (TTPs) of a specific APT group. Alternative strategies for constructing the MAA also include attack graph analysis to only consider threats feasible in a specific deployment context, thus improving verification times. In fact, having specific MAAs would lead to better scalability at the loss of completeness in the sense of having covered all possible threats against affecting the SCS.

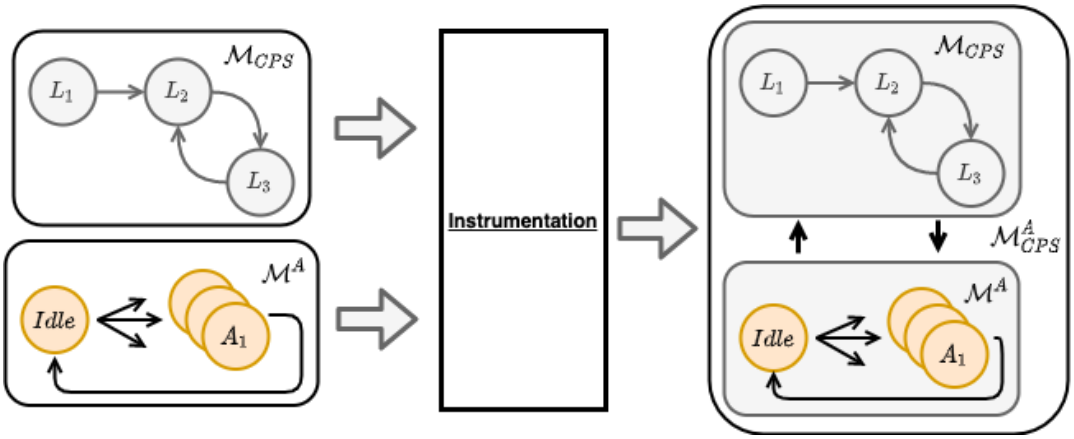


Figure 5.4: Process of adversarial instrumentation

5.3 Discovery of Threat Scenarios

The discovery of threat scenarios implemented in Cassandra (Figure 5.2) is a top-down staged process which uses, as input, the UCAs obtained from the execution of the third step of STPA. We have shown that for each UCA we can define the safety properties as reachability properties in TCTL (Section 4.2). We now use model-checking to verify these safety properties against the

instrumented model of the CPS, i.e., the parallel composition of the SBM and the synthesised attacker model MAA. When a violation of the safety properties in the instrumented model is reachable, the model checker outputs a trace leading to that violation (witness trace). This trace contains the ordered (and timed) sequence of adversarial actions that the threat actor needs to perform to trigger a hazard. We call this sequence of adversarial actions leading to a safety violation a threat scenario.

A traditional verification process would return a single trace that violates the safety properties, whereas we are interested in *all* the traces, i.e., all the ways in which the property can be violated. Therefore, we re-iterate the model-checking process multiple times, each time excluding the traces already found. When no more traces (and threat scenarios) are found, the safety property cannot be violated by the attacker, assuming that the model is correct and complete.

5.3.1 Formalisation

Formally, let H be the hazard caused by the application of $UCAs$

$$H \leftarrow \{UCA\}$$

Let $\mathcal{M}_{CPS}$ be the safe behavioural model of the CPS and $\Phi_H = \{\phi_1, \phi_2, .., \phi_N\}$ the set of safety properties for $UCA_1, ..., UCA_N$, where $UCA_1, ..., UCA_N$ are the UCAs leading to H. Also, let $\mathcal{M}_A$ be the model of adversarial actions built with respect to a specific threat model, and $\hat{\mathcal{M}} = \mathcal{M}_{CPS}|\mathcal{M}_A$ be the parallel composition of $\mathcal{M}_{CPS}$ and $\mathcal{M}_A$. We verify that $\hat{\mathcal{M}} \models \phi_i$, $\forall i \in \{1, .., N\}$ and call π_i the symbolic trace that satisfies ϕ_i. Therefore, a threat scenario (attack) A_i consists of the locations of $\mathcal{M}_A$ visited in π_i, and is defined as $A_i \coloneqq \{e_{i1}, e_{i2}, ..., e_{iK_i}\}$, where e_{ik} are the locations of $\mathcal{M}_A$ contained in π_i. This can be easily generalised to multiple hazards and UCAs per hazard. For example, given hazards $H_1, H2, ..., H_N$,

$$H_1 \leftarrow \{UCA_{11}, ..., UCA_{1M_1}\}$$

$$...$$

$$H_N \leftarrow \{UCA_{N1}, ..., UCA_{NM_N}\}$$

we define $K = \Pi_{i \in N} M_i$ sets of safety specifications (one for each hazard)

$$\Phi_1 = \{\phi_{11}, ..., \phi_{1M_1}\}$$

$$...$$

$$\Phi_N = \{\phi_{N1}, ..., \phi_{NM_N}\}$$

where ϕ_{ij} is the safety specification that verifies the reachability of UCA_{ij}. Thus for $\hat{\mathcal{M}} \models \phi_{ij}$ ($\phi_{ij} \in \Phi_i$), π_{ij} is the respective trace, where there exists an attack A_{ij} leading to UCA_{ij}, and therefore to H_i. An attacker aiming to cause H_i must therefore be able to carry out all the adversarial actions in A_{ij} for any i and j.

5.3.2 Attack enumeration

As mentioned earlier, each verification of each $\phi \in \Phi$ returns a single witness trace that satisfies ϕ. To obtain all the traces that violate ϕ we need to re-iterate the verification excluding the previous found traces until no more traces that satisfy $\hat{\mathcal{M}}$ are found. We have, therefore, developed an algorithm for this.

Notation Let N be the number of timed automata in $\hat{\mathcal{M}}$, we introduce the notation $\phi(\hat{L}_1, \hat{L}_2, \ldots, \hat{L}_{\bar{N}})$ with $\bar{N} < N$ and $\hat{L}_i \subseteq L_i$ a subset of locations of the automaton T_i. Let $\hat{L} = \bigcup_{i=1}^{\bar{N}} \hat{L}_i$ and let s_i be a state of $\hat{\mathcal{M}}$. Formally, we write

$$\hat{\mathcal{M}}, \pi \models \phi(\hat{L}_1, \hat{L}_2, ..., \hat{L}_N) \tag{5.1}$$

if there exists a trace π such that $(s_1, s_2, ..., s_M) \notin \pi$ and $l_j \notin s_i$, with $l_j \in \hat{L}\ \forall\ i$ and $\forall\ j$. In other words, Equation 5.1 holds if there exists trace π that satisfies ϕ *and* for which the automaton T_i *does not transit through any location* $l_{ij} \in \hat{L}_i\ \forall j$ and $\forall\ i \in \bar{N}$.

The verification of queries can be unfeasible for a very large state space. To this extent, we introduce a time bound $\bar{t}$ for the search. This gives us $\phi(l_1, l_2, ..., l_P)_{\leq \bar{t}}$ with $P, \bar{t} \in \mathbb{N}$, where we search for a trace that does not contain $(l_1, l_2, ..., l_P)$.

We show the algorithm to discover threat scenarios leading to the application of an unsafe control action in Listing 1 below. In the description of the algorithm, the notation $\pi \leftarrow (\hat{\mathcal{M}} \models \psi)$ identifies π as one trace proving $\hat{\mathcal{M}}$ and is equivalent to $\hat{\mathcal{M}}, \pi \models \psi$.

Algorithm 1: Iterative generation of specifications for complete enumeration of threat scenarios affecting $\mathcal{M}_{CPS}$, given the initial specification ϕ and the model of adversarial actions $\mathcal{M}_A$.

```
Data: φ ∈ Φ_UCA, M_CPS, M_A, t̄
Result: Threat Scenarios traces
Queries ← φ ;
History = [ ] ;
Threat Scenarios = [ ] ;
M̂ := M_CPS|M_A ;
while Queries not empty do
    φ ← queries.pop() ;
    if π ← (M̂ ⊨ φ) then
        Actions ← A(π, attacker) ;
        ψ ← φ(Actions)_≤t̄ ;
        if ¬ History.contains(ψ) then
            Queries ← ψ
        end
        History ← φ ;
        Threat Scenarios ← Actions ;
    end
end
```

Description The algorithm to enumerate safety-critical attacks against the specification ϕ of a CPS is shown in Algorithm 1. We use the same notation employed earlier, where $\mathcal{M}_{CPS}$ is the safe behavioural model of the CPS, $\mathcal{M}_A$ a representation of adversary behaviour, and $\hat{\mathcal{M}}$ the instrumented model. Given the hazard H triggered by the unsafe control action UCA and being $\Phi_{UCA} = \{\phi\}$ the set of specifications that verifies the presence of H in $\hat{\mathcal{M}}$, the algorithm starts with the verification of the specification ϕ. If $\hat{\mathcal{M}} \models \phi$ with witness trace π, we call *threat scenario* $A(\pi)$ the sequence of locations of the MAA contained in π. We expand $A(\pi)$ with the sequence of adversarial actions contained $(a_1, a_2, ..., a_K)$, which all are locations of $\mathcal{M}_A$. Thus, we generate the new specification $\phi(A(\pi))_{\leq \bar{t}}$ to verify the existence of other attacks different from the one that has been found. We add the sequence $(a_1, a_2, ..., a_K)$ to the

set threat scenarios found. The set *Threat Scenarios*, contains all the *one-sided* combinations of the elements of $A(\pi)$. We use the term *one-sided* as we consider combinations with the same adversarial actions but in different order to be equivalent to each other. The process is repeated until the newly generated specification does not satisfy $\hat{\mathcal{M}}$.

Completeness of results When iterated over all ϕ_{ij}, the algorithm allows us to discover all the threat scenarios, i.e., sequences of adversarial control actions, leading to the violation of a specification ϕ. The enumeration process thus leads to the discovery of threat scenarios that contain redundant adversarial actions. To this extent, discovered scenarios need to be reduced before further processing. Let $A_p = (e_{p1}, \ldots, e_{pN})$ and $A_q = (e_{q1}, \ldots, e_{qM})$, A_p is redundant if and only if $A_p \subset A_q$. Redundant traces can be discarded as they contain adversarial actions that are irrelevant to the successful outcome of the attack.

The process is sound. However, its completeness depends on the attacker model [AK86] and the bound imposed on the model-checking process itself. For large models, it is necessary to set a bound for the search in time, i.e., we use a finite time horizon $t = \bar{t}$, with $\bar{t}$ chosen empirically.

5.3.3 Threat scenarios tree

We leverage a tree-like formalism to summarise the relations between threat scenarios and hazards which we uncovered during the attack enumeration. Let H be a hazard which can be caused by hazardous control actions $UCA_1, UCA_2, \ldots, UCA_N$, and let $A_1, A_2, \ldots, A_N$ the uncovered attacks that the adversary can use to perform to trigger H through the causation of unsafe control actions, we represent the threat scenarios tree of H in Figure 5.5. In Figure, e_{ij} the $i-th$ adversarial action (threats) in the $j-th$ attack. As an example, we have highlighted with a red background the threat scenario $TS-1$, in Figure 5.5. In the next chapter, we also make use of the synthetic representation shown in Figure 5.6 which does not make use of explicit AND/OR gates. The two representations are equivalent for our purposes.

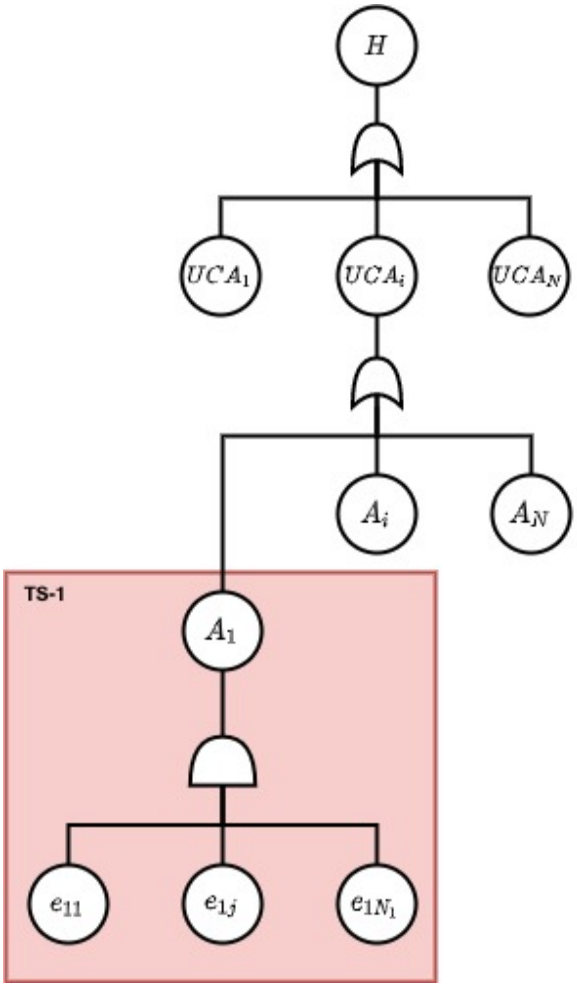


Figure 5.5: Example of threat scenarios tree.

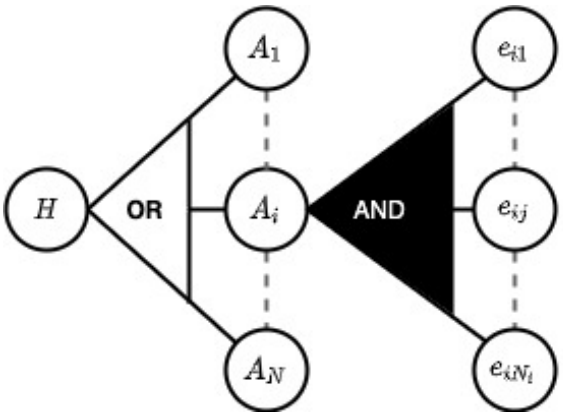


Figure 5.6: Example of threat scenarios tree - synthetic representation.

5.3.4 Implementation

We have implemented the attack enumeration process (Figure 5.2) in a JAVA tool featuring two main components: a query generator (*enquirer*) and a model checker service (*wrUPPAL*). The two processes exchange asynchronous messages using Java Message Service (JMS) [HBS+02] over Apache ActiveMQ [Fou]. The workflow of the program is outlined in Figure 5.7.

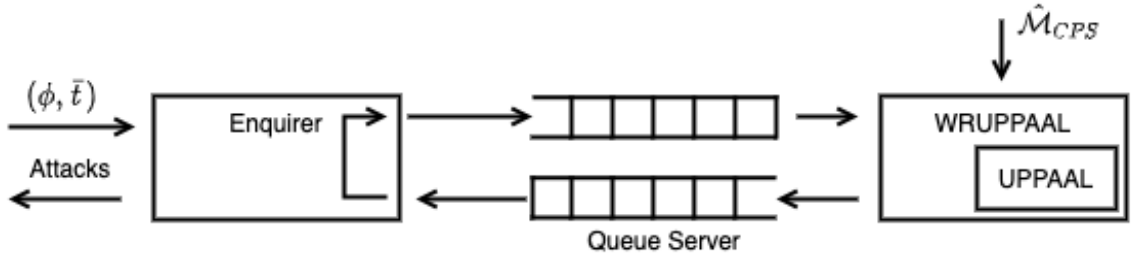


Figure 5.7: Implementation of attack enumeration procedure.

When the application is started, the model checking service is initialised loading the instrumented model $\hat{\mathcal{M}}_{CPS}$. The safety specification ϕ (TCTL query) is provided in input to the query generator by the safety analyst, along with $\bar{t}$. The latter is the time constraint which bounds the verification. The pair $(\phi, \bar{t})$ is sent over the queue to the model checker which checks if there is a trace π such that (ϕ) is verified in $\hat{\mathcal{M}}_{CPS}$. The results of the verification are sent back to the query generator which uses Algorithm 1 to compute the next query to enumerate attack scenarios. The process is repeated until all the generated queries have been checked against $\hat{\mathcal{M}}_{CPS}$; thus, the enumeration process is complete.

Given a safety specification ϕ and a time limit for the verification $\bar{t}$, the pair $(\phi, \bar{t})$ is sent to the *wrUPPAAL* service. When verification results are ready, the query generator extracts the sequence of adversarial actions $A_1 = e_{11}, e_{12}, \ldots, e_{1K}$ from the witness trace and computes the next query $\phi(A_1)$ until no more traces can be found (i.e., no more attacks are possible given the current implementation and threat model).

Our model checker service, *wrUPPAAL* implements a wrapper of the programmatic interface (APIs) available with UPPAAL [BDL+12]. Other than wrapping UPPAAL APIs, *wrUPPAAL* enriches its functionality by implementing a cache for the queries and a watchdog. Since model-checking is a memory-expensive and time-consuming process, having a dedicated cache

substantially reduces the time required for the enumeration of threat scenarios. The caching mechanism remembers the results from previous queries and avoids repeating the verification of queries that have been evaluated already. The presence of a watchdog, on the other hand, is key to limiting the verification in time during the exploration of a large sate-space. Verification bounds are, in fact, not supported by UPPAAL APIs.

The Enquirer (Figure 5.7) implements the attack enumeration algorithm described earlier (Algorithm 1) where each time a trace is returned, the enquirer extracts the respective sequence of adversarial actions $A_1 = e_{11}, e_{12}, \ldots, e_{1K}$ from the witness trace and computes the next query $\phi(A_1)$ until no more traces can be found i.e., no more attacks are possible given the current implementation and threat model.

5.4 Application to the communication-based train control scenario

We apply the process of automatic identification of threat scenarios to the communication-based train control (CBTC) system use case.

We start by applying STRIDE to the safe control structure of the CPS (Figure 2.4) and determine the Adversarial Actions (AA) in Table 5.1. In particular, we consider Spoofing, Tampering, ID and DoS threats against control action CA_1. For each tampering and spoofing threat, we instantiate an adversarial action for each significant[1] value to inject, in this case *allow* and *deny*. Similar threats affect CA_2 and CA_4. We assume that RTK messages are signed and cannot be tampered with or spoofed, but they can be disrupted (a_{14}). However, the communication with the GNSS can be spoofed. We introduce a dependency between information disclosure and tampering i.e., we assume that the communication protocol requires an attacker to read a message before modifying it. Three further tampering actions can be carried out against the process model variables held in the zone controller (a_1), and the location unit (a_2, a_{15}). Each adversarial action in Table 5.1 is mapped to one or more sets of privileges. Adversarial

[1]STPA provides values r for which a UCA is hazardous.

Table 5.1: Resulting attack library

#	**Object**	**Type**	**Value**	**Privileges**
a_1	ZC : Zone[i]	Tamper	$[0-1]$	(p_{10}, p_{11}, p_{12})
a_2	LU : GNSS	Tamper	$[1-5]$	p_{13}
a_3	CA_1 : Auth	Spoof	$[0-1]$	p_2
a_4	CA_1 : Auth	DoS	-	$(p_{14}), (p_8)$
a_5	CA_1 : Auth	Disclosure	$[0-1]$	$(p_{10}), (p_{11}), (p_{12}), (p_2)$
a_6	CA_1 : Auth	Tamper	-	$(p_{10}, p_{11}, p_{12}), (p_2)$
a_7	CA_2 : Zone	Spoof	$[1-5]$	p_7
a_8	CA_2 : Zone	DoS	-	$(p_{14}), (p_8)$
a_9	CA_2 : Zone	Tamper	$[1-5]$	$(p_3, p_6, p_5), (p_7)$
a_{10}	CA_2 : Zone	Disclosure	-	$(p_3), (p_6), (p_5), (p_7)$
a_{11}	CA_4 : Clear	Spoof	$[1-5]$	(p_7)
a_{12}	CA_4 : Clear	Tamper	$[1-5]$	$(p_3, p_6, p_5), (p_7)$
a_{13}	F_3 : GNSS	Spoof	$[1-5]$	(p_9)
a_{14}	F_2 : RTK	Dos	-	$(p_4), (p_1)$
a_{15}	LU RTKOn	Tamper	$true$	p_{13}

actions are associated with privileges in $\{p_1, \ldots, p_{14}\}$, which the adversary obtains through the exploitation of vulnerabilities in the system architecture (Chapter 6). These are usually flaws affecting components responsible for the exchange of control actions and feedback.

5.4.1 Synthesis of adversary behaviour

We systematically synthesise the model of adversarial actions (MAA) starting from the results of the application of STRIDE to the safe control structure. The model of adversarial actions (MAA), shown in Figure 5.8, is produced by applying the strategy discussed earlier in this section (Section 5.2). Initial and buffer locations are shown in blue, while locations corresponding to adversarial actions are shown with a white background. Transitions towards the execution of adversarial actions are shown in red, whilst those in blue return to the initial location. The colour coding only aims to aid the visualisation. There is no difference in terms of the model-checking semantics. In Chapter 7, we discuss the implementation of the model of adversarial actions, which we have built in UPPAAL to execute model-checking operations.

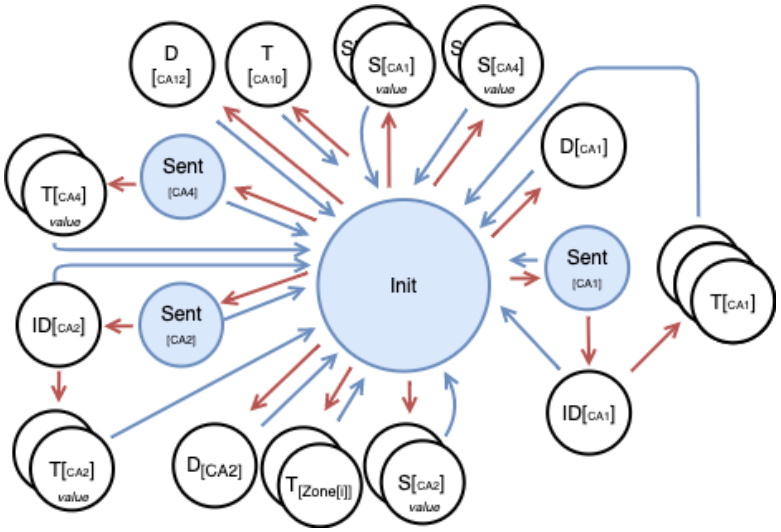


Figure 5.8: MAA for the CBTC use case.

5.4.2 Enumeration of threat scenarios

We proceed with the adversarial instrumentation and build the parallel composition $\hat{\mathcal{M}} = \mathcal{M}_{CPS}|\mathcal{M}_A$, which combines the safe behavioural model of the CBTC system with the model of adversarial actions. Thus, we use our algorithm defined (Algorithm 1) to enumerate threat scenarios that affect our implementation of the CBTC system.

In this section, we verify the safety specifications $\{\phi_1, \ldots, \phi_4\}$ defined earlier (Section 4.2) against the *instrumented model* of the CPS. The enumeration of threat scenarios carried out for this use case is extended in Chapter 7 with the verification of further safety specifications.

Enumeration process given a safety specification As an example, we now show perform enumeration of threat scenarios given the specification ϕ_2 (Section 4.2), which verifies the reachability of UCA_2 with our toolchain (Section 4.2). The unsafe control action UCA_2 leads to hazard H_1, thus a collision accident.

In the case of train id $i = 1$, ϕ_2 becomes.

```
φ2 = E<> (zones[1] == 2 and rbc_tid != 2 and loc[1] == loc[0] and RBC.
ReplyProvided and rbc_tmp_reply==1)
```

If it is reachable, the model checker returns a witness as an example of the reachability of ϕ_2.

The program extracts the sequence of adversarial actions (threat scenario) from the witness traces found and automatically builds and submits the next query, where it asks the model checker to verify whether ϕ_2 is reachable in any other way that is different from those already found. The process is repeated until no new threat scenarios are found. We show an extract of the output of our enumeration program in Listing 5.1. We report the complete output of the program for reference is reported in Chapter 7.

```
[User input] Type query: E<> (zones[1] == 2 and rbc_tid != 2 and loc[1] ==
   loc[0] and RBC.ReplyProvided and rbc_tmp_reply==1)


.......
.......
.......

[+] 6 attack traces have been found in 501.295s (8.354916666666666 min)
   (1) Read (Approach) ((-- AND --)) Tamper (Approach)
   (2) Disrupt (RTK Loc)  ((-- AND --)) Spoof (GPS Loc)
   (3) Tamper (LU.GNSSOn)  ((-- AND --)) Spoof (GPS Loc)
   (4) Spoof (Approach)
   (5) Tamper (ZC.Zones)
   (6) Spoof (Clear)

Removing redundancies...
   (1) Read (Approach)  ((-- AND --)) Tamper (Approach)
   (2) Disrupt (RTK Loc)  ((-- AND --)) Spoof (GPS Loc)
   (3) Tamper (LU.GNSSOn)  ((-- AND --)) Spoof (GPS Loc)
   (4) Spoof (Approach)
   (5) Tamper (ZC.Zones)
   (6) Spoof (Clear)
```

Listing 5.1: Execution of Threat Scenario Enumeration Algorithm

Enumeration of threat scenarios on multiple UCAs We provide an example of applications of our methodology by enumerating threat scenarios caused by unsafe control ac-

tions $UCA_1, \ldots, UCA_6$ (Section 4.2). This is achieved through the verifying specifications $\{\phi_1, \ldots, \phi_4\}$.

We show the outcome in Table 5.2. For each specification ϕ we report: the number of threat

Table 5.2: Attack traces violating safety specifications. Size of trace set (TS). Size of reduced trace set (RTS). Verification time (TE).

Hazard	Spec	TS	RTS	Threat Scenarios	TE
H_4	ϕ_1	1	1	$\{a_4\}$	$500.14s$
H_1	ϕ_2	6	6	$\{a_{10}, a_9\}$, $\{a_{14}, a_{13}\}$, $\{a_{15}, a_{13}\}$, $\{a_7\}$, $\{a_1\}$, $\{a_{11}\}$	$500.93s$
H_4	ϕ_3	1	1	$\{a_4\}$	$500.17s$
H_2	ϕ_4	2	2	$\{a_1\}, \{a_3\}$	$500.3s$

scenarios found (TS), the number of threat scenarios after elimination of redundant attacks (RTS), the respective hazards and threat scenarios and the time taken to verify each query ϕ_i and those derived from it i.e., the time elapsed (TE) (Table 5.2).

In Figure 5.9, we show the threat scenarios tree for hazard H_1 (Table 5.2), which we have uncovered through model checking ϕ_2. The hazard can be caused by the adversary which exploits, individually, the following threats

- Tamper with zone controller process model (a_1).
- Spoofing an approach request (CA_2) with an area code of a *free* zone (a_7).
- Spoofing a clear request (CA_4) with an area code of an *engaged* zone (a_{11}).

or through the combined exploitation of

- Information disclosure (read) and tampering with (write) approach request (CA_2) ($\{a_{10}, a_9\}$).
- Denial of service of RTK and spoofing GNSS ($\{a_{15}, a_{13}\}$).
- Tampering with Location Unit process model and spoofing GNSS ($\{a_{14}, a_{13}\}$).

The *completeness* of the algorithm was discussed in section 5.3. Our algorithm relies on UPPAAL to perform such operations, using symbolic verification along with a breadth-first search

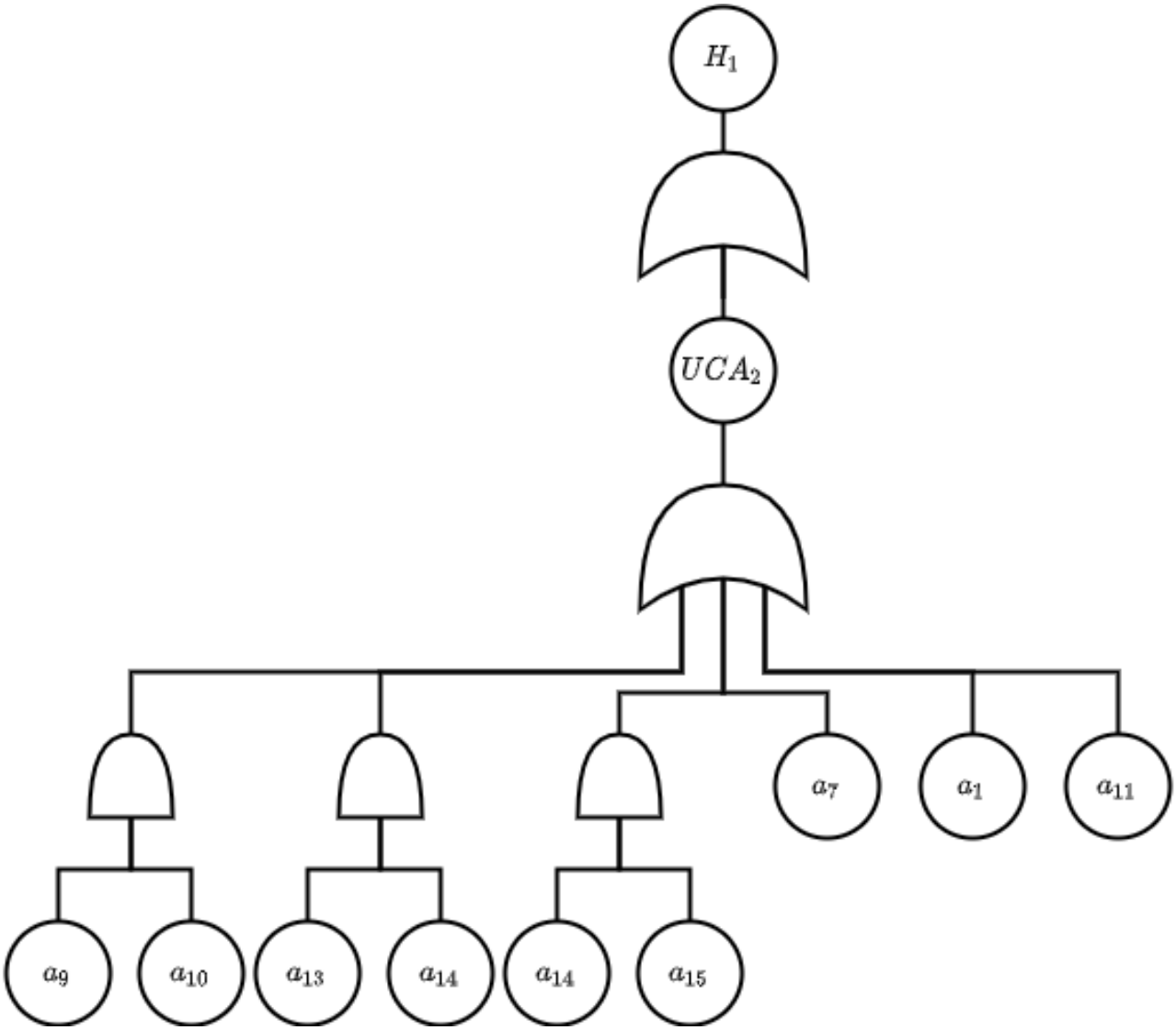


Figure 5.9: Threat scenarios tree (H_1)

approach to explore the state space to verify the reachability of queries. To limit memory explosion problems usually affecting model checking, we have limited the verification time to $\bar{t}$ seconds. In the worst case, our algorithm takes $n * \bar{t}$ seconds to execute, where n is the size of the queries in the queue. Space complexity depends on the size of the state space, which is directly handled by the model-checker.

5.5 Summary & Conclusions

In this chapter, we have presented a novel strategy that integrates safety and security analysis to identify and enumerate threats that can lead to hazards. The key points for the integration between the two analyses are the generation of a synthetic model of the attacker and the formal verification of the reachability of hazardous states on the parallel composition of the CPS and the constructed attacker model. We perform an exhaustive exploration of feasible threat scenarios.

The main limitation of our approach relates to its scalability and stems from the formal verification process which we use to identify and enumerate the safety-critical attacks. We have taken several steps to mitigate this which includes the use of a partial knowledge approach (section 4.3) to model the high-level behaviour of the CPS as well as limiting the analysis in time.

In the next chapter, we build a set of relations between threats and privileges in a deployed architecture of the CPS. By doing so, we can identify, via attack graph analysis, specific vulnerabilities the exploitation of which facilitates threat scenarios as well as quantitatively reasoning about the risk. The defender can then prioritise the remediation of relevant vulnerabilities (e.g., through patching, traffic isolation, etc.).

Chapter 6

Threat Scenarios Feasibility and Risk Analysis

In the previous chapter, we introduced our approach grounded in System Theoretic Process Analysis for security (STPA-Sec) and formal verification to automatically identify threats and sequences of threats that an adversary can harness to cause harm. At the same time, we have emphasised (Chapter 3) that a determination carried out on a safety model alone, such as STAMP, cannot provide insights in regards to the feasibility of threats nor can be relied upon to quantify the risk associated with threat scenarios in specific deployment contexts. In this chapter, we discuss steps ❷ and ❸ of Cassandra, where we address the gap between functional and architectural representations by computing and studying the attack paths leading towards the exploitation of safety-critical threat scenarios on the architecture of the cyber-physical system (Figure 6.1).

In particular, in ❷, we derive a system-wide attack graph leading to safety-critical threat scenarios. In doing so, we set the stage for the study of the viability of identified threat scenarios as well as for the understanding of the impact that individual vulnerabilities have with respect to the overall safety of the system. In addition to this, the use of attack graphs also allows for a quantitative estimation of the risk. Thus, in step ❸, we use exact Bayesian inference to analyse safety-critical attack paths.



Figure 6.1: Cassandra - Overview.

This chapter is organised as follows. In Sections 6.1 and 6.2 we introduce our threat model, which lies at the intersection of functional and architectural domains, and elaborate on the notion of safety-critical attack graphs. We discuss the use of safety-critical attack graphs to identify security measures to protect against potentially harmful attacks in Section 6.3. Thus, we elaborate on the use of Bayesian attack graphs to estimate the marginal probabilities of successful exploitation of threat scenarios in Section 6.4. Finally, Section 6.5 presents the conclusions of this chapter.

6.1 Threat model

In order to map the output from the threat scenarios analysis to the specific exploitation of vulnerabilities on system components, we need to ground the analysis in the specific deployment context. Our objective is the identification of vulnerabilities and weaknesses enabling threat scenarios, i.e., those attacks where the attacker can trigger *losses* through the causation of *accidents*. With this in mind, we have built the *Accident-Threat-Vulnerability* model, which ultimately relates low-level attack steps, such as the exploitation of a vulnerability, to hazards and their related accidents. Figure 6.2 shows the relations that we have established among the elements of the STAMP model, system-level threats, and low-level attack steps. The diagram extends the concepts presented in [MF14] with elements from the STAMP domain. In STAMP *losses* are defined as the consequences of *accidents*, which constitute the high-level target of *threat actors*; *threat actors* use *threats* to cause *accidents*. From a functional perspective, *threats* affect the *control actions*, *feedback signals* and the *process model*. The unsafe application of *control actions* leads the system into a hazardous state, where *accidents* can manifest. From an architectural point of view, *threats* are facilitated by the exploitation of vulnerabilities and

weaknesses affecting system components (e.g. credentials, network access, etc.), which lead the adversary to obtain the *privileges* to *spoof/tamper/read/delay/drop* the information flow of the CPS and thus cause the unsafe application of *control actions*. This application can be through direct actions, e.g. through compromising *control actions*, *feedback* or *process models*, or through cascading effects. The representation shown in Figure 6.2 plays a key role in linking

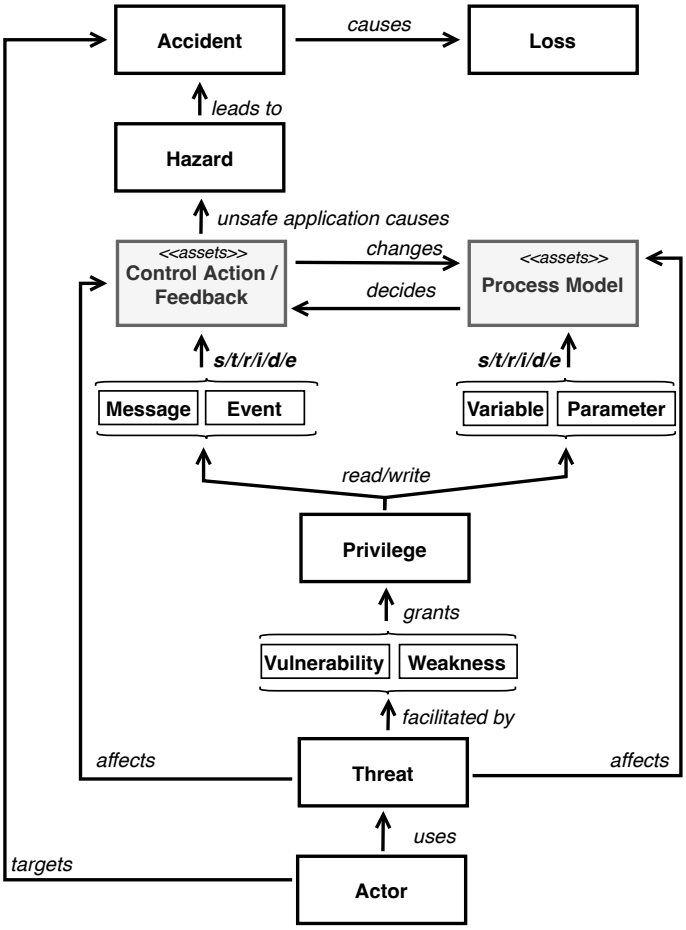


Figure 6.2: Accident-Threat-Vulnerability model

specific vulnerabilities to malicious actions, *accidents* and *losses*. Security engineers can define adversary goals in terms of the system level *losses* and *accidents*. Our approach provides them with the tools to identify the *privileges* required to trigger a specific *accident* and to find the respective attack paths. Security engineers and operators can then address the remediation of these paths.

6.1.1 From hazards to privileges: derivation of safety-critical vulnerabilities

In this section, we formalise the relations among privileges, threats, and hazards established by the Accident-Threat-Vulnerability model (Figure 6.2).

Given the hazard H, the set of unsafe control actions (UCA_1, ..., UCA_N), the set of safety specifications ($\phi_1, \ldots, \phi_M$), and the instrumented model of the CPS $\hat{\mathcal{M}}_{CPS}$, we assume that each specification ϕ_i holds true if and only if unsafe control action UCA_i, $i \in N$ is reachable. We expressed the case using the syntax

$$\hat{\mathcal{M}}_{CPS}, \Pi_i \models \phi_i (i \in M)$$

with $\Pi_i = \{\pi_{i1}, \ldots, \pi_K\}$ being the set of K traces for which ϕ_i is true (Chapter 5).

In the previous chapter, we have also defined the generic adversarial action, i.e., instance of threat, as the tuple

$$a_k = (t, obj, r, privileges)_k$$

In particular, the attribute *privileges* holds a number of privileges organised in sets. The attacker needs to acquire **all the privileges** in at least one of the sets to be able to execute a_k. We recall that a_k is an instance of a threat of threat-type t against the element of the control hierarchy obj and with parameter r.

Let p be a privilege in the system architecture, we call P its associated Boolean variable. P is *true* following a low-level security event, i.e., the successful exploitation of a vulnerability or a weakness, that allows the adversary to acquire p. From a logic standpoint, the capability of an adversary to perform the k-th adversarial action can be expressed in the disjunctive normal form (DNF) starting from the attribute $privileges_k$. In fact, in order to execute a_k the attacker needs to acquire **all the privileges** (AND) in **at least one of the sets** (OR) in $privileges_k$. We employ this formulation to identify the privileges of the system architecture the adversary needs to acquire to facilitate hazards.

First, we define the following Boolean predicate which holds *true* if the attacker acquires the object x.

$$attacker(x) \tag{6.1}$$

With this in mind, given the hazard H, we can define the predicate $attacker(H)$ as

$$attacker(H) := \bigvee_{i=1}^{N} attacker(\phi_i) \tag{6.2}$$

where $attacker(\phi_i)$ is *true* if the attacker can cause a violation of the safety specification ϕ_i. In other words, $attacker(\phi_i)$ is *true* if they can execute at least one attack A_{ij} in π_{ij} with $\pi_{ij} \in \{\pi_{i1}, \ldots, \pi_{M_i}\}$ being the j-th witness trace resulting from the verification of ϕ_i and $j \in (0, M_i]$.

We have defined (Section 5.2.1) the attack library $AL = \{a_1, a_2, ..., a_K\}$ as the collection of size K of adversarial actions gathered during the threat modelling phase. An attack A_{ij} is a finite ordered sequence of adversarial actions, such that $A_{ij} \in AL^*$. We denote with K_{ij} the length of the attack A_{ij}, and its component adversarial actions with $A_{ij} = (e_{ij1}, e_{ij2}, ...e_{ijK_{ij}})$, with $e_{ijk} \in AL^*$. The capability of an adversary to perform all the actions in A_{ij} for at least one $j \in M_i$ and one $i \in N$ empowers them the execute the attack and cause the hazard H.

Given the adversarial action e_{ijk} with privilege sets $(Q_1, \ldots, Q_{Nijk})$, the predicate $attacker(a_{ijk})$ is *true* if the attacker can execute the action, which is when they have acquired all the required privileges (i.e., all the privileges in at least one of its sets of privileges). Hence, $attacker(a_{ijk})$ is defined as

$$attacker(a_{ijk}) := \bigvee_{q=1}^{N_{ijk}} attacker(Q_q) = \bigvee_{q=1}^{N_{ijk}} \Big(\bigwedge_{p=1}^{N_{ijkq}} attacker(p_{ijkqp}) \Big) \tag{6.3}$$

Given the attack $A_{(ij)} = \{e_{ij1}, e_{ij2}, \ldots e_{ijK_j}\}$, the predicate $attacker(A_{ij})$ defined in Equation

6.4 is *true* when the attacker can perform the attack A_{ij}.

$$attacker(A_{ij}) := \bigwedge_{k=1}^{K_j} attacker(e_{ijk}) \tag{6.4}$$

To be able to cause hazard H, the attacker needs to be able to execute at least one attack trace in ϕ_i. We define the predicate $attacker(\phi_i)$ in function of its attacks

$$attacker(\phi_i) := \bigvee_{j=1}^{M_i} attacker(A_{ij}) = \bigvee_{j=1}^{M_i} \bigwedge_{k=1}^{K_j} attacker(e_{ijk}) \tag{6.5}$$

We can now rewrite (6.2) with the results from (6.5)

$$attacker(H) := \bigvee_{i=1}^{N} attacker(\phi_i) = \bigvee_{j=1}^{M_i} \bigwedge_{k=1}^{K_j} attacker(e_{ijk}) \tag{6.6}$$

Finally, by combining the results from (6.6) and (6.3), we can rewrite (6.2) as a function of the privileges that are needed to cause it.

$$attacker(H) = \bigvee_{i=1}^{N} \bigvee_{j=1}^{M_i} \bigwedge_{k=1}^{K_j} \bigvee_{q=1}^{N_{ijk}} attacker(Q_{ijkq}) = \bigvee_{i=1}^{N} \bigvee_{j=1}^{M_i} \bigwedge_{k=1}^{K_j} \bigvee_{q=1}^{N_{ijk}} \bigwedge_{p=1}^{N_{ijkq}} P_{ijkqp} \tag{6.7}$$

in fact

$$attacker(Q_{ijkq}) := \bigwedge_{p=1}^{N_{ijkq}} attacker(p_{ijkqp}) = \bigwedge_{p=1}^{N_{ijkq}} P_{ijkqp}$$

6.1.2 Threats-Privileges mapping

A key aspect of the Accident-Threat-Vulnerability model is the map between threats to elements of the control hierarchy - that we have also defined as adversarial actions - and set(s) of privileges that the adversary needs to steal to carry them out. In our work, we have constructed these relations manually, relying on two assumptions. First, we assume that the architecture of the cyber-physical system *is known*, which includes the network topology and also contains information on the hosts, e.g., hardware model, software version, etc. (Chapter 3). Then,

we also assume that a security assessment has been carried out to identify the vulnerabilities and weaknesses that affect the different components and sub-networks and the privileges that such vulnerabilities would grant to an adversary following their successful exploitation. Network scanning (e.g., Nmap [Lyo08]) along with vulnerability assessment (e.g., OpenVAS[Gre], Nessus [Ten]) tools can be employed in production environments to assist with the description of system structure and the discovery of vulnerabilities.

Relations between adversarial actions and privileges required to exploit them can, in some cases, be identified automatically by leveraging existing relations between the safety model of the CPS and its architecture. For example, the tampering of a message requires the adversary to achieve *read* and *write* privileges on the communication medium. These are linked to the exploitation of eventual weaknesses and vulnerabilities in the authentication and integrity mechanisms actuated by the protocol, as well as vulnerabilities affecting hosts involved in the communication. Likewise, the exploitation of a vulnerability that affects a program running on a host can also grant the privileges to execute adversarial actions if the host implements a component in the safe control structure. More generally, existing maps between CAPEC [MIT23a] and STRIDE, as well as between CAPEC, CWE and CVE, can be used to automate the process. However, investigating the means of producing this map automatically is left for future work.

6.2 Safety-critical attack graphs

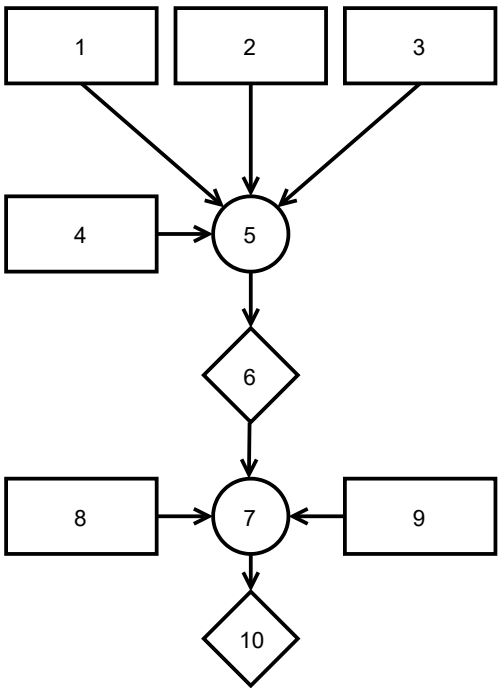
The attack graph of the cyber-physical system encompasses all attack paths starting from the system perimeter - usually the points of contact between the IT network and the external world, e.g., the Internet - and leading to a set of objectives, which we express as privileges $\mathcal{P} = \{p_1, \ldots, p_M\}$ required by the attacker. These are the privileges that directly map to adversarial actions and are often on the operational technology (OT) side.

Cassandra interfaces with MulVAL to generate system-wide attack graphs. MulVAL employs the deduction engine XSB [SSW94] to build the attack graph of the system starting from a set of

assumptions on the adversary's objective and initial position, given the system configuration, and a set of exploitation rules [OGA+05]. The result is a directed tripartite graph $AG = (V, E, \mathcal{L}, \mathcal{G})$ with vertex set V and edge set E, which shows all the paths an adversary could take towards the objective from an initial position. In particular, $V = V_p \cup V_r \cup V_d$ and V_p, V_r and V_d, respectively, represent primitive facts, derivation rules and derived facts.

- Primitive facts (PFs) are simple facts about the system which are *true* when the security assessment is performed. These include the existence of a vulnerability, the connectivity of a host to a sub-network, a weak configuration, etc..
- Derived facts (DFs), on the other hand, are facts which are non-trivial and are inferred by the application of Derivation Rules. From a security engineering perspective, DFs usually represent privileges obtained following a lateral movement, i.e., exploitation of a weakness or a vulnerability.
- Derivation rules (DRs) describe how PFs and DFs combine into the execution of the lateral movements mentioned above. From a logical standpoint, Derivation rules are Horn clauses [Hor51].

Being simple (and always *true*) predicates, primitive facts are characterised by an empty *in-edge set*, i.e., they have no parents. DR nodes, on the other hand, represent **logical and**. Edges in the *in-edge* set of a derivation rule are considered **and** with each other; in fact, they represent the preconditions the adversary must meet to perform the lateral movement (e.g., Horn clause true). DR nodes only admit PF and DF as their parents. At the same time, DF nodes represent **logical or**. Edges in the *in-edge* set of a derived fact are considered in *or* with each other; in fact, different paths can lead to the same privilege (DF). DF nodes only admit parents of type DR. $\mathcal{L}$ and $\mathcal{G} \subseteq V_d$ are the sets of *labels* and adversary *goals*. For each *adversarial action*, one or more sets of privileges (DF nodes) are specified in the actions library. The adversary needs to satisfy at least one of the set of privileges to successfully engage in the respective adversarial action. From a defender perspective, each of the sets of nodes associated with a specific adversarial action enables us to find out which are the *attack paths* leading to its execution.



Label	Description
1	*attackerLocated(subnet)*
2	*hacl(host, subnet)*
3	*vulnExist(host, s, cve-x, remote)*
4	*networkService(host, user, subnet, s)*
5	*remoteExploitation, cve-x*
6	*hasAccess(host, user)*
7	*localExploitation, Exploitation, cve-y*
8	*localService(host, user, p)*
9	*vulnExist(host, p, cve-y, priv-escalation)*
10	*hasAccess(host, root)*

Figure 6.3: Example of logical attack graph

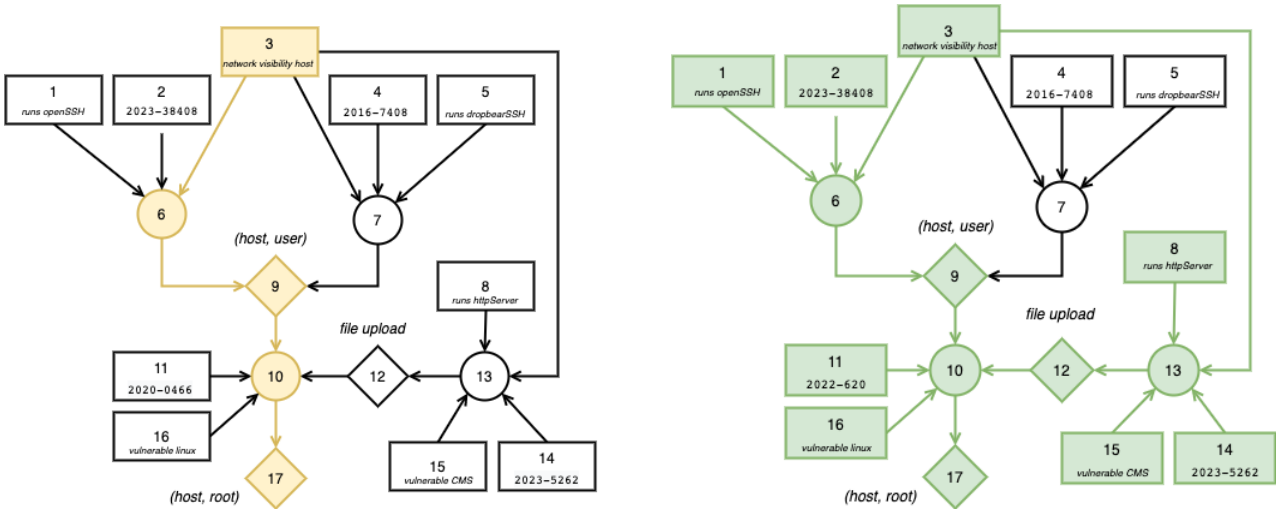
In Figure 6.3, we show an example of a logic attack graph which uses the definition employed by MulVAL. The AG is rooted in 1 while we have assumed that the objective of the attacker is the privilege in node 10. The adversary has direct access to a *database* software hosted by *host*. The program runs with *user* privileges and is affected by a known vulnerability `CVE-X` (node 3). Through the exploitation of `CVE-X`, the attacker obtains a *user* shell on *host* which can leverage to exploit `CVE-Y` (node 9) and escalate privileges. The Boolean nature of logical attack graphs makes them well-suited for performing the selection of countermeasures based on satisfiability. As an example, in Figure 6.3, we observe that it is possible to prevent the attacker from reaching node 10 by promptly patching `CVE-Y` (node 9), by shutting down the vulnerable service p (node 8), or, alternatively, by preventing them to reach node 6. This is because node 10 is defined as an *AND* among nodes 6, 8, and 9. Likewise, we can defend 6 by patching `CVE-X`, by removing the vulnerable software s, or by tearing down the connection with *subnet*. In fact, any of the three actions would render node 6 false as per the rule in node 5, which would also falsify node 10.

The attack graph in Figure 6.3 contains just one attack path. This is because the *graph path* between node 1, adversary's entry point, and node 10, the assumed objective, includes two **or**

nodes (6, 10) which both have *in-degree* equal to 1.

Attack paths and graph paths To improve the clarity of the exposition, we introduce a distinction between *graph paths* and *attack paths*. With the expression graph path, we refer to any path between two nodes on the attack graph. On the other hand, an attack path is a graph path between two nodes in the AG, where all *and* nodes are satisfied, i.e., it includes those facts (derived or primitive) that satisfy all *and* dependencies in the graph path. An attack path is *complete* when all the *and* nodes (derivation rules) are satisfied for all non-primitive facts. In the context of a logical AG, graph paths do not necessarily coincide with attack paths and, generally, the former is a subset of the latter. We use graph paths to denote graph reachability between two nodes, whereas attack paths are those paths where the *and/or* are satisfied.

Given a graph path, we can derive the attack path associated by adding the parents of each *and* node (derivation rule) to the path. In fact, the associated attack path must contain **all parents** for each *and* node in it. In order to *complete* the attack path, for each *or* node (derived fact) in the graph path we also add *any of its parents* to the associated attack path. This can lead to new *unsatisfied and* nodes, which also need to be addressed. The operation is repeated until the following conditions are met: for each *or* node, the attack path contains any of its parents, and, for each *and* node, the attack path contains all of its parents.



(a) Graph path **(b)** Complete attack path

Figure 6.4: *Attack path* and *graph path* compared on the same attack graph.

We provide a graphical highlight of the differences between graph paths and attack paths in Figure 6.4. In the figure, the graph path is highlighted in yellow (left), whereas the attack path is shown in green (right). As an entry point, we assume that the adversary has network visibility of a certain *host* machine (node 3) and that their objective is to gain *root* privileges on *host* (node 17). On the left, we portray a possible graph path between nodes 3 and 17. The existence of at least one graph path reveals node 17 is reachable, i.e., the attack is potentially feasible; however, it does not report a full account. To do so, we need to compute an associated attack path as follows. Initially, we assume that the attack path coincides with the attack graph, thus $\mathcal{A}'_p = \{3, 6, 9, 10, 17\}$. Node 6 brings nodes 1 and 2 to the attack path; similarly, node 10 comes with $11, 12$, and 16. Thus, $\mathcal{A}'_p = \{1, 2, 3, 6, 9, 10, 11, 12, 16, 17\}$. Finally, we compute the complete attack path by also considering any parent of 12 (just one in this case) along with its dependencies. We repeat the process until all parents of **and** nodes in the attack paths are also in the attack path, and until for each *or* node in the attack path, any of their parents is also in the attack graph. The complete attack path is then $\mathcal{A}_p = \{1, 2, 3, 6, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17\}$ (Figure 6.4b).

6.2.1 Graph generation

The process of attack graph generation with MulVAL is shown in Figure 6.5. It takes *four* inputs: the threat scenarios, the attack library, the system architecture, and the related security properties. We use the adversarial actions library (also shortened as *Attack Library* in Figure 6.5) to identify the privileges leading to the threat scenarios which were discovered during step ❶ of Cassandra. The privileges found are the potential objectives of the adversary.

At the same time, we produce a description of the *architecture of the CPS*, which includes a list of the hosts, their configurations, and how they are connected. This is provided in XSB Prolog, as required by MulVAL. Security properties such as vulnerabilities and weaknesses also need to be provided as input to MulVAL. We merge in a single script file (*SystemDescription.pl* in Figure 6.5) the description of the architecture of the system and its security properties along with the potential objectives of the adversary.

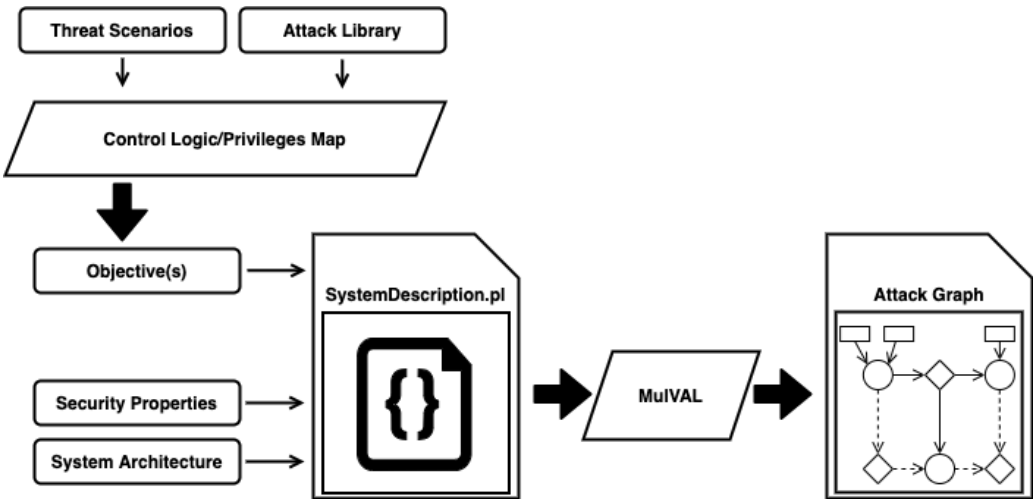


Figure 6.5: Derivation of Attack Graph

Given the *SystemDescription.pl* file, MulVAL produces a system-wide logical attack graph that contains all the paths leading to safety-critical privileges.

6.2.2 AGTools library

We have developed *AG-Tools*, a Java library which implements our custom algorithms for manipulations of logical and Bayesian attack graphs. The main classes in this library are *BayesianAG* and *MulVALAttackGraph*; they both inherit from a common interface *AttackGraph* (Figure 6.6). The actual attack graph is represented using a directed weighted graph data structure which we import from JGraphT [MKNS20]. We define a common interface *AGNode* to represent graph vertices. We use classes *MAGNode* and *BAGNode*, which implement *AGNode* to respectively encode MulVAL and Bayesian nodes. Finally, we use JGraphT own *DefaultWeightedEdge* to represent edges in the MulVAL graph. We implemented a custom *BAGEdge* class which inherits from *DefaultWeightedEdge* to encode vertices in the Bayesian networks. Graph manipulation and transformation algorithms described in the paragraphs below are implemented as methods of *BayesianAG* or *MulVALAttackGraph*.

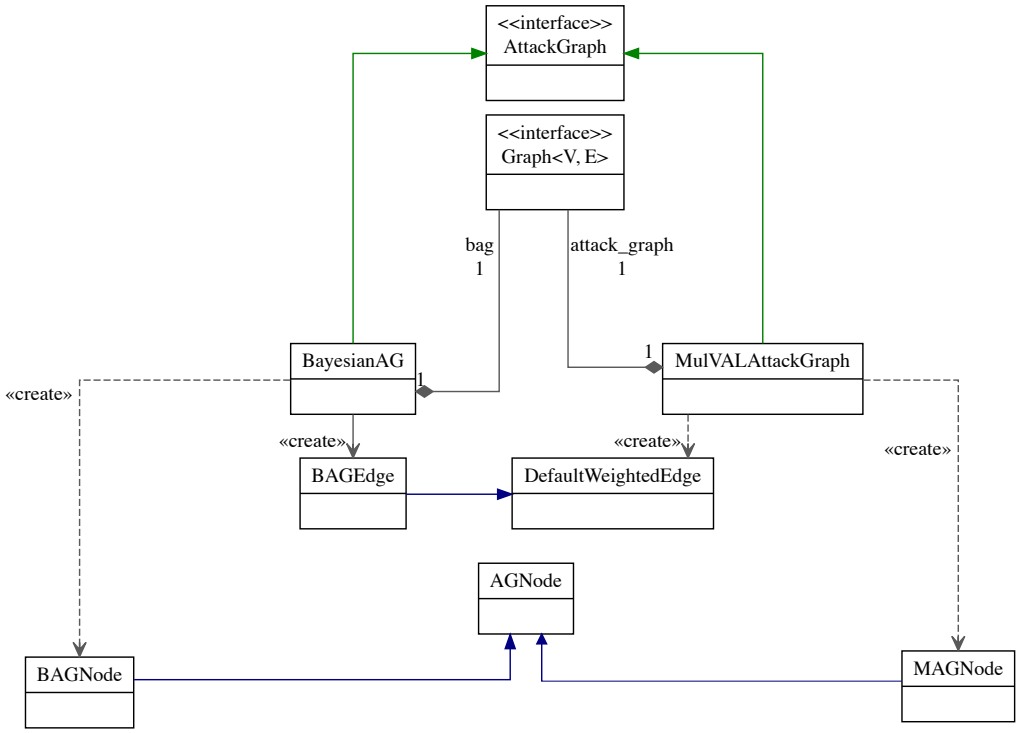


Figure 6.6: UML Diagram AGTools

6.2.3 Extraction of directed acyclic subgraph

Attack graphs such as those produced by MulVAL admit the existence cycles. Cycles can appear independently from the network topology and lead to a steep increase in the number of attack paths found, which has been estimated to be polynomial in the number of nodes in the AG [AJN12]. More generally, the presence of cycles significantly complicates the analysis of attack graphs [HOM08, WIL+08]. Authors of [WIL+08] classify three types of cycles: *removable*, *breakable*, and, *unbreakable*. They show that cycles in the first two classes can be easily removed or broken (i.e., disconnected) without impacting the overall security metrics of the attack graph, e.g., connectivity and total number of attack paths. On the other hand, *unbreakable* cycles cannot be broken without inevitably impacting said metrics.

We have developed our own routine to detect and break cycles in attack graphs produced by MulVAL, thus deriving the directed acyclic subgraph in a MulVAL graph. It was implemented as part of the *MulVALAttackGraph* class of *AGTools* and works by enforcing the monotonic assumption on attack paths; in other words, we assume that the attacker never releases privileges, which is reasonable in the overwhelming majority of cases [MGSPL17].

We use Kosaraju's algorithm [CLRS22] to identify strongly connected components (SCS) in the graph. We define the *head* of the SCS as any node (in the SCS) closer to the entry point of the attacker. Thus we break the cycle by pruning *returning* edges. Namely, let (u, v) be an edge between nodes u and v in the SCS and let v be the head; we remove (u, v) if $distance(r, v) \leq distance(r, u)$ where r is the entry point of the attacker. We repeat the process until all strongly connected components have been broken. Kosaraju's algorithm to identify SCS performs two traversals of the AG, showing a linear time complexity $O(V + E)$.

Extracting a DAG from a MulVAL graph output brings two main advantages to our analysis. First, it significantly reduces the overall number of paths, de facto easing the heavy path extraction process discussed in the section below. Besides, by breaking cycles, we set the stage for using Bayesian networks to perform risk analysis (Section 6.4). However, the operation is not lossless and can potentially hide a negligible number of attack paths in the original graph whenever an *unbreakable cycle* is disconnected [WIL+08].

6.3 Selection of security controls

In step ❸ of Cassandra, we exploit the structure of logical attack graphs to identify critical attack paths and select those vulnerabilities $v_i \in V_c \subseteq V$ whose remediation would defend from safety-critical attacks.

We start by introducing the notion of *minimal safety set* (MSS), namely, the minimal set of privileges to protect in order to prevent the adversary from causing accidents. We use MulVAL to generate the attack graph of the CPS and propose an algorithm to enumerate *complete attack paths* leading to privileges in the MSS (Algorithm 3). The algorithm is complete and enables us to list attack paths and easily discover key vulnerabilities whose remediation can protect $\mathcal{S}$. However, its complexity does not scale well with the depth of the attack graph and we will discuss how its application might not be feasible in some cases. We elaborate on possible improvements to the enumeration algorithm that combine *graph paths* and *attack paths* in a more efficient approach, trading off completeness for scalability (Section 6.3.2). We apply the

proposed algorithms to our run-through example, where we compute the intersection of the attack paths and select the subset of vulnerabilities $v_i \in V_c \subseteq V$ common among the privileges in the minimal safety set $\mathcal{S}$ of the CPS (Section 6.3.3). In doing so, we can identify potential remediation to defend from safety-critical attacks.

6.3.1 Minimal Safety Set

Earlier in this thesis we have shown that, given the hazards H, our methodology can enumerate threat scenarios $A_1, A_2, ..., A_N$ leading to H. Given a set of adversarial actions $AL = \{a_1, a_2, ..., a_K\}$, a threat scenario can be caused by one or more ordered sequences of adversarial actions. To cause harm, the adversary must execute an entire sequence of adversarial actions contained in at least one scenario A_i.

Formally, let $\{p_1, \ldots, p_M\}$ be the set of privileges that map to the adversarial actions in AL and $\{P_1, \ldots, P_M\}$ be a set of Boolean variables, such that P_i is true if the attacker has acquired the privilege p_i. We have shown (Section 6.1.1) that we can write the hazard H as a function of the $P_1, \ldots, P_M$ as follows

$$\mathcal{H}(P1, \ldots, P_M)\{0,1\}^M \rightarrow \{0,1\}$$

where $\mathcal{H}$ is a Boolean function that evaluates to *true* for those combinations of privileges that enable the attacker to cause the hazard. We provide a graphical representation of this relation in Figure 6.7.

We denote with $\mathcal{S}$ the minimal safety set(s), i.e., the minimal set of privileges that falsifies $\mathcal{H}$. We compute $\mathcal{S}$ during the second step of Cassandra. If the defender can remediate the vulnerabilities leading to privileges $p_i \in \mathcal{S}$, P_i is always *false* and the attacker cannot cause H.

Discovery and enumeration We have developed an enumeration algorithm, which uses Boolean satisfiability to compute the minimal safety sets given a Boolean function $\mathcal{H}$. We

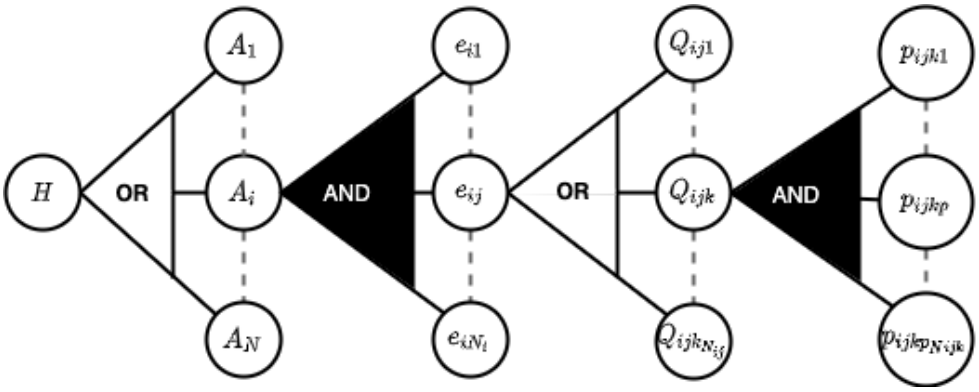


Figure 6.7: Hazards in functions of privileges.

have implemented it as a program which employs Microsoft Z3 [MB08] solver and expects the following data as input

- System level hazard H, and the respective Boolean function $\mathcal{H}$.
- Threat scenarios tree.
- Adversarial actions library (AAL), which encodes relations adversarial actions and privileges.

We have discussed how the function $\mathcal{H}$, the threat scenarios tree, and the adversarial actions library are identified within the application of Cassandra. Our implementation returns all the combinations of privileges that, if protected (i.e., the respective Boolean variable P *false*), would prevent the attacker from achieving H. Among these, we select the K combinations $\mathcal{S}_1, \ldots, \mathcal{S}_K$ of the smallest size, i.e., with a smaller number of privileges. The choice of returning the smallest set(s) was made with the assumption that the size of a set is proportional to the effort required to *protect* it. Although there are instances where this assumption might not hold, e.g., if a fix is not available or it is associated with high costs, we describe here the case when sets are selected based on their size.

The algorithm employed by our program is shown in Algorithm 2 and is conceptually similar to the algorithm shown in the previous chapter to enumerate threat scenarios (Algorithm 1). In the setup phase, we first declare Boolean variables corresponding to privileges in the CPS architecture (lines $2-5$). Then, we use the relations among hazard, threat scenarios, and losses

Algorithm 2: Algorithm for MSS enumeration

Data: System-level *Hazard*, threat *scenarios*, adversarial *actions*, *privileges*
Result: Solutions *solutions*

```
/* Define a Boolean variable for each privilege ... */
for p_i in privileges do
    P ← Boolean(p_i) ;
    privileges_as_bool.append(P);
end

/* Define actions as AND/OR combinations of privileges ... */
for a_i in actions do
    A ← And(Or([p_1, ..., p_{K_i}]));      p_i ∈ privileges_as_bool
    actions_as_bool.append(A);
end

/* Define threat scenarios as AND/OR combinations of adversarial actions ... */
for scenario in scenarios do
    TS ← And(Or([a_1, ..., a_{K_i}]));      a_i ∈ actions_as_bool
    scenarios_as_bool.append(A_i);
end

/* Define hazard as AND/OR combinations of actions ... */
H ← And(Or([TS_1, ..., TS_{K_i}]));      TS_i ∈ scenarios_as_bool

s.add(H);
res ← s.check();
counter ← 0;
while res = sat do
    m ← s.model();
    block ← [ ];
    solution ← [ ];
    for privilege in privileges_as_bool do
        p ← m.evaluate(privilege, model_completion = True);
        block.append(privilege ≠ p);
        solution.append(p);
    end
    solutions.append(solution);
    s.add(Or(block));
    counter ← counter + 1;
    res ← s.check();
end

return solutions
```

encoded in the threat scenarios tree to represent adversarial actions as *and/or* combinations of privileges (lines $7-10$). Similarly, we formalise threat scenarios as *and/or* combinations of adversarial actions (lines $12-15$). Then, we encode the function $\mathcal{H}$ as *and/or* combination of threat scenarios and pass it to the solver. We enumerate suitable combinations of privilege values by re-iterating the evaluation of the satisfiability of $\mathcal{H}$; for each new enumeration, we add previous solutions as constraints to force the solver to find a new solution. This strategy reflects the approach that we employed for enumerating threat scenarios (Section 5.3).

Starting from the solutions. we perform a simple selection of the minimal safety set(s) by counting the occurrences of *false* Boolean variables in the solutions provided by the algorithm in Algorithm 2. We now discuss how the notion of MSS can be successfully applied to identify security controls applicable to the wide system architecture (which includes both IT and OT infrastructure) (Section 6.3.2). Such mitigating actions aim to prevent the application of safety-critical threat scenarios through the protection of key attack paths.

6.3.2 Extraction of paths leading to Minimal Safety Set

Given the logical attack graph $AG = (V, E, \mathcal{L}, \mathcal{G})$ and the minimal safety set $\mathcal{S}$, we propose Algorithm 3 to enumerate attack paths leading to privileges in the $\mathcal{S}$.

We provide, as input, a set of objective nodes in the graph; in our settings, these are the privileges in the minimal safety set. Starting from these objectives, we perform a backward traversal of the graph to identify and enumerate those attack paths that start from the perimeter of the CPS and end to the objectives provided as input. We execute the depth-first traversal according to the following rules:

- In the first iteration, we instantiate *one path* for every of the different *objective* provided as input. At this stage, each path contains only one node - its objective.
- For each path, our algorithm tracks the pair (*path index*, *fork index*). The former provides global identifiers for the path. The *fork index* is the next node to be processed on the attack graph.

- We leverage the *fork index* to select the *current node*, i.e., the next node to process on the AG - this coincides with the objective in the first iteration. We select the *current node* on the attack graph to identify its parents.
  - if *current node* is a *and* node, i.e., *a derivation rule*, then *all parents* are added to the current path.
  - if *current node* is a **or** node, i.e., *a derived fact*, then we add any parent to the *current path* and create $n-1$ forks of the current path, where n is the number of parents of the current node. We distribute one parent to each fork. New forks are added to the list of paths.
- The backward traversal of each path is carried out until all the *parents* of the current nodes have in-degree equal to 0, i.e., they are *primitive facts*.

An algorithm similar to the one we have proposed in Algorithm 3 was also independently proposed by [Zen22]. However, authors of [Zen22] draw a generic pseudo-code for path extractions whereas we propose a working implementation based on the formalism output by MulVAL.

Identification of security controls Our proposed algorithm (Algorithm 3) enumerates the *complete* attack paths leading to one or more objectives, e.g., the privileges in the MSS. We identify security controls by looking at the intersection of the attack paths found. In particular, we select *primitive facts* nodes that are common to all - or most of - the attack paths. In our use case (Section 6.3.2), we show that it is possible to prevent the reachability of safety-critical attacks by acting on these common nodes. Candidate actions include the fixing of vulnerabilities and the disconnection of hosts.

Time Complexity and scalability Given the enumerating nature of the algorithm, its time complexity can be expressed as $O(B^d)$. More specifically, it is a $O(B_{OR}^d)$, with B_{OR}^d being the branching factor of *or* nodes and d the depth of the attack graph. From a logic reasoning standpoint, it implements an exhaustive search approach to a Boolean satisfiability problem, which is known to be NP-complete.

Algorithm 3: The algorithm returns all paths in $\mathcal{AG}$ leading to privileges in Objectives

```
Data: Objectives
Result: Paths
paths ← [ ] ;
pathIndex ← 0 ;
foreach objective in objectives do
    path ← objective;
    forkIndexes.add(i, 0) ;
    paths.add(path);
    i ← i + 1;
end
while pathIndex < size(paths) do
    currentPath = paths[pathIndex];
    nodeIndex = forkIndexes[pathIndex];
    while nodeIndex < size(currentPath) do
        currentNode ← path[nodeIndex];
        parents ← [ ];
        foreach parent in parents(currentNode) do
            parents ← parent;
        end
        if currentNode is Derived Fact then
            currentPath.add(parents[0]) ;
            parents.remove(parents[0]) ;
            foreach parent in parents do
                fork ← path[0, nodeIndex];
                fork.add(parent);
                paths.add(fork);
            end
        end
        if cuurentNode is Derivation Rule then
            foreach parent in parents do
                if parent ∉ currentPath then
                    currentPath.add(parent) ;
                end
            end
        end
        nodeIndex ← nodeIndex +1 ;
    end
    pathIndex ← pathIndex +1 ;
end
```

Limiting the depth of the search Thus, the application becomes unfeasible as the depth of the attack graphs grows. A trivial solution consists of limiting the depth of the search. In this direction, we have developed a modified version of Algorithm 3 which explores each path for a limited number of steps. The exact number of nodes to explore in the limited backward search is found empirically and varies depending on the topology of the graph. This approach can hide vulnerabilities that are common to two or more privileges in the MSS and are far apart from each other in the graph. However, in attack graphs characterised by a high degree of clustering, we can still obtain insights that are useful for our purposes. Given the clustered nature of OT networks, it is likely that a subset (if not all) of the privileges in the MSS are located in the same sub-network and significant common nodes can be found at short depth.

Graph path aided search We use the variation of the Dijkstra algorithm proposed in [MKNS20] to find all the (directed) *graph paths* from the assumed entry point of the attacker to privileges in the MSS. For each path, we analyse the number of AG nodes that are common among all paths and identify the hub nodes $\{p_1^*, \ldots, p_H^*\}$ which were not in the MSS but need to be traversed to reach the MSS. We then apply our reverse enumeration algorithm to find all the attack paths leading to $\{p_1^*, \ldots, p_H^*\}$. This is feasible in the case of $\{p_1^*, \ldots, p_H^*\}$ being closer to the perimeter than the privileges in the initial set $\{p_1, \ldots, p_{14}\}$.

6.3.3 Example of application

In the previous chapters, we have described the application of off-the-shelf STPA to our communication-based train control (CBTC) system. Namely, we have found that a *collision* event between two trains can occur in the context of hazards H_1, H_3, and H_6 (Section 2.6). Thus, in step ❶ of Cassandra we have determined the related threat scenarios, i.e., instances of threats that an adversary can exploit to trigger a change of state in the CPS where a collision is possible (Section 5.4).

In this section, we leverage the mappings between the privileges on system architecture and adversarial actions, i.e., instances of threats, to identify the minimal safety set (MSS) related

to hazards H_1, H_3, and H_6. We then use MulVAL to generate a system-wide logical attack graph that encompasses paths leading to privileges in the MSS. We then analyse attack paths to identify security measures.

System architecture We use the architecture for the CBTC system shown in Figure 6.8, which features *three* layers: enterprise, ground control and mobile. The enterprise layer consists of *three* interconnected sub-networks, including both workstations and servers. The ground network ensures the infrastructure control operations, including a redundant implementation of the zone controller and a GSM receiver/transmitter module. Each train has its own local network that hosts a redundant implementation of the train controller, a GSM module, the GNSS receiver and a radio module for positioning and the HMI. We omit components not connected to the TPC/IP layer, such as speed controller, odometer and actuation units, as they communicate with the train controller over a dedicated bus (i.e. CAN), which we assume is out of reach for the adversary. Within the architecture, we have instantiated 54 known vulner-

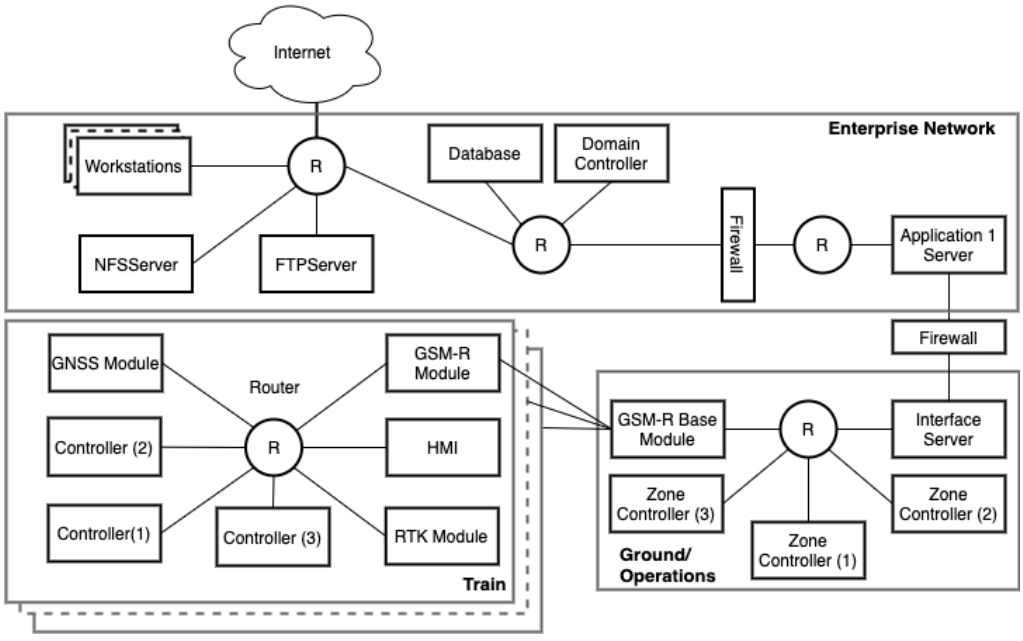


Figure 6.8: Deployment architecture of the CBTC system

abilities (`Vuln-01`, `Vuln-02`, ..., `Vuln-54`). In particular, 36 vulnerabilities on components of the mobile and ground layers lead to the privileges $\{p_1, ..., p_{14}\}$ (Section 7.3, Table 7.5).

Minimal Safety Set We now use Z3 to find a minimal safety set $\mathcal{S}$ for which the Boolean function $\mathcal{H}$ (Equation 6.8) is false (i.e. a collision cannot be caused).

$$\mathcal{H} = (H_1 \vee H_3 \vee H_6) \tag{6.8}$$

As the set of privileges is contained in size, the solver can completely enumerate all the 9 minimal safety sets $\mathcal{S}_1, \ldots, \mathcal{S}_9$ in $< 1s$ We list below the minimal safety sets for the CBTC use case. Note that privileges in the set $\{p_2, p_7, p_8, p_9, p_{14}\}$ are common across all the $\mathcal{S}_i$ found in this use case. From the same sets we also notice that the defender must protect unauthorised access to any of the privileges in the sets $\{p_3, p_5, p_6\}$ and $\{p_{10}, p_{11}, p_{12}\}$ to prevent a *collision*.

Set	Privileges in set
$\mathcal{S}_1$	$p_2, p_3, p_7, p_8, p_9, p_{11}, p_{13}, p_{14}$
$\mathcal{S}_2$	$p_2, p_6, p_7, p_8, p_9, p_{10}, p_{13}, p_{14}$
$\mathcal{S}_3$	$p_2, p_5, p_7, p_8, p_9, p_{10}, p_{13}, p_{14}$
$\mathcal{S}_4$	$p_2, p_6, p_7, p_8, p_9, p_{11}, p_{13}, p_{14}$
$\mathcal{S}_5$	$p_2, p_5, p_7, p_8, p_9, p_{11}, p_{13}, p_{14}$
$\mathcal{S}_6$	$p_2, p_3, p_7, p_8, p_9, p_{12}, p_{13}, p_{14}$
$\mathcal{S}_7$	$p_2, p_6, p_7, p_8, p_9, p_{12}, p_{13}, p_{14}$
$\mathcal{S}_8$	$p_2, p_3, p_7, p_8, p_9, p_{10}, p_{13}, p_{14}$
$\mathcal{S}_9$	$p_2, p_5, p_7, p_8, p_9, p_{12}, p_{13}, p_{14}$

Table 6.1: Minimal safety sets CBTC system

Graph path analysis We employ MulVAL to build the attack graph for the system, assuming the workstation is the entry point. In addition to this, we also assume that they can execute GNSS spoofing attack. To carry out an attack that leads to a safety violation (hazard), the attacker needs to obtain relevant privileges associated with any of the threat scenarios $(p_1, \ldots, p_{14})$. To this extent, we use these privileges as targets for the attack graph generation. The graph generated with MulVAL includes a considerable number of nodes and attack paths and is discussed in greater detail in Section 7.3.

In Figure 6.9, we summarise the progression of the attacker within the system architecture for improved readability. The exploitation of vulnerabilities `Vuln-03` and `Vuln-04` provides the attacker with a *foothold* in the enterprise network. Thus, an attack path through the *Interface Server (Operation)* allows them to pivot towards the operation zone, where they gain network

visibility of the zone controllers. Further exploitations on the operation network allow them to reach privileges $p_2, p_4, p_8, p_{10}, p_{11}$, and p_{12}. In addition, the exploitation of `Vuln-26` creates a path towards the on-board network of a train, where the adversary can access the remaining safety critical privileges ($p_1, p_3, p_5, p_6, p_7, p_9, p_{13}$, and p_{14}).

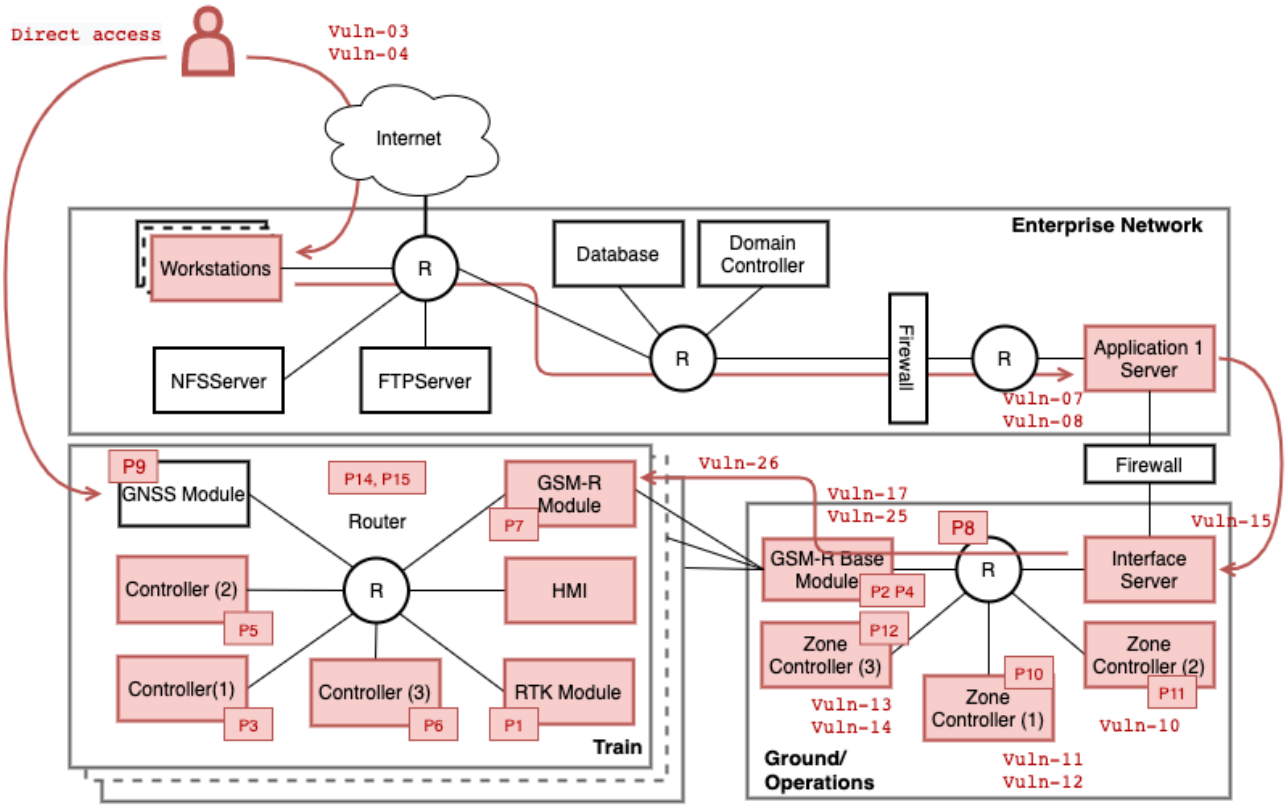


Figure 6.9: Attack graph (CBTC)

The automatic identification, on the attack graph, of the vulnerabilities that should be remediated requires an enumeration of the attack paths, which is challenging given the depth of the graph. To solve the problem, we select one of the minimal safety sets that satisfies $\mathcal{H} = false$, and use JGrapht to evaluate, individually, the paths leading to the privileges in the set.

For each path, we analysed the number of occurrences of AG nodes across all the paths and we found the following occurrences to be common to all (or most of) the paths

```
netAccess(opgInterfaceServer,_,_)
execCode(opgInterfaceServer,root)
execCode(application_server_1,root)
```

Listing 6.1: Common nodes

These privileges were not part of the initial set $\{p_1, \ldots, p_{14}\}$ but are necessary to reach them.

We select p^*=*execCode(opgInterfaceServer, root)* and apply our backward enumeration algorithm to find all the attack paths leading to p^*. This is feasible now since p^* is closer to the perimeter than the privileges in the initial set $\{p_1, \ldots, p_{14}\}$. We identify the following vulnerabilities that are shared across the attack paths leading to p^*.

```
(opgInterfaceServer,Vuln-15,kernel,localExploit,privEscalation)
(applicationServer1,Vuln-08,vulnerable_web_app,remoteExploit,
 privEscalation)
(workstationHost10,Vuln-03,rdp_server,remoteExploit,privEscalation)
```

Listing 6.2: Common vulnerabilities

p^* is common to all but one *graph paths* found in the original attack graph, from the perimeter to $\{p_1, \ldots, p_{14}\}$. We fix the five vulnerabilities above and generate a new attack graph for the entire system. Since p^* was a common node, by removing it, we remove all but one path leading to the minimal safety set. In fact, the new AG contains only one attack path, which is the one leading to p_9 which, together with any privilege in the set $\{p_1, p_4, p_{15}\}$ can lead to a *collision*. This corresponds to a GNSS spoofing attack against the system, which is not easy to remediate.

6.4 Likelihood of threat scenarios

We employ Bayesian Attack Graphs (BAG) [LM05] to compute the marginal probabilities of an adversary being successful in a threat scenario. Namely, we leverage exact inference techniques for inference on Bayesian networks presented in [MGSBL17, MGSPL17].

In previous sections, we have elaborated on the generation of attack paths leading to safety-critical privileges using MulVAL (Section 6.2.1). Unfortunately, the produced attack graph carries two limitations which do not allow the direct application of Bayesian inference techniques to estimate the marginal probabilities associated with threat scenarios. A few manipulations are thus in order to achieve our goal. The first limitation concerns the semantics of the objectives

node in the attack graph. In particular, the attack graph employed in earlier sections (Section 6.3.1) was produced providing objectives to MulVAL in the form of privileges on the system architecture that we had previously associated with threats in the attack library (e.g., *execute code* as *root* on a certain *host*, *network visibility* of some machines, etc.). In other words, we were querying MulVAL to display the paths leading from an assumed entry point towards the input privilege. While formally correct, this approach leads to an attack graph which only models architectural elements and has no means of representing adversarial actions or threat scenarios. In order to compute the marginal probabilities associated with threat scenarios in a BAG, it is crucial, to extend our attack graph to also represent threat scenarios. The second limitation in the AG produced by MulVAL is that they admit strongly connected components. In fact, a BAG is a Bayesian network and, as such, does not traditionally admit cycles. In order to apply Bayesian inference and compute the probability of a successful attack, we also need to transform the attack graph into a directed acyclic graph. We select a DAG that is a subgraph of the original AG. In particular, in the DAG, we enforce the assumption of monotonicity of the attack as discussed earlier in this chapter (Section 6.2.3).

In this section, we explain how we overcome these two limitations and, starting from the results produced by MulVAL, we produce a Bayesian attack graph with the objective of estimating the likelihood of exploitation of threat scenarios.

6.4.1 Extending AG representation

We overcome the first limitation by extending MulVAL attack graphs to also include nodes to represent adversarial actions and threat scenarios. This is done in two stages. First, we rely on the mapping between the architectural model and STAMP as a source to expand the MulVAL default knowledge base (KB). In particular, we introduce a set of primitive facts to model whether a host in the system architecture changes and/or transports STAMP control actions, feedback, and process model variables. We also define a new set of predicates which evaluate *true* when the adversary acquires enough privileges to exploit an instance of a threat against an element of the functional model. Second, we perform a post-processing of the attack

graph using the threat scenarios tree as an input. The operation aims to join the adversarial actions introduced in the previous pass in threat scenarios. By applying these two steps, we are able to bring the description of threat scenarios into the attack graph, thus overcoming the first limitation. We discuss implementation details later in the following paragraphs.

Expanding MulVAL knowledge base We have extended MulVAL with three categories of rules to model the impact of the adversary on the CPS information flow and bind adversarial actions to the privileges required to actuate them. The first category of rules describes *tampering* with the information flow or with a process model variable x. For example:

$$canTamper(x) \leftarrow execCode(C, root), controlsPMV(C, x) \tag{6.9}$$

which states that an adversary can tamper with the STAMP element x, if they have obtained high-privilege code execution rights on controller C, and the controller C controls x. Similarly, the adversary can tamper with the element of the STAMP model x if they have obtained high-privilege code execution rights on host C, and C transports (e.g., router, relay, etc.) x:

$$canTamper(x) \leftarrow execCode(C, root), transportsFlow(C, x) \tag{6.10}$$

The second category of derivation rules we have added describes the privileges an attacker needs to *spoof* x. For example, an adversary can spoof the element of the STAMP model x if they have obtained access to the OT network where controller C is located, and the controller C controls the element of the STAMP model x:

$$canSpoof(x) \leftarrow netAccess(C, Prot, P), transportsFlow(C, x) \tag{6.11}$$

Finally, a further category of derivation rules describes the privileges that the attacker needs to perform an availability attack on the information flow, e.g., *drop* x. For example, an adversary can drop x if they have obtained access to the OT network where x is transmitted by actuating

a denial of service attack:

$$canDrop(x) \leftarrow netAccess(C, Prot, P), transportsFlow(C, x) \tag{6.12}$$

It must be noted that the last category is currently not complete. In fact, the adversary could *drop* x also through the exploitation of a denial of service vulnerability on the controller, which produces or transports x. To this extent, an availability attack should also include rules 6.13 and 6.14

$$canDrop(x) \leftarrow unavailable(C, root), controlsPMV(C, x) \tag{6.13}$$

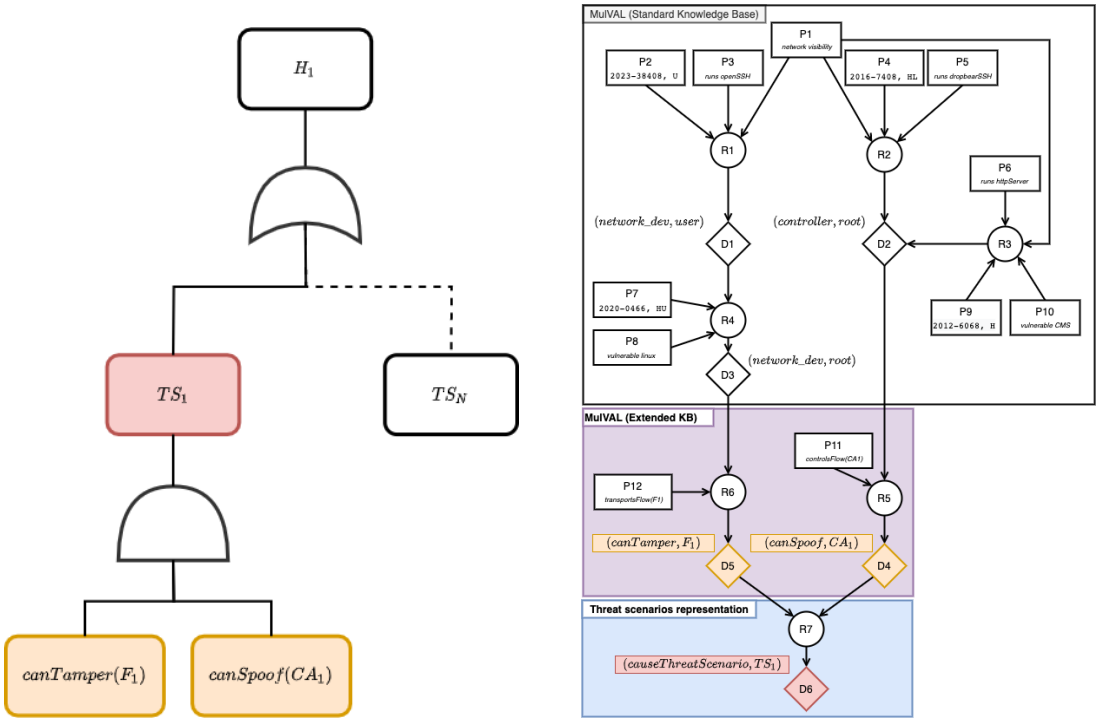
$$canDrop(x) \leftarrow unavailable(C), transportsFlow(C, x) \tag{6.14}$$

where *unavailable* is true if the attacker can exploit an *availability* vulnerability on component C, with $controlsPMV(C, x)$ and $transportsFlow(C, x)$ being the same as defined in 6.9 and 6.10. However, at the moment the default knowledge base provided with MulVAL [OGA+05, OBM06] does not define predicates aimed to model the availability of a network host nor consider exploits targeting their availability.

In terms of graph semantics, we have defined the primitive facts $controlsPMV(C, x)$ and $transportsFlow(C, x)$ which bind elements of the STAMP realm to the system architecture. On the other hand, predicates $canTamper(x)$, $canSpoof(x)$, and $canDrop(x)$ create *or* nodes (derived facts), which highlight the acquisition of privileges on the information flow. At the same time, the set of rules has been extended with rules in equations 6.9, 6.10, 6.11, 6.12, 6.13, and 6.14. These create nodes of type *derivation rule* which combine the exploitation of architectural elements with facts about system configuration, including the control and transportation of functional elements, to infer whether exploitation allows access to relevant functional elements.

Modelling threat scenarios Having represented adversarial actions as part of the AG, adding threat scenarios is now a straightforward operation. We have developed a software routine - as a part of the AG-Tools library - which imports threat scenarios as *and/or* combinations

of adversarial actions from the threat scenarios tree (Section 5.3.2) into the attack graph. In



(a) Threat scenarios tree **(b)** Attack graph

Figure 6.10: Adversarial actions (orange) and threat scenarios (red) on threat scenarios tree (6.10a) and attack graph (6.10b).

terms of graph semantics, an *AND* in the threat scenarios tree introduces a derivation rule node in the attack graph, which is true iff the adversary gains enough privileges to perform all adversarial actions needed to exploit it. On the other hand, a threat scenario is a *derived fact* which comes as a consequence of the derivation rule.

On the right-hand side of Figure 6.10, we provide an example of an extended attack graph which models the paths to a threat scenario given the simple threat tree in Figure 6.10a and the following system configuration

$$transportsFlow(network\_dev, F_1)$$

$$controlsFlow(controller, CA_1)$$

Starting from the threat scenario tree - which we produced during step ❶ of Cassandra- we query MulVAL to generate given the objectives $canTamper(F_1)$, $canSpoof(CA_1)$. MulVAL

relies on the extended KB to infer nodes D_4 and D_5. At the same time, the standard KB is used to infer D_1, D_2 and D_3. Having been generated using two adversarial actions as objectives, the produced attack graph terminates in D_4 and D_5. A call to our AG-Tools library allows adding nodes R_7 and D_6. In this simple case, we find *two* attack paths leading from the system perimeter - we assume that the adversary has access to the control network, which also gives them visibility of the controller host - to D_6. One path requires the exploitations R_1 (`CVE 2023-38408`), R_2 (`CVE 2016-7408`), and R_4 (`CVE 2020-0466`). The second path follows a different route via R_3 (`CVE 2012-6068`); it requires R_1, R_3, and R_4. Both paths reach D_4 and D_5 via R_5, R_6, and R_7.

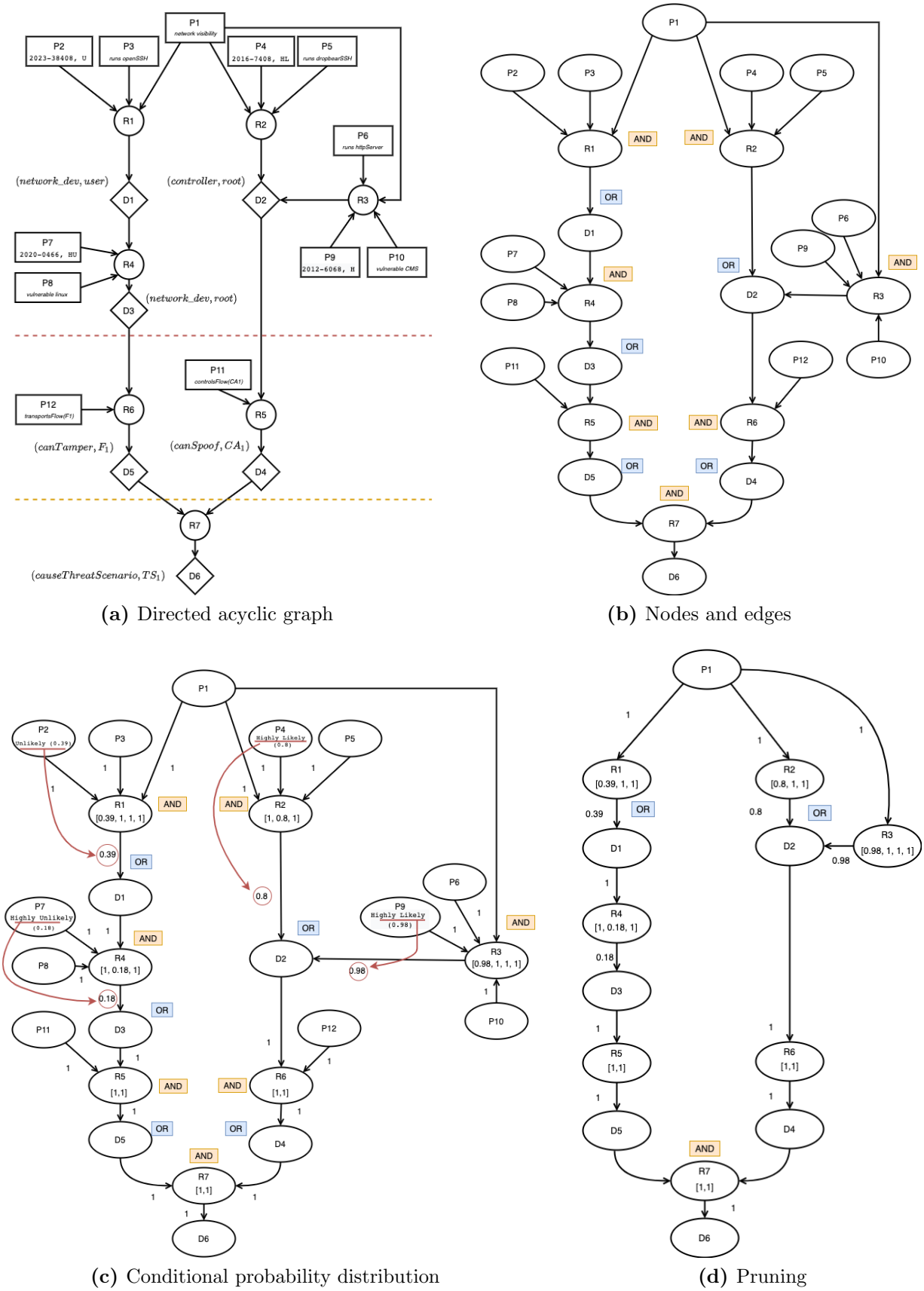
6.4.2 BAG Extraction

We have elaborated on an extended representation for MulVAL attack graphs which introduces the capability of using primitive facts, derivation rules, and derived facts to represent higher-level aspects such as adversarial actions and threat scenarios. Such extended representation allows for extracting attack paths which relate the attack surface to specific threat scenarios. Instead, a traditional representation (Section 6.2.1) would only be able to represent the privileges needed to execute adversarial actions (i.e., without including adversarial actions and threat scenarios).

In order to compute the risk associated with threat scenarios, we need to transform the extended MulVAL attack graph into a Bayesian attack graph (BAG). We first extract a DAG, subgraph of the AG as outlined earlier (Section 6.2.3). Thus, we construct the BAG in three passages: we populate the graph with nodes and edges, we define the conditional probability distribution (CPT), and, finally, we prune the BAG. These steps have been implemented as part of our AGTools library.

In Figure 6.11, we show the transformation DAG to BAG applied to the attack exemplary attack graph originally outlined in Figure 6.10 above.

A BAG is a graph $\mathcal{G}_{BAG} = (V_{BAG}, E_{BAG})$ where each node represents a Bernoulli variable X



(a) Directed acyclic graph

(b) Nodes and edges

(c) Conditional probability distribution

(d) Pruning

Figure 6.11: Transformation from directed acyclic graph to Bayesian attack graph.

which takes value 1 with probability p (and 0 with probability $1-p$) [MGSPL17, MGSBL17]. Let X_i be the variable represented by node $w_i \in V_{BAG}$, we interpret the probability of X_i being true as the probability of an attacker reaching (compromising) w_i [MGSPL17, MGSBL17].

Nodes and edges Let $\mathcal{G}_{DAG} = (V_{DAG}, E_{DAG})$ be a DAG (subgraph of a MulVAL AG) and let $\mathcal{G}_{BAG} = (V_{BAG}, E_{BAG})$ be the associated BAG, we maintain a one-to-one correspondence between vertex and edge sets of the BAG, and those of the DAG. To this extent, for each node $v_i \in V_{DAG} = \{V_p \cup V_d \cup V_r\}$, we instantiate a new variable and a BAG node $w_i \in V_{BAG}$. Similarly, for each edge $(v_i, v_j) \in E_{DAG}$, we instantiate a BAG edge $(w_i, w_j) \in E_{BAG}, \quad i \neq j$. This passage is shown in Figures 6.11a and 6.11b.

Conditional probability distribution The conditional probability distribution of each node $p(X_i|pa_i)$ lays out the probability of X_i being true, i.e., adversary reaching node $v_i \in V_{DAG}$. The state of X_i (*compromised* or *not compromised*) depends on the state of parent nodes pa_i and on $p(v_i)$. The latter is the likelihood of an adversary exploiting $v_i \in V_{DAG}$ [MGSPL17].

To calculate the CPD at each node, we consider the two cases of logical *'and'* and logical *'or'* and employ the formulas for conditional probability tables outlined in [MGSBL17]. In particular, if $v_i \in V_d \subset V_{DAG}$ is a *derived fact*, we compute $p(X_i|pa_i)$ using a logical *'or' (noisy)* formulation. On the other hand, if $v_i \in V_r \subset V_{DAG}$ is a *derivation rule*, we compute $p(X_i|pa_i)$ using the logical *'and'* formulation [MGSBL17].

For simplicity, we assume that all *primitive facts* in the DAG are *certain*, i.e., we are not dealing with stochastic information such as the likelihood of an application being vulnerable to an exploit, or the likelihood of a host running a vulnerable program or being poorly configured. A primitive fact has no parents in the BAG and it is treated as an *evidence* during the inference process, e.g., it takes value 1 (true) with probability $p = 1$ (always).

Our MulVAL representation admits a number of different types of derivation rules that the attacker can exploit to obtain a privilege, given a set of preconditions. These include but are not limited to the exploitation of a vulnerability, the exploitation of a weakness which is not

a vulnerability such as misconfiguration, password re-use, etc, as well as network access and network visibility. We estimate $p(v_i)$, the likelihood of the adversary exploiting rule $v_i \in V_r \subset V_{DAG}$ as follows.

- If $v_i \in V_r$ is the exploitation of a vulnerability, we use the Common Vulnerability Scoring System (CVSS) [NIS] to estimate $p(v_i)$. In particular, we rely on the exploitability metric as proposed in [MGSBL17]. Let $v_j \in V_p$ be a primitive fact such that v_j is a vulnerability and there exists a directed edge $(v_j, v_i) \in E_{DAG}$, then we set $p(v_i)$ according to the exploitability score assigned to v_j by CVSS.

- If $v_i \in V_r$ is the exploitation of a different type of weakness, we assume that its likelihood depends on the difficulty of the exploitation itself. The default knowledge base provided with MulVAL contains a set of metrics that estimate the difficulty of common weaknesses such as password re-use under different assumptions, e.g., *competent* or *incompetent* user [OGA+05].

- If $v_i \in V_r$ is the direct consequence of the acquisition of a privilege and does not require any explicit action from the adversary, e.g., network visibility of a host gained following the acquisition of some privilege on a machine that is connected to the same network of the host, then we consider $p(v_i) = 1$. In other words, if the adversary has gained access to a host h_1, then they also have network visibility of h_2 with no additional effort $\iff$ all the other preconditions, e.g., h_1 and h_2 are on the same network, eventual firewall rules are satisfied, etc.

Finally, once that a derivation rule is exploited the adversary also obtains the privilege associated with it. Thus, derived facts also have $p(v_i) = 1$, with $v_i \in V_d$.

Pruning Finally, in the third passage, we prune the Bayesian Attack Graph by removing all primitive facts (PFs) except for the entry point(s) of the attacker (Figure 6.11d). The latter constitute the root of the Bayesian network. In the case of multiple entry points, these are aggregated in one root as mentioned in [MGSBL17].

The main differences between our approach and others in the literature, such as [MMSvM20], concern, among other things, the construction of the CPT and the pruning operations. In particular, with the latter, we substantially reduce the number of nodes in the BAG without impacting the computation of marginal probabilities (PFs are all assumed to be *certain*).

6.4.3 Inference on the BAG

We use the software developed by authors of [MGSPL17, MGSBL17] to compute marginal probabilities in the BAG. This is a MATLAB script which implements the CPT formulas outlined in the previous paragraph and uses BNET Toolbox [Mur01, Mur14] APIs to perform inference allowing to choose among exact approximate approaches including variable elimination (VE), junction trees (JT), belief propagation (BP), and loopy belief propagation (LBP).

In particular, we extract the function *margProb.m* (Figure 6.12) which takes as input the attack graph (adjacency matrix), the individual (apriori) likelihood of exploitation of individual nodes, node types (*and/or*), and the desired inference approach. It returns, as its output, the marginal probability of each node being compromised along with the inference time. We leverage our

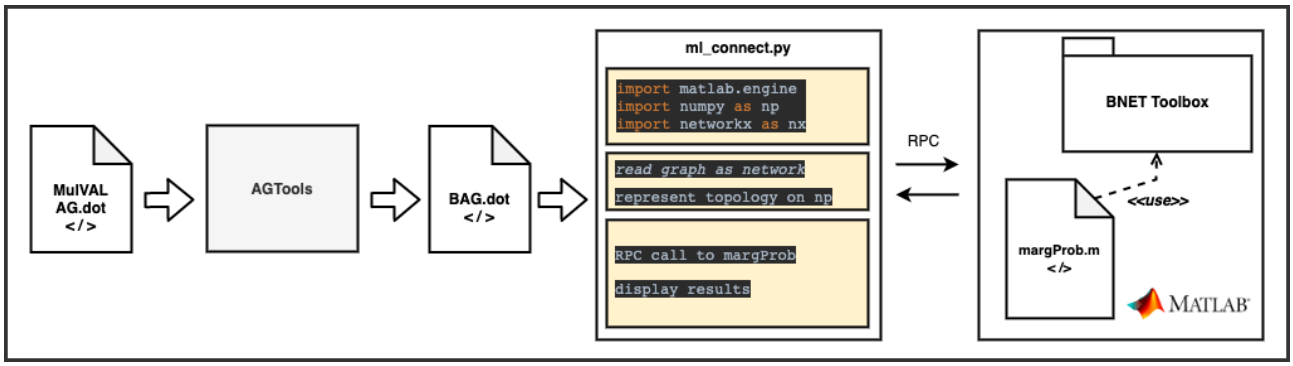


Figure 6.12: Overview of the proposed methodology.

own AGTools library to apply the passages described earlier and extract a BAG, given the AG produced by MulVAL and a set of potential objectives nodes. The latter are usually threat scenarios related to the system hazard we are investigating. We have developed a Python 3 program which reads the BAG from the file and the MATLAB function as a remote procedure call (RPC) via Matlab Engine [Mat] as shown in Figure 6.12.

While Mathworks also provide JAVA APIs for remote procedure calls of MATLAB functions, we have found Python APIs to be better documented and easier to integrate with the rest of our toolchain. In addition, in Python we were able to rely on numPy [HMvdW+20] for an efficient computation BAG adjacency matrix and vector of probabilities, arguments of the RPC. These two reasons underlie our decision to write a separate Python 3 program to call MATLAB remote engine which is not part of our JAVA AG-Tools library. At the same time, we found that a solution that relies on Python 3 and numPy would be more efficient than importing the BAG in MATLAB directly. In fact, we believe that operations such as the extraction of the adjacency matrix and other graph attributes from the Graphviz [EGK+02] file produced by AGTools can be performed more efficiently in Python than in MATLAB.

6.4.4 Example of application

In Chapter 9, we apply Cassandra to analyse the safe operation of a Flight Management System (FMS) when the adversary tries to access safety-critical information by compromising the device used as the Electronic Flight Bag (EFB). We use Bayesian inference to evaluate the probability of successful attacks under three different security policies around the management of the EFB. We discuss the efficacy of analysed controls in preventing potentially catastrophic threats in modern avionic architectures.

6.5 Summary & Conclusions

In this chapter, we have proposed the *Threat-Vulnerability-Accident* model, which bridges the gap between the functional and the architectural domains of the cyber-physical system. Through this model, we can represent instances of threats as functions of the privileges within the system architecture and, thus, threat scenarios as functions of the privileges that the adversary must acquire to cause them. Using this representation, we can look at low-level weaknesses and vulnerabilities in the system architecture from a novel perspective, i.e., being able to visualise the role they have in enabling safety-critical attacks. In this direction, we have elaborated

on the use of attack graphs to assess the viability of safety-critical threat scenarios and quantify their risk.

We have introduced an algorithm to enumerate the minimal set of privileges that should be protected to guarantee safety, which we defined as the Minimal Safety Set (MSS). The application of our proposed approach to the CBTC use case and the qualitative analysis of the produced attack paths yields two important results. First, given the current system architecture and security posture (e.g., vulnerabilities, network configurations, etc.), only a subset of threat scenarios identified in the previous step of Cassandra are concretely exploitable by an attacker. Second, the identification of the MSS, combined with the qualitative analysis of the attack paths leading to privileges in the MSS, aids the identification of mitigations to prioritise to prevent safety-critical threat scenarios.

Due to the size of the attack paths, a qualitative approach might not always be feasible. In this chapter, we have also developed a quantitative approach which employs Bayesian Attack Graphs (BAG) [LM05] to estimate the risk of safety-critical attacks. This required us to extend MulVAL with new rules, and to define a procedure to extract a BAG given an attack graph produced by MulVAL.

The content presented in this chapter provides Cassandra with the capabilities of analysing the architecture of the CPS. This enables qualitative reasoning for mitigation of the attack paths and probabilistic (Bayesian) reasoning for risk quantification and assurance. Together, they enable us to evaluate different system design alternatives and make informed choices.

The content presented in this chapter concludes the presentation of Cassandra. We will showcase its application to increasingly more realistic use cases in the next three chapters. In particular, in the upcoming chapter, we will conduct the integrated safety and security analysis of the Communication-Based Train Control (CBTC) system. This CPS has also been employed as run thorough illustrative example in the last three chapters of the thesis. To this extent, the next chapter combines and expands previous discussions related to the CBTC system into a single analysis. The analysis is oriented to the discovery of privileges that should be protected to preserve safety.

Chapter 7

Analysing the safe operations of a Communication-Based Train Control system: a case study

In this chapter, we employ Cassandra to analyse the safe operation of a representative Communication-Based Train Control (CBTC) system in adversarial conditions. In doing so, we recap in a single analysis the examples used as run-through examples in the previous chapters (Chapters 4, 5, and 6).

The overarching objective of the analysis proposed in this chapter is the identification of weaknesses and vulnerabilities, both in the Information Technology (IT) and Operation Technology (OT) layers, the mitigation of which should be prioritised in order to preserve system-level safety properties. We use Cassandra to achieve it in this use case in three steps. First, we discover high-level safety-critical threat scenarios; we then leverage the integration with system architecture to identify the minimal safety set (MSS) and use MulVAL to generate paths leading these privileges. Thus, the qualitative analysis of found paths tells which weaknesses and vulnerabilities should be resolved to cut (or reduce) the reachability of privileges in the MSS.

This chapter is organised as follows. In Section 7.1, we present an overview of the represen-

tative CBTC system under analysis. In particular, we recap the safety analysis of the system and discuss the behaviour of its components. We also identify the threats against its control structure and synthesise a behavioural model for the attacker. In Section 7.2, we proceed with the identification and enumeration of threat scenarios. Then, we compute the MSS and produce the attack paths leading to privileges in the MSS in Section 7.3. We discuss results in Section 7.4. Lastly, in Section 7.5, we present the conclusions of the chapter.

The analysis presented in this chapter was presented as a case study part of a wider article which was published in IEEE Transactions on Dependable and Secure Computing (TDSC) [CL23].

7.1 Preliminaries

Earlier in this thesis, we provided an overview of the CBTC scenario analysed (Section 2.2). In the use case, shown in Figure 2.1), we analyse a system of N trains that share access to the same infrastructure. A zone controller (ZC) is responsible for ensuring exclusive access to all the M zones. Trains are equipped with an onboard Train Controller (TC) which receives the current position from an onboard location unit (LU) and speed from the Speed Controller (SC). The TC sends *access* requests to the zone controller upon approaching a new zone. Based on the reply, TC decides whether it can enter the new zone. The TC also sends *clear* requests to the zone controller when the previous zone is left (cleared). In our scenario, the LU combines data from a GNSS receiver with radio beacons to improve the accuracy of the estimated position [Wan04], in a configuration similar to [CFP+20]. In addition, a back-off strategy allows the LU to rely solely on GNSS if the radio signal is unavailable. Train motion is controlled by the speed controller, which computes the acceleration profile and commands the brake and the acceleration unit (AU) over a local bus.

7.1.1 Safety Analysis

We have applied the first three steps of STPA/STPA-Sec to the use case (Sections 2.7). We have identified the following accidents and hazards. In particular, we considered two safety-critical accidents, namely, *collision* (A_1) and *derailment* (A_2) accidents. These lead to the heaviest losses and can endanger human lives [CDP+17]. For completeness, we have also considered *traffic disruption* (A_3). The latter is an example of a less important accident. In fact, although (A_3) is a non-life-threatening accident, an adversary can still aim to cause it, to induce financial loss. Thus, we identified *seven* hazards leading to accidents A_1, A_2 and A_3. These are reported in Table 7.1 (also Table 2.3).

Table 7.1: Table of Hazards for the CBTC system.

#	**Hazard**	**Accidents**
H_1	Train enters a *not cleared* zone	A_1, A_3
H_2	Train authorised but does not enter a *cleared* zone	A_3
H_3	State of zone controller is not consistent with the trains' position	A_1, A_2, A_3
H_4	Train travels below minimum speed	A_3
H_5	Clear zone wrongly marked as *not clear*	A_3
H_6	Not clear zone wrongly marked as *clear*	A_1
H_7	Train travels above maximum speed	A_1, A_2

We report the safe control structure (SCS) of the system in Figure 7.1 (also Figure 2.4). Control actions are shown in red, while feedback signals are in blue. These are exchanged over digital (dashed) and physical (dotted) channels. The system comprises *one* zone controller, and *three* controllers for each train: a *train controller*, a *location unit* and a *speed controller*. ZC is responsible for issuing control actions CA_1 (authorisation) and CA_3 (speed limit) in a timely manner over a GSM-R channel. The process model of the ZC stores the state of the controlled zones, along with the associated speed limits. The status of a zone is a binary value indicating whether it is *free* or *engaged*, while the speed limit is an integer value. The control action CA_1 carries the value *allow* or *deny*, while CA_3 is a positive integer. Both control actions are

computed as a function of the internal status of the controller (*zones* and *speed limits* arrays, respectively) and of the position of the train. The process model of TC stores the current position (zone) and speed as well as a binary value, which indicates whether the train has been cleared to proceed or not towards the next zone. TC issues CA_2 and CA_4 to *request access* or to *clear* a zone respectively. Control actions CA_2 and CA_4 carry the ID of the issuing train ($tid \in N$) along with the id of the zone ($i \in M$) and also depend on the internal state of the controller for the current position. TC also sets the target speed CA_5 on the speed controller and receives the current speed from the speed controller (F_1). The Location Unit is responsible for communicating the current location CA_6 to TC. The value of CA_6 depends on

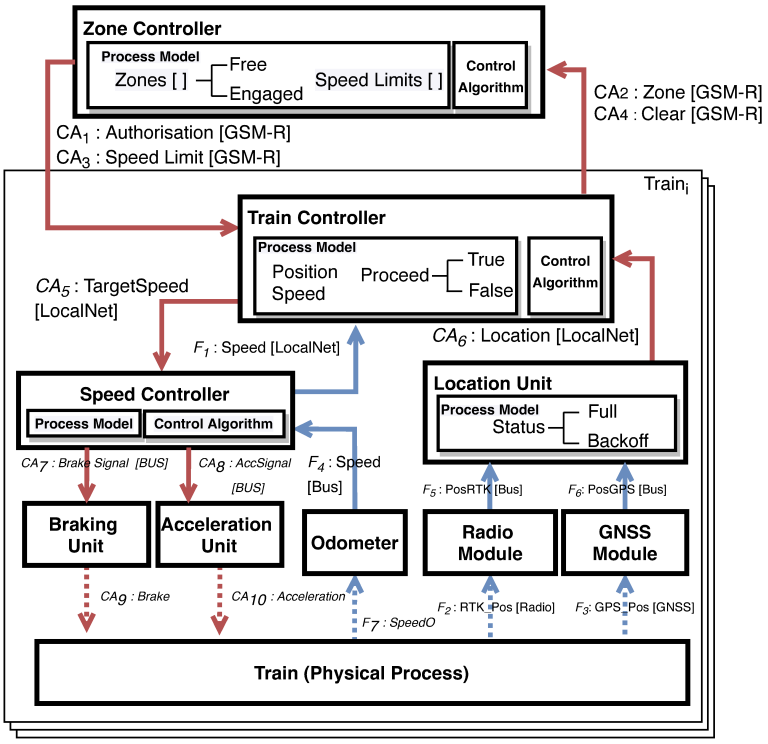


Figure 7.1: Safe Control Structure of Communication Based Train Control system.

the current operating mode (*full, fallback*) of the LU, which is stored in LU's process model as a binary variable. The LU receives its current position from the Radio (F_5) and GNSS (F_6) sensors. The Speed Controller receives the target speed from the TC and, given the current speed (F_4) computes an acceleration profile. The latter is used to command the brake (CA_7) and acceleration (CA_8) units. The odometer, radio sensor and GNSS module receive feedback signals (F_7), (F_2) and (F_3) respectively.

We applied the third step of STPA to the CBTC system (Figure 7.1) and derived the context

table in Table 7.2(also Table 2.6). For brevity, we have only analysed the unsafe application of CA_1, CA_2, CA_3, CA_4 and CA_5. We have mentioned that these actions are all in the discrete domain; therefore, UCAs of the type *applied for too long* or *applied for too short* are not included in Table 7.2.

Table 7.2: Context Table of Zone Controller (ZC) and Train Controller (TC). r is the value for which CA is unsafe, v is the speed limit.

Controller	**CA**	$Zone[i]$	Limit	Position	Previous	Proceed	**Not Provided**	**Provided**	**PE**	**PL**
ZC	CA_1	Engaged	-	-	-	-	H_4	H_1 $[r = allow]$	-	H_4
ZC	CA_1	Free	-	-	-	-	H_4	H_2 $[r = deny]$	-	H_4
ZC	CA_2	-	-	$i \in M$	-	-	H_4	H_3, H_5, H_1 $[r \neq i]$	-	H_4
ZC	CA_3	-	v	-	-	-	H_4	H_7 $[r > v]$	H_7	H_4
ZC	CA_4	-	-	-	$i \in M$	-	H_3, H_5	H_3, H_5, H_6 $[r \neq i]$	-	H_6
TC	CA_5	-	-	-	-	False	-	H_1 $[r > 0]$	-	-
TC	CA_5	-	-	-	-	True	H_2	-	-	-

The following UCAs facilitate the zone management and the i-th train. We also indicate the hazards they lead to. A list of UCAs derived from Table 7.2 for CA_1, CA_2, CA_4 and CA_5 and is shown below. These expand the UCAs already presented in Table 2.5.

UCA_1: CA_1 *not provided* when $zone[i]$ is *engaged* $\longrightarrow (H_4)$

UCA_2: CA_1 *provided* with *allow* when $zone[i]$ is *engaged* $\longrightarrow (H1)$

UCA_3: CA_1 *provded late* when $zone[i]$ is *engaged* $\longrightarrow (H_4)$

UCA_4: CA_1 *not provided* when $zone[i]$ is *free* $\longrightarrow (H_4)$.

UCA_5: CA_1 *provided* with *deny* when $zone[i]$ is *free* $\longrightarrow (H_2)$.

UCA_6: CA_1 *provided* too late when $zone[i]$ is *free* $\longrightarrow (H_4)$.

UCA_7: CA_2 *not provided* $\longrightarrow (H_4)$.

UCA_8: CA_2 *provided* with wrong parameter $\longrightarrow (H_3, H_5, H_1)$.

UCA_9: CA_2 *provided* too late $\longrightarrow (H_4)$.

UCA_{10}: CA_3 *not provided* $\longrightarrow (H_4)$.

UCA_{11}: CA_3 *provided* with parameter r, with $r > v$ and v is the allowed speed limit $\longrightarrow (H_7)$.

UCA_{12}: CA_3 *provided* early $(r > v) \longrightarrow (H_7)$.

UCA_{13}: CA_4 *provided late* $\longrightarrow (H_4)$.

UCA_{14}: CA_4 *not provided* $\longrightarrow (H_3, H_5)$.

UCA_{15}: CA_4 *provided* with wrong position $\longrightarrow (H3, H_5, H_6)$.

UCA_{16}: CA_4 *provided* too late $\longrightarrow (H_6)$.

UCA_{17}: CA_5 *provided* when controlled replied *deny* (H_1)

UCA_{18}: CA_5 *not provided* when *Proceed* is true (H_2)

Finally, the fourth step of STPA-Sec is the identification of *threat scenarios*, which we automatically identify and enumerate in the first step of Cassandra (Section 7.2).

7.1.2 Behavioural Model

We have designed, implemented and tested the safe behavioural model of the CBTC subsystem responsible for the zone management. This includes aspects related to the exchange of the train position between the zone controller and multiple trains as well as coordination, permissions and clearances. For brevity, we only focus on zone access control and omit the aspects related to the control of the trains' speeds.

In particular, we have used UPPAAL [BDL+06b] to implement the safe behavioural model of the system and verify safety-critical safety properties. Due to implementation constraints imposed by the tool, such as the order of execution of instruction on edges, only one update being admitted for each edge, etc., the implementation of the automata might contain a few artefacts, thus differ from the formal description provided in Chapter 4. In the Figures and in the Sections below, we also refer to the zone controller (ZC) using the acronym RBC, which stands for Remote Base Controller. From now on, the acronyms RBC and ZC refer to the same component and are used interchangeably. The discussed behavioural model provides an implementation of the CBTC system, in the case of $N = 2$ trains.

As mentioned in Chapter 4, the behavioural model is a network of timed automata (NTA) and consists of $N_p + N_c + N_M + N_k$ timed automata synchronised over $N_{Ch} = 2 * N_f$ synchronisation channels, with $|B| = N_f$ buffer variables.

Controllers: Our implementation features $N_c = 2N + 1$: two controllers for each train, the train controller (TC) and location unit (LU), and the zone controller (ZC).

Physical Process: Our implementation features $N_p = 1$ number of physical processes. We use one TA to summarise the movements of all the trains across the zones.

Observers: Also, for each train, $N_M = 2$ observers monitor unsafe control actions of the type *Not Provided* and *Provided too Late* on CA_1, CA_2 and CA_4. To reduce verification times, we use the same automaton to observe CA_2 and CA_4.

Logic Channels: Our model uses one automaton ($N_k = 1$) to model the behaviour of the GSM-R channel for communications between ZC and trains. For simplicity, we do not include in the NTA the automata that defines the behaviour of the train's local networks and buses. Instead, we assume that the communication on these channels is reliable and instantaneous. As a result of this simplification, our behavioural model does not consider the impact of threats against feedback and control actions exchanged inside each train.

Synchronisation: *Three* pairs of synchronisation channels and *three* buffer variables for each train, allow to orchestrate the exchanges of control actions CA_1, CA_2 and CA_4 ($N_f = 3$). Two additional synchronisation channels are used, together with dedicated buffer variables to enable communication between location units and train controllers (CA_6) and to trigger the evolution of the physical process when the train controllers *decide* to enter a zone.

In Listing 7.1 we show the declaration, in UPPAAL, of the three pairs of channels (*s_approach, r_approach*, *s_reply, r_reply*, and *s_clear, r_clear*) and the buffer variables that they use (*area_code* and *reply_buff*).

```
/* Constants */
```

```
const int N = 2; // Number of trains
const int M = 3; // Number of zones

/* Types */
typedef int[0, M-1]  zone_t;
typedef int[0,N-1]   id_t;
typedef int[0,1]     reply_t;

/* Channels for Control Actions */
broadcast chan s_approach[N];
broadcast chan r_approach[N];

broadcast chan s_reply[N];
broadcast chan r_reply[N];

broadcast chan s_clear[N];
broadcast chan r_clear[N];

/* Shared Buffer Variables */
zone_t  area_code;
reply_t reply_buff;
```

Listing 7.1: Channels

Overall, to reduce the verification times, we make the following optimisations.

- We employ the same observer automaton to monitor control actions CA_2 and CA_4.
- We do not use an automaton to model the local bus of each train. This is based on the assumption that the adversary only targets communications between trains and zone controllers. In fact, we will observe that this optimisation does not enable us to model attacks on control actions and feedback exchanged on the internal bus.
- We employ one automaton to model the physical process instead of using one automaton for each train.

Through these optimisations, we were able to reduce the overall number of timed automata in the NTA to 11.

In the next paragraphs, we discuss the behavioural model of individual automata. The following paragraphs expand the content already presented in Chapter 4 by describing all the automata implemented.

Components In Figure 7.2 we show the template of the timed automaton implementing the behaviour of the zone controller (ZC). The automaton starts in a *Ready*. When a *zone* message

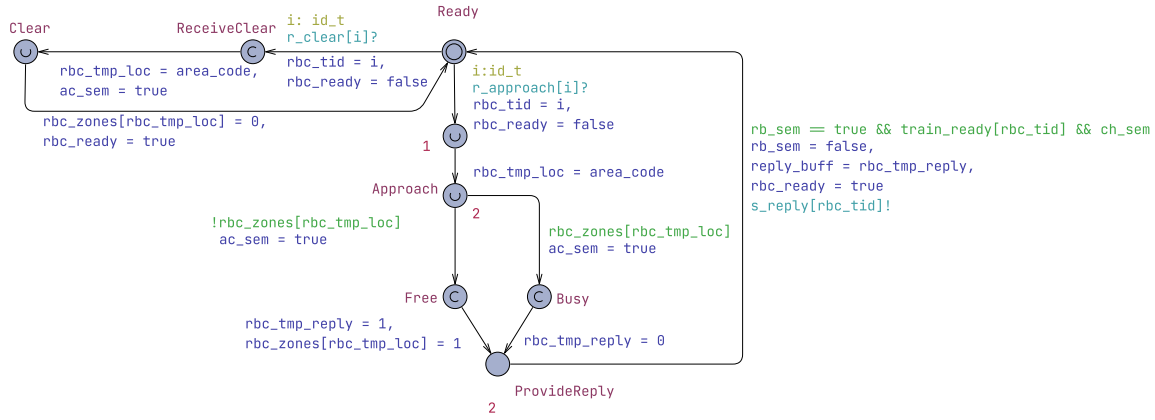


Figure 7.2: Zone Controller

is received (CA_2), ZC saves the *id* of the sender, copies the value from the shared buffer to a local variable and moves to an *Approach* state. ZC evaluates if the requested zone is busy or free and replies to the sender by issuing a *authorisation* message (CA_1). Then, it returns to the *Ready* state, waiting for the next message. From the *Ready* state, when *clear* message (CA_4) is received from train i, the zone controller copies the data from the shared buffer to a local variable and marks as *clear* the respective zone in memory. Then, it returns to the *Ready* state, waiting for the next message. A set of semaphores is used to enforce synchronisation between the zone controller and trains.

In Figure 7.3 we show the template of the timed automaton implementing the behaviour of the train controllers. In our use case, we instantiate 2 automata using this template, one for each train, respectively, with ids 0 and 1. It is possible to instantiate it multiple times to model a higher number of trains.

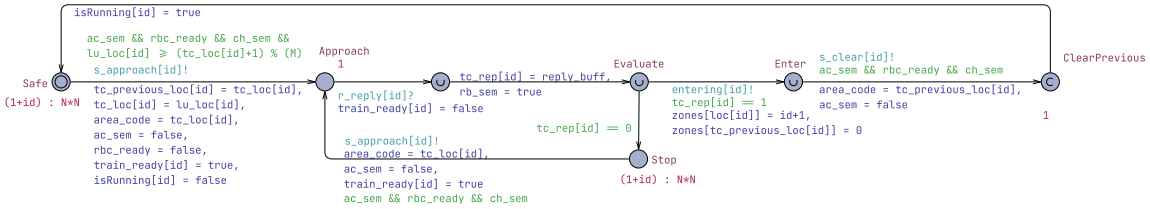


Figure 7.3: Train Controller

Each train starts in a *Safe* state. We define a train to be in a *Safe* state when it is travelling within its assigned movement authority (MA). When the train senses, through a change on the variable *lu_loc[train_id]*, that it is approaching the end of the current MA, the timed automaton sends a *zone* message (CA_2) to the zone controller and moves to the *Approach* state while it waits for the reply. When an *authorisation* message (CA_1) is received, the train evaluates its content and, if positive, proceeds towards the next movement authority. If ZC denies access, the train enters a *Stop* state and sends a new authorisation request. When the train enters the next zone, it sends a *clear* message (CA_4) to the zone controller to signal that it has left the previously occupied zone.

In Figure 7.4 we show the template of the timed automaton implementing the behaviour of the location units. In our use case, we instantiate 2 automata using this template, one for each train, respectively, with ids 0 and 1. The location unit simulates the acquisition of the data from track-side equipment (Radio RTK) and GNSS. The process starts in a *Healthy* state

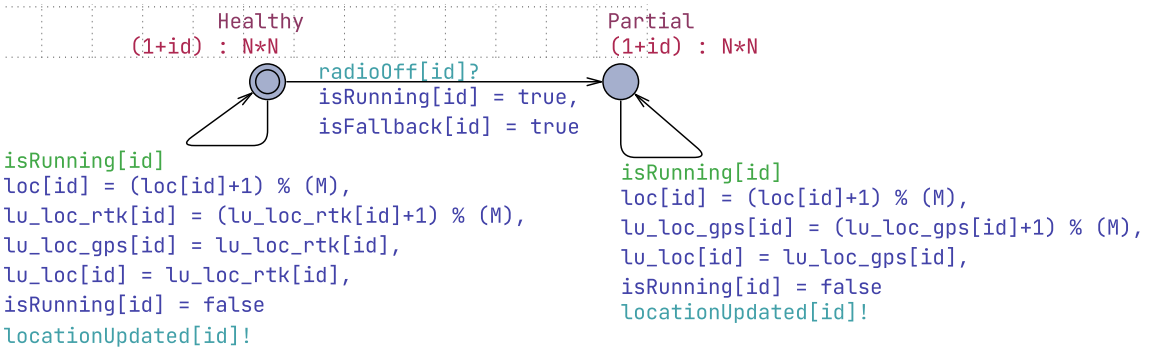


Figure 7.4: Location Unit

where the position is non-deterministically updated using the value from the local variable *lu_loc_rtk*. The value is sent to the train controller through the buffer *loc[id]*. To simplify the model, the location unit does not signal the variable update to the train explicitly through a synchronisation channel. The adversary can force the location unit to switch into a *Partial*

(degraded) mode where the position is updated using solely the value from the local variable *lu_loc_gps*.

In Figure 7.5 we show the template of the timed automaton implementing the behaviour of the physical process. This automaton provides a discrete simulation of the evolution of the state of the mutual exclusive access zones. The state machine stores an array that represents

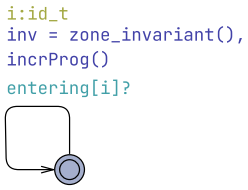


Figure 7.5: Physical Process

the current state (free or engaged) of the M zones. Every time that a train enters a zone, the respective zone is marked as *engaged*. Similarly, everytime that a train leaves a zone, it is marked as *free*.

Channels In Figure 7.6 we show the template of the timed automaton that models the transmission of a message on the zc-trains channel.

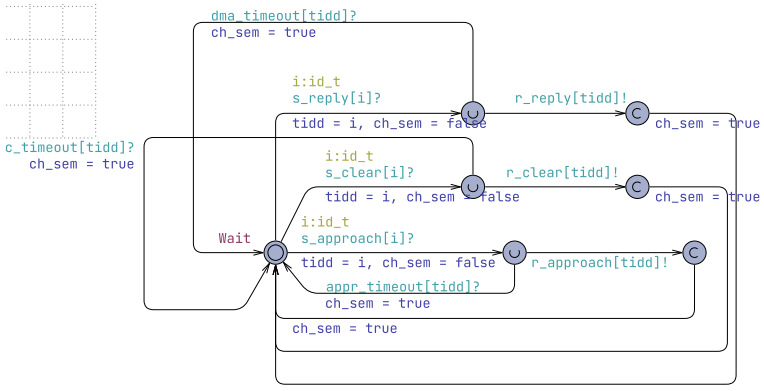


Figure 7.6: Communication Channel

The channel starts in a *Wait* state. When a component sends a *approach* (CA_2), *clear* (CA_4) or *reply* control action (CA_1), it sends a synchronisation message on the respective synchronisation channel (*s_approach*, *s_clear* or *s_reply*). The channel automaton synchronises with the sender and makes a transition towards an intermediate state. While the channel automaton is in the intermediate state, the message has been sent but has not been received yet. This allows an

adversary to drop the message or change its payload before it is received. If the message is not dropped as a consequence of an adversarial action, the channel synchronises with the receiver on the respective synchronisation channel (*r_approach*, *r_clear* or *r_reply*). On this synchronisation, the receiver copies the value of the payload from the shared buffer and evaluates it. Modelling functional channels through dedicated automata enables to split the send and receive phases of the communication between two automata in two subsequent transitions. By doing so we can verify safety properties related to attacks on the integrity and the availability of the information flow, such as altered, missing or delayed communications.

Observers In Figure 7.7 we show the observers for control actions CA_2 and CA_4. In our model, two instances of the observer monitor unsafe applications of CA_2 and CA_4 in the two trains.

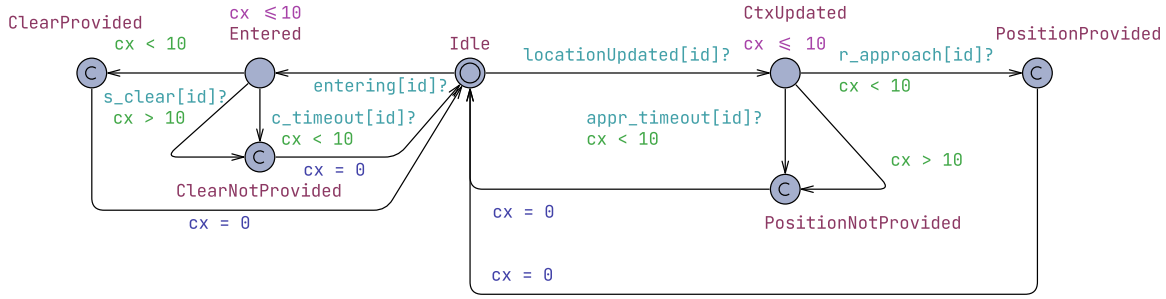


Figure 7.7: Observer 1

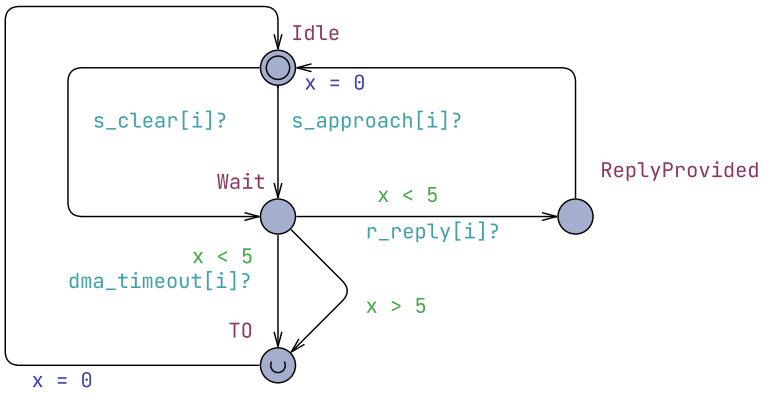


Figure 7.8: Observer 2

The automaton in Figure 7.7 starts in a *Idle* state and, when the system enters a context where the control action is expected it moves to *Entered* (if CA_4 is expected) or *CtxUpdated* (if CA_2

is expected). If the adversary *disrupts* the control action or if the control action is not issued within 10 time units (i.e., the adversary disrupted the process model, the controller or other elements in the control loop), the automaton enters a *NotProvided* state. In Figure 7.8, we show the observer for control action CA_1. The structure of this observer is analogous to the observer in Figure 7.7.

Verification Using the UCAs of the CBTC, and the safe behavioural model of its implementation we define the following TCTL specifications that verify UCAs:

UCA_1, UCA_4 - $\phi_1 := E <> CA1Monitor.NotProvided \text{ and } clk < \bar{t}$

Explanation: The ZC must always reply to CA_2. A violation occurs if it is not provided within $\bar{t}$ from CA_2.

UCA_2 - $\phi_2 := E <> zones[i] == j$ and $t\_id \neq j$ $loc[j] == loc[zones[i-1]]$ and $ZC.ReplyProvided$ and $reply == allow$

Explanation: Find an example of trace where, if i-th zone is engaged by train j the ZC replies *allow* to an access request for the same zone incoming from a train with an *id* that is different from the one in the zone i.

(UCA_3, UCA_6) - $\phi_3 := E <> CA1Monitor.Timeout$

Explanation: The automaton that monitors the time delay between CA_2 and CA_1 reaches timeout.

UCA_5 - $\phi_4 := E <> zone_i == 0$ and $t\_id = k$ and $ZC.Busy$

Explanation: Finds examples where a controller infers that a zone is "busy" when the zone is not occupied by any train.

UCA_7 - $\phi_5 := E <> CA2Monitor.NotProvided \text{ and } clk < \bar{t}$

Explanation: Control action CA2 is not provided within $\bar{t}$ from a change of location.

UCA_8 - $\phi_6 := E <> (trains.Approach \text{ and } area\_code! = location)$

Explanation: The train communicates a location that is different from its actual current location to the zone controller.

$UCA_{15} - \phi_7 := E <> (trains.Clear\ and\ area\_code! = previous\_location)$

Explanation: The train asks the controller to clear an area different from that it has just left.

$UCA_{17} - \phi_8 := E <> (zone_i == j\ and\ trains(k).Enter)$

Explanation: Train Controller commands to proceed through a zone already occupied by another train.

These safety properties are satisfied by the safe behavioural model of the CPS in the absence of an adversary. To this extent, we have verified that our model checker does not find any traces that violate the above TCTL specifications on the initial model. In the next sections, we will investigate what happens in adversarial conditions.

7.1.3 Attacker synthesis

We have applied STRIDE (Chapter 5) to the safe control structure of the CBTC system (Figure 7.1) and determined the adversarial actions (AA) in Table 7.3.

In particular, we have considered Spoofing, Tampering, ID and DoS threats against control action CA_1. For each tampering and spoofing threat and for each significant[1] value to inject, *allow* and *deny* in this case, we have instantiated an adversarial action. We have considered similar threats to affect also CA_2 and CA_4. We have assumed that RTK messages are signed and cannot be tampered with or spoofed, but they can be disrupted (a_{14}). However, the communication with the GNSS can be spoofed. We introduce a dependency between information disclosure and tampering i.e., we assume that the communication protocol requires an attacker to read a message before modifying it. Three further tampering actions can be carried out against the

[1]STPA provides values r for which a UCA is hazardous.

Table 7.3: Resulting Attack Library

#	Object	Type	Value	Privileges
a_1	ZC : Zone[i]	Tamper	$[0-1]$	(p_{10}, p_{11}, p_{12})
a_2	LU : GNSS	Tamper	$[1-5]$	p_{13}
a_3	CA_1 : Auth	Spoof	$[0-1]$	p_2
a_4	CA_1 : Auth	DoS	-	$(p_{14}), (p_8)$
a_5	CA_1 : Auth	Disclosure	$[0-1]$	$(p_{10}), (p_{11}), (p_{12}), (p_2)$
a_6	CA_1 : Auth	Tamper	-	$(p_{10}, p_{11}, p_{12}), (p_2)$
a_7	CA_2 : Zone	Spoof	$[1-5]$	p_7
a_8	CA_2 : Zone	DoS	-	$(p_{14}), (p_8)$
a_9	CA_2 : Zone	Tamper	$[1-5]$	$(p_3, p_6, p_5), (p_7)$
a_{10}	CA_2 : Zone	Disclosure	-	$(p_3), (p_6), (p_5), (p_7)$
a_{11}	CA_4 : Clear	Spoof	$[1-5]$	(p_7)
a_{12}	CA_4 : Clear	Tamper	$[1-5]$	$(p_3, p_6, p_5), (p_7)$
a_{13}	F_3 : GNSS	Spoof	$[1-5]$	(p_9)
a_{14}	F_2 : RTK	Dos	-	$(p_4), (p_1)$
a_{15}	LU RTKOn	Tamper	*true*	p_{13}

process model variables held in the zone controller (a_1) and the location unit (a_2, a_{15}). Each adversarial action in Table 7.3 is mapped to one or more sets of privileges.

Privileges in $\{p_1, \ldots, p_{14}\}$ are obtained through the exploitation of critical vulnerabilities within the system architecture. These are vulnerabilities affecting components responsible for the exchange of control actions and feedback. We have synthesised the model of adversarial actions (MAA) and shown it in Figure 5.8 (Chapter 5). In Figure 7.9, we show the implementation of the model of adversarial actions that we synthesised following the application of STRIDE to the control structure of the CBTC system. The attacker starts in a *Idle* state. At each time step,

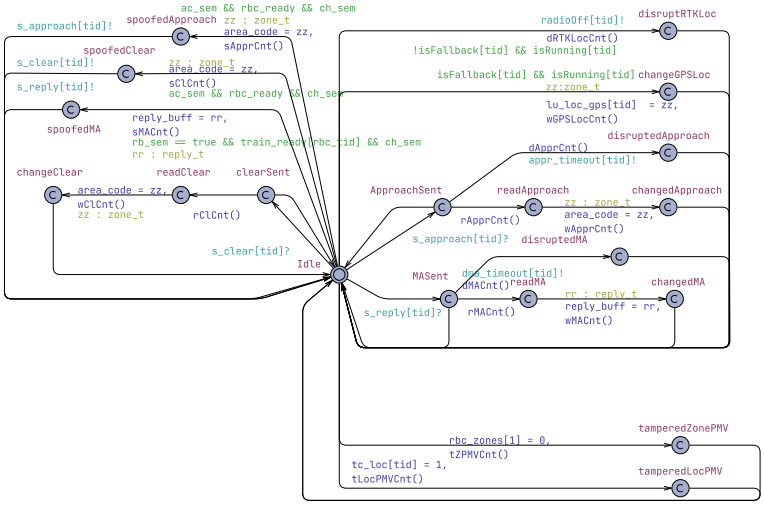


Figure 7.9: Model of Adversarial Actions (CBTC)

the attacker can non-deterministically perform an adversarial action, i.e., execute a threat. The MAA synchronises with the behavioural model when a control action (or feedback) is exchanged between two components. From there, the MAA non-deterministically chooses whether to perform the attack or not. The attacker chooses the next action non-deterministically and after each action, the attacker returns to the initial location. We used *committed* locations to synchronise control action and feedback when they are exchanged. This is an artefact introduced by UPPAAL.

7.2 Discovery & Enumeration of threat scenarios

In this section, provide an example of the application of Cassandra to identify and enumerate threat scenarios caused by unsafe control actions $UCA_1, \ldots, UCA_{18}$. We use our toolchain to verify safety specifications $\phi_1, .., \phi_8$ defined earlier in this chapter. The outcome of the verification with a time-bound set to $\bar{t} = 500s$ is shown in Table 7.4. For each specification

Table 7.4: Attack traces violating safety specifications. Size of trace set (TS). Size of reduced trace set (RTS). Verification time (TE).

Hazard	Spec	TS	RTS	Threat Scenarios	TE
H_4	ϕ_1	1	1	$\{a_4\}$	$500.14s$
H_1	ϕ_2	7	7	$\{a_{10}, a_9\}$, $\{a_{14}, a_{13}\}$, $\{a_{15}, a_{13}\}$, $\{a_7\}$, $\{a_1\}$, $\{a_{11}\}$	$500.93s$
H_4	ϕ_3	1	1	$\{a_4\}$	$500.17s$
H_2	ϕ_4	2	2	$\{a_4\}$	$500.3s$
H_4	ϕ_5	1	1	$\{a_8\}$	300.15
H_1, H_3, H_5	ϕ_6	8	4	$\{a_{10}, a_9\}$, $\{a_4\}$, $\{a_{15}\}$	$962.3s$
H_3, H_5, H_6	ϕ_7	6	6	$\{a_{10}, a_9\}$, $\{a_5, a_6\}$, $\{a_{14}, a_{13}\}$, $\{a_{15}, a_{13}\}$, $\{a_{11}\}$, $\{a_2\}$	$537.3s$
H_1	ϕ_8	10	6	$\{a_5, a_6\}$, $\{a_{10}, a_9\}$, $\{a_{14}, a_{13}\}$, $\{a_{15}, a_{13}\}$ $\{a_2\}$, $\{a_3\}$	$864.11s$

ϕ we report: the number of threat scenarios found (TS), the number of threat scenarios after elimination of redundant attacks (RTS), the respective hazards and threat scenarios and the time taken to verify each query ϕ_i and those derived from it i.e., the time elapsed (TE) (Table 7.4).

Verification of single specification In the listing below, we show the output of our program when verifying the specification ϕ_2, which verifies the reachability of UCA_2.

```
UCA2: CA1 provided with allow when zone[i] is engaged
```

In the case of the zone $i = 1$, the safety specification becomes

```
φ2 = E<> (zones[1] == 2 and rbc_tid != 2 and loc[1] == loc[0] and RBC.
ReplyProvided and rbc_tmp_reply==1)
```

In Listing 7.2 we show the output of our program that uses UPPAAL APIs to verify ϕ_2 (with $\bar{t} = 500s$).

```
[User input] Type query: E<> (zones[1] == 2 and rbc_tid != 2 and loc[1] ==
   loc[0] and RBC.ReplyProvided and rbc_tmp_reply==1)

[info] Query sent: E<> (zones[1] == 2 and rbc_tid != 2 and loc[1] == loc[0]
   and RBC.ReplyProvided and rbc_tmp_reply==1)

[info] Adversarial actions: Spoof (Approach)
[info] Next Query: E<> (zones[1] == 2 and rbc_tid != 2 and loc[1] == loc[0]
   and RBC.ReplyProvided and rbc_tmp_reply==1) and !(cain.sApproachBool)

[info] Adversarial actions: Tamper (ZC.Zones)
[info] Next Query: E<> (zones[1] == 2 and rbc_tid != 2 and loc[1] == loc[0]
   and RBC.ReplyProvided and rbc_tmp_reply==1) and !(cain.sApproachBool) and
    !(cain.tamperedZPMVBool)

[info] Adversarial actions: Read (Approach) Tamper (Approach)
[info] Next Query: E<> (zones[1] == 2 and rbc_tid != 2 and loc[1] == loc[0]
   and RBC.ReplyProvided and rbc_tmp_reply==1) and !(cain.sApproachBool) and
    !(cain.tamperedZPMVBool) and !(cain.rApproachBool and cain.wApproachBool
   )

[info] Adversarial actions: Disrupt (RTK Loc) Spoof (GPS Loc)
[info] Next Query: E<> (zones[1] == 2 and rbc_tid != 2 and loc[1] == loc[0]
   and RBC.ReplyProvided and rbc_tmp_reply==1) and !(cain.sApproachBool) and
```

```
    !(cain.tamperedZPMVBool) and !(cain.rApproachBool and cain.wApproachBool
   ) and !(cain.dRTKLocationBool and cain.wGPSLocationBool)

[info] Adversarial actions: Tamper (LU.GNSSOn) Spoof (GPS Loc)
[info] Next Query: E<> (zones[1] == 2 and rbc_tid != 2 and loc[1] == loc[0]
   and RBC.ReplyProvided and rbc_tmp_reply==1) and !(cain.sApproachBool) and
    !(cain.tamperedZPMVBool) and !(cain.rApproachBool and cain.wApproachBool
   ) and !(cain.dRTKLocationBool and cain.wGPSLocationBool) and !(cain.
   LuTamperBool and cain.wGPSLocationBool)

[info] Adversarial actions: Spoof (Clear)
[info] Next Query: E<> (zones[1] == 2 and rbc_tid != 2 and loc[1] == loc[0]
   and RBC.ReplyProvided and rbc_tmp_reply==1) and !(cain.sApproachBool) and
    !(cain.tamperedZPMVBool) and !(cain.rApproachBool and cain.wApproachBool
   ) and !(cain.dRTKLocationBool and cain.wGPSLocationBool) and !(cain.
   LuTamperBool and cain.wGPSLocationBool) and !(cain.sClearBool)



[info] Trace reply: (Timeout - 500s)
[info] Adversarial actions:
[info] Next Query: E<> (zones[1] == 2 and rbc_tid != 2 and loc[1] == loc[0]
   and RBC.ReplyProvided and rbc_tmp_reply==1) and !(cain.sApproachBool) and
    !(cain.tamperedZPMVBool) and !(cain.rApproachBool and cain.wApproachBool
   ) and !(cain.dRTKLocationBool and cain.wGPSLocationBool) and !(cain.
   LuTamperBool and cain.wGPSLocationBool) and !(cain.sClearBool)

[+] 6 attack traces have been found in 501.295s (8.354916666666666 min)
   (1) Read (Approach) ((-- AND --)) Tamper (Approach)
   (2) Disrupt (RTK Loc)  ((-- AND --)) Spoof (GPS Loc)
   (3) Tamper (LU.GNSSOn)  ((-- AND --)) Spoof (GPS Loc)
   (4) Spoof (Approach)
   (5) Tamper (ZC.Zones)
   (6) Spoof (Clear)

Removing redundancies...
```

```
(1) Read (Approach)  ((-- AND --)) Tamper (Approach)
(2) Disrupt (RTK Loc)  ((-- AND --)) Spoof (GPS Loc)
(3) Tamper (LU.GNSSOn)  ((-- AND --)) Spoof (GPS Loc)
(4) Spoof (Approach)
(5) Tamper (ZC.Zones)
(6) Spoof (Clear)
```

Listing 7.2: Execution of Threat Scenario Enumeration Algorithm

The *completeness* of the algorithm implemented is discussed in Section 5.3. Its complexity depends on the complexity of model-checking operations. We rely on UPPAAL to perform such operations, using symbolic verification along with a breadth-first search approach to explore the state space to verify the reachability of queries. To limit memory explosion problems usually affecting model checking, we bound our verification to $\bar{t}$ seconds.

Threat scenarios tree Hazards H_1, H_3, and H_6 lead to a *collision* event. We enumerate attacks leading to these hazards through the verification of ϕ_2, ϕ_6, ϕ_8 and find, respectively, 7, 3, 6, and 6 attacks. Overall, we have identified 11 distinct ways an attacker can cause a *collision* between two trains (some of the 22 traces enumerated in Table 7.4 appear more than once). In Figure 7.10, we show the threat scenario three for a collision event, which we have uncovered through model checking ϕ_2, ϕ_6 and ϕ_8. For brevity, we only expanded the branches related to H_1 and UCA_2. We have represented in light grey the *or* gates that we have not expanded. The unsafe control action UCA_2 can be caused by the adversary which exploits, individually, threats a_1, a_7 or a_{11}, or through the combined exploitation of $\{a_9, a_{10}\}$, $\{a_{13}, a_{14}\}$, or $\{a_{14}, a_{15}\}$. Such information is also available and summarised in Table 7.4.

7.3 Analysis of safety-critical attack paths

In this section, we leverage the mappings between the privileges on system architecture and adversarial actions, i.e., instances of threats, to identify the minimal safety set (MSS) related

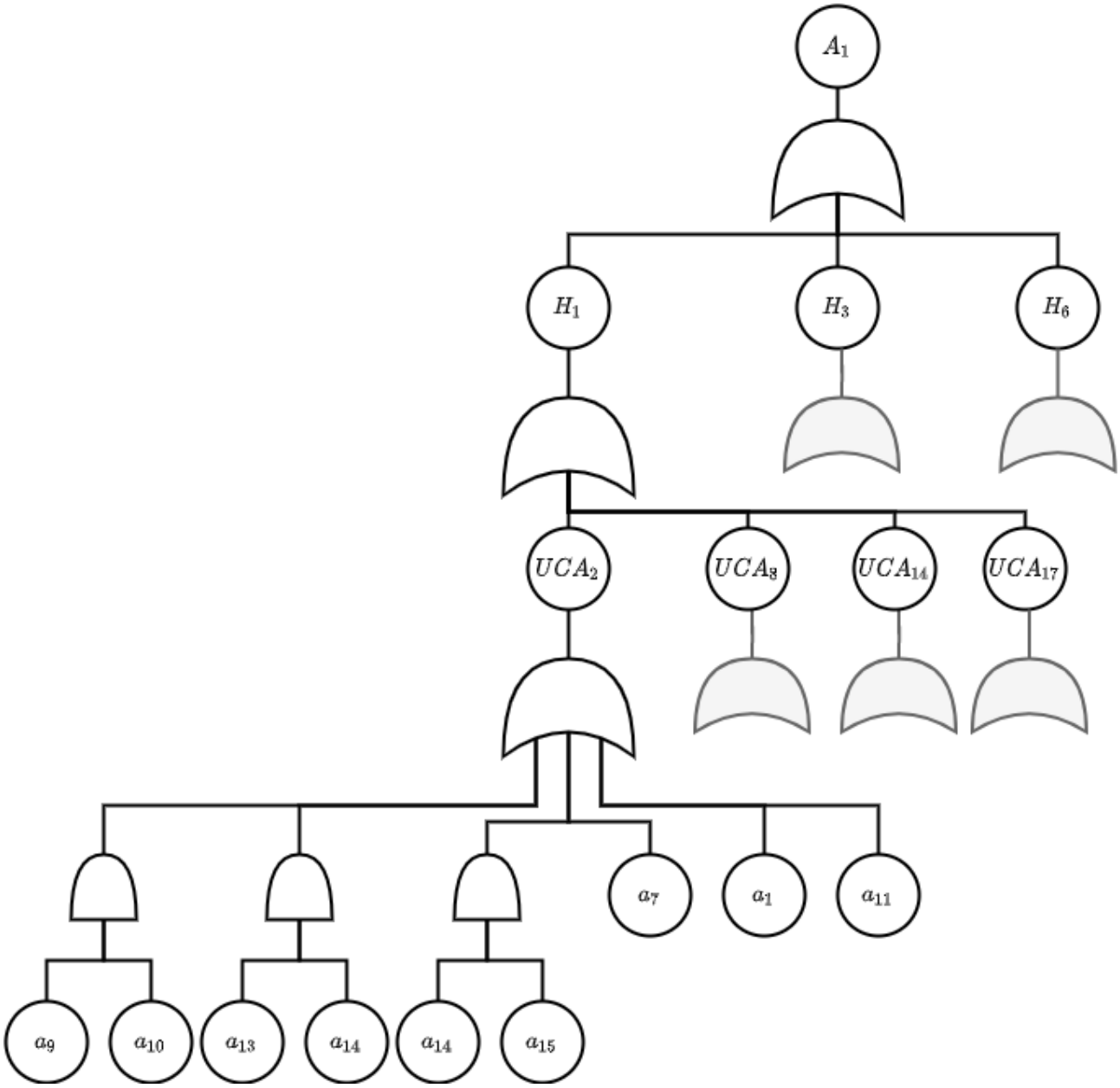


Figure 7.10: Threat scenarios tree (A_1)

to hazards H_1, H_3, and H_6. We then use MulVAL to generate a system-wide logical attack graph that encompasses paths leading to privileges in the MSS. We then analyse attack paths to identify security measures. The content presented in this section is a recap of the attack path analysis presented in Section 6.3.3.

7.3.1 Architecture

We have designed our implementation of the architecture for the CBTC (Figure 7.11). It features three layers: enterprise, ground control, and mobile. The enterprise layer consists of *three* interconnected sub-networks, including both workstations and servers. The ground network ensures the infrastructure control operations and comprises three modular redundant (TMR) [LV62] implementations of the zone controller (RBC) that orchestrates access to the shared infrastructure and a GSM receiver/transmitter module that also acts as arbitrator of the TMR. Each train has its own local network that hosts a TMR implementation of the train controller, a GSM module, the GNSS and radio modules for positioning and the HMI.

For brevity, we omit components not connected to the TPC/IP layer, such as speed controller, odometer and actuation units, as they communicate with the train controller over a dedicated bus (i.e. CAN), which we have assumed being out of the reach of the adversary. The architec-

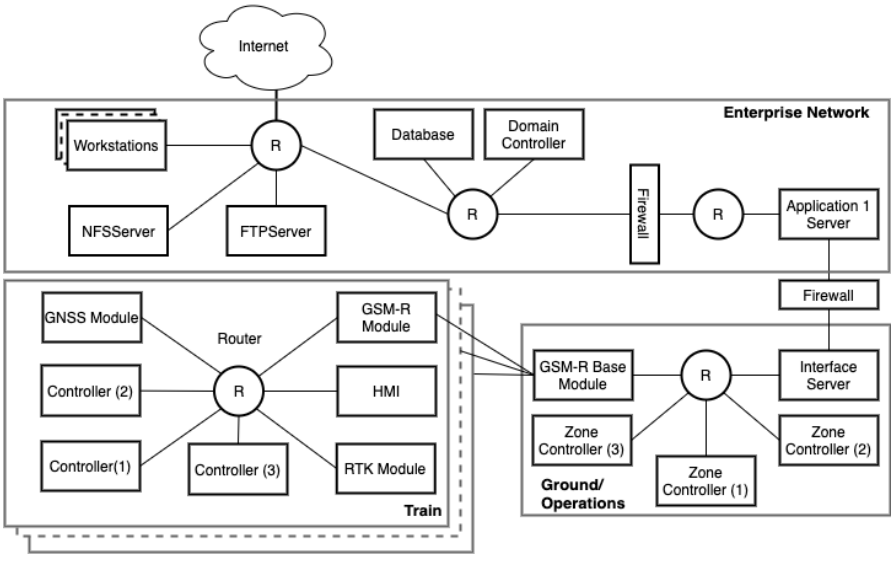


Figure 7.11: Communication-Based Train Control System (Overview)

ture is affected by 54 known vulnerabilities (`Vuln-01`, `Vuln-02`, ..., `Vuln-54`), which include generic Remote Code Execution (RCE) and Local Privilege Escalation (LPE). Among these, 36 vulnerabilities affect components within the mobile and ground layers. The exploitation of the latter grants access to privileges $\{p_1, ..., p_{14}\}$ (Table 7.5). Our objective is to identify which of the 54 vulnerabilities can lead to a safety-critical attack. In particular, we consider two types of privileges:

- **execCode(**c, u**)**: Privileges of code execution on component *component* with user *user*.

- **netAccess(**h, p, *protocol*)**: Ability to exchange network packets with the host h, on port p and protocol *protocol*.

As an example, the adversary that acquires the privilege P_1 can execute code as *root* on the *RTKModule* of train t_0. Similarly, P_4 enables to exchange packets with *RTKModule* of train t_0 on any port (_) and any protocol (_).

Table 7.5: Privileges on train and ground networks components

#	**Privilege**
P_1	$execCode(t_0.RTKModule, root)$
P_2	$execCode(Ground.GSMModule)$
P_3	$execCode(t_0.Controller(1), root)$
P_4	$netAccess(t_0.RTKModule, \_, \_)$
P_5	$execCode(t_0.Controller(2), root)$
P_6	$execCode(t_0.Controller(3), root)$
P_7	$execCode(t_0.GSMModule, root)$
P_8	$netAccess(Ground.GSMModule, \_, \_)$
P_9	$netAccess(t_0, .GPSModule, radio, \_)$
P_{10}	$execCode(Ground.Controller(1), root)$
P_{11}	$execCode(Ground.Controller(2), root)$
P_{12}	$execCode(Ground.Controller(2), root)$
P_{13}	$execCode(t_0.LocationUnit, root)$
P_{14}	$netAccess((t_0.GSMModuleModule, \_, \_)$

7.3.2 Minimal safety set

We now use Z3 to find a minimal safety set $\mathcal{S}$ for which the Boolean function $\mathcal{H}$ (Equation 7.1) is false (i.e. a collision cannot be caused).

$$\mathcal{H} = (H_1 \vee H_3 \vee H_6) \tag{7.1}$$

As the set of privileges is contained in size, the solver can completely enumerate all the 9 minimal safety sets $\mathcal{S}_1, \ldots, \mathcal{S}_9$ in $< 1s$ We list below the minimal safety sets for the CBTC use case. Note that privileges in the set $\{p_2, p_7, p_8, p_9, p_{14}\}$ are common across all the $\mathcal{S}_i$ found in this use case. In this sense, it is crucial to address vulnerabilities leading to these privileges if we want to prevent the adversary from causing a *collision*. From the same sets, we also notice that the defender must protect unauthorised access to any of the privileges in the sets $\{p_3, p_5, p_6\}$ and $\{p_{10}, p_{11}, p_{12}\}$ to prevent a *collision*.

Set	Privileges in set
$\mathcal{S}_1$	$p_2, p_3, p_7, p_8, p_9, p_{11}, p_{13}, p_{14}$
$\mathcal{S}_2$	$p_2, p_6, p_7, p_8, p_9, p_{10}, p_{13}, p_{14}$
$\mathcal{S}_3$	$p_2, p_5, p_7, p_8, p_9, p_{10}, p_{13}, p_{14}$
$\mathcal{S}_4$	$p_2, p_6, p_7, p_8, p_9, p_{11}, p_{13}, p_{14}$
$\mathcal{S}_5$	$p_2, p_5, p_7, p_8, p_9, p_{11}, p_{13}, p_{14}$
$\mathcal{S}_6$	$p_2, p_3, p_7, p_8, p_9, p_{12}, p_{13}, p_{14}$
$\mathcal{S}_7$	$p_2, p_6, p_7, p_8, p_9, p_{12}, p_{13}, p_{14}$
$\mathcal{S}_8$	$p_2, p_3, p_7, p_8, p_9, p_{10}, p_{13}, p_{14}$
$\mathcal{S}_9$	$p_2, p_5, p_7, p_8, p_9, p_{12}, p_{13}, p_{14}$

Table 7.6: Minimal safety sets CBTC system

7.3.3 Graph path analysis

Finally, we employ MulVAL to build the attack graph for the system, assuming that the attacker can establish a connection with any of the workstations in the enterprise network, e.g., as a result of phishing or through a physical backdoor [BBC19]. At the same time, we also assume that they can carry out a GNSS spoofing attack. To carry out an attack leading to a safety

violation, the attacker must obtain privileges $p_1, \ldots, p_{14}$ (Table 7.5). We use these privileges as targets for generating the attack graph with MulVAL.

The attack graph (Figure 7.12) includes 333 nodes, of these 170 are *and* nodes, 45 are *or* nodes, and 118 are basic facts (e.g., security conditions, network conditions, etc.). For improved readability, we have partitioned the attack graph in *seven* colour-coded regions, zone 1 (grey), zone 2 (blue), zone 3 (red), zone 4 (yellow), zone 5 (green), zone 6 (light blue), and zone 7 (purple). These zones are for visual clarity and do not correspond to any semantic divisions within the graph. Figures reporting the structure of each zone in greater detail are shown in Appendix *A*.

From a first high-level analysis of the attack graph, we notice the existence of *four* clusters, one for each network area. In particular, zone 1 counts two clusters (Figure A.2). A first cluster includes privileges, network conditions, and exploitation rules from the enterprise network. The exploitation of vulnerabilities `Vuln-03` (RCE) and `Vuln-04` (RCE) provides the attacker with a *foothold* in the enterprise network of the system. The second cluster represented within zone 1 is a smaller component disconnected from the rest of the graph (top right, in Figure A.2), which shows the attack path towards a GNSS spoofing attack. We assume that any adversary who can access a powerful enough transmitter can spoof the GNSS signal. A third cluster (zones 2, 3) relates to the ground/operations network. This is accessible through the exploitation of `Vuln-15` (RCE), which grants execution privileges on the *Interface Server (Operation)*. In fact, from the *Interface Server (Operation)* the attacker gains network access to all devices in the operation network (p_8), and can execute lateral movements towards the he GSM-Base and the zone controllers, respectively, $p_2, p_4, p_8, p_{10}, p_{11}$, and p_{12}. A further RCE `Vuln-26` allows, if exploited, the adversary to reach the fourth cluster (zones 4, 5, 6, and 7), which coincides with the on-board network. From here, the adversary gains network visibility of onboard networked components and can obtain the remaining safety-critical privileges ($p_1, p_3, p_5, p_6, p_7, p_9, p_{13}$, and p_{14}) through further lateral movements.

Identifying which vulnerabilities should be remediated requires an enumeration of the attack graphs and is challenging given its depth. To solve the problem, we select one of the minimal

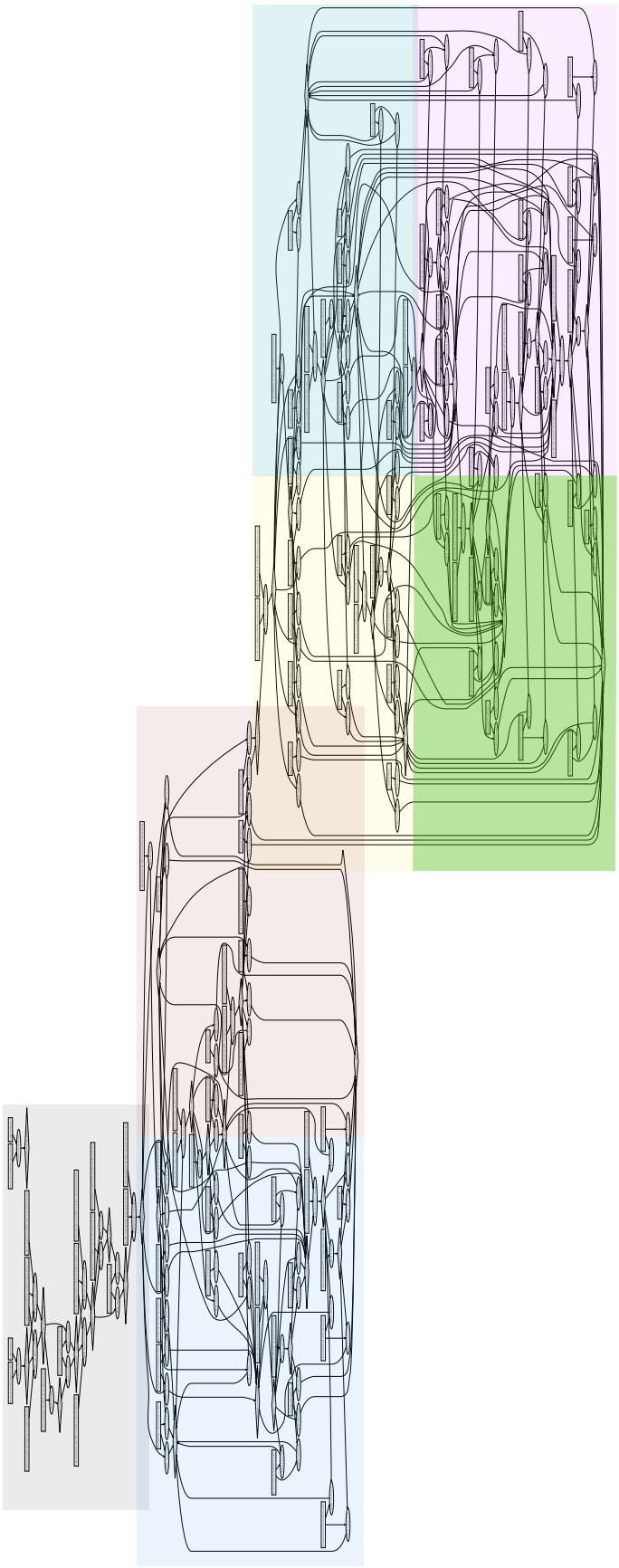


Figure 7.12: Attack graph (CBTC)

safety sets that satisfies $\mathcal{H} = false$, and use JGrapht to evaluate, individually, the paths leading to the privileges in the set. For each path, we count the occurrences of AG nodes that are common among all paths.

Our program identified 119353 potential graph paths in the graph in $< 5s$. However, some nodes shown below occur very frequently, in all or most of the paths. We analysed the number of occurrences of AG nodes across all the paths (Figure 7.13) and we found the following privileges to be common to all (or most of) the paths

```
netAccess(opgInterfaceServer,_,_)
execCode(opgInterfaceServer,root)
execCode(application_server_1,root)
```

Listing 7.3: Common nodes

These privileges were not part of the initial set $\{p_1, \ldots, p_{14}\}$ but are necessary to reach them.

We arbitrarily select p^*=*execCode(opgInterfaceServer, root)* and apply our backward enumeration algorithm to find all the attack paths leading to p^*. This is feasible now since p^* is closer to the perimeter than the privileges in the initial set $\{p_1, \ldots, p_{14}\}$. We identify the following vulnerabilities that are shared across the attack paths leading to p^*.

```
(opgInterfaceServer,Vuln-15,kernel,localExploit,privEscalation)
(applicationServer1,Vuln-08,vulnerable_web_app,remoteExploit,
 privEscalation)
(workstationHost10,Vuln-03,rdp_server,remoteExploit,privEscalation)
```

Listing 7.4: Common vulnerabilities

p^* is common to all but one *graph paths* found in the original attack graph, from the perimeter to $\{p_1, \ldots, p_{14}\}$. We fix the five vulnerabilities above and generate a new attack graph for the entire system. Since p^* was a common node, by removing it we remove almost but one paths leading to the minimal safety set. In fact, the new AG contains only one attack path, which is the one leading to p_9 which, together with any privilege in the set $\{p_1, p_4, p_{15}\}$ can lead to a *collision*. This corresponds to a GNSS spoofing attack against the system, which is not easy to

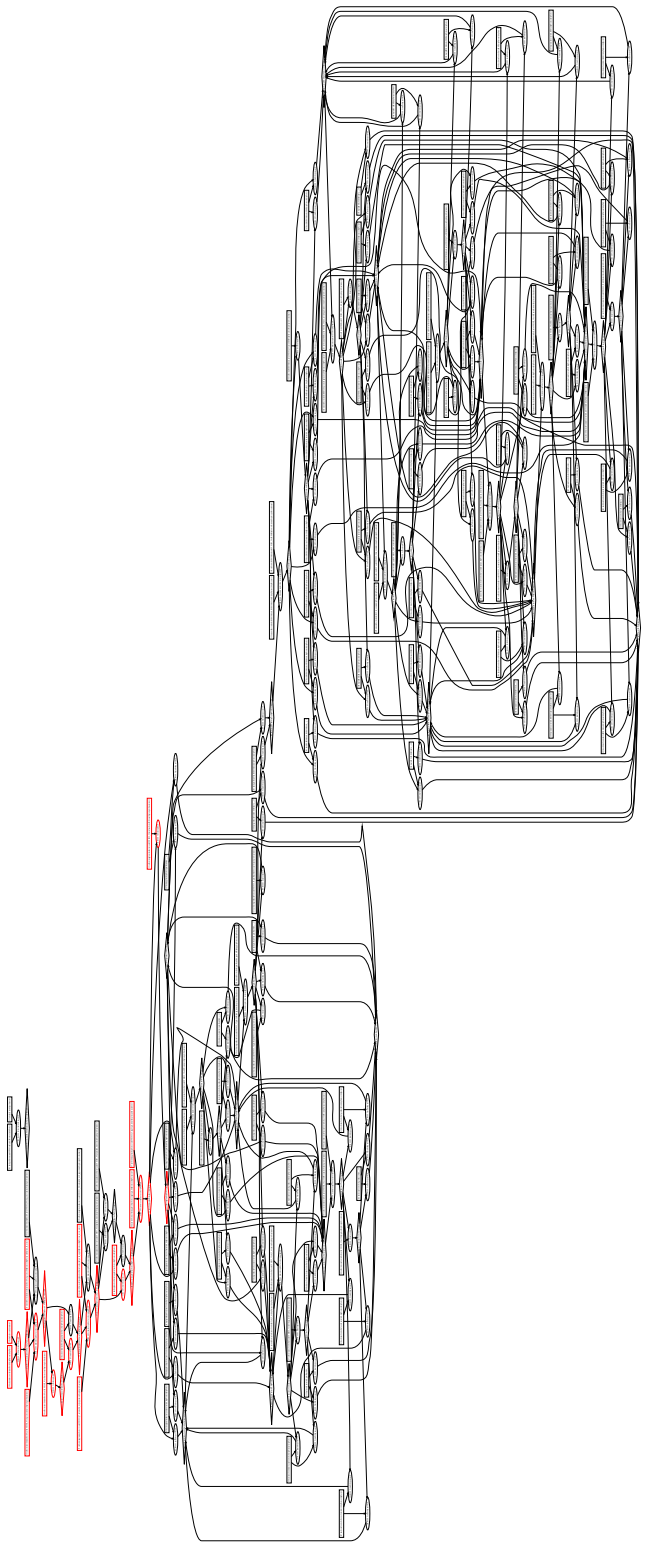


Figure 7.13: Attack graph (CBTC) - Common nodes

remediate. A possible solution is to remediate vulnerabilities leading to $\{p_1, p_4, p_{15}\}$. Protecting these privileges can prevent the attacker from disrupting the radio beacons (Table 7.3), making any eventual GNSS spoofing attempt not safety threatening anymore.

7.4 Discussion

By applying the first step of Cassandra, we have identified 11 distinct threat scenarios leading to a *collision* in our system. The key points for the integration between safety and security analyses at the functional level are the generation of a combined model of adversarial actions (MAA) from the results of the STRIDE threat analysis and the creation of a correspondence between the behavioural model of the system and its safe control structure. After combining the MAA with the behaviour of system components, we use formal verification to verify the safety properties we identify the sequences of adversarial actions that lead to the violation of the safety properties. The existence of a map between STRIDE threats and privileges (Table 7.3) in the deployment context enables to identify, in the second step of Cassandra , the minimal safety set, and generate a system-wide attack graph leading to privileges in the MSS. Finally, we identify via attack graph analysis specific vulnerabilities that should be addressed with priority (e.g., through patching, traffic isolation, etc.) in order to defend safety.

The attack graph analysis carried out during the final stage of Cassandra shows three main components, one associated with each layer in the CPS architecture. At the same time, by studying the occurrences of individual nodes in feasible attack paths, we have discovered that some of the nodes (Figure 7.13) are crucial to the successful exploitation of all the paths. By addressing shared vulnerabilities, we are able to cut paths leading to privileges in the MSS, thus guaranteeing safety. Finally, the analysis of the attack paths suggests that mitigating `CVE-2022-55` on *InterfaceServer* would prevent the attacker from reaching the *Ground Network* from *Enterprise*. At the same time, tearing *InterfaceServer* down, e.g., by cutting the link with *ApplicationServer* and *InterfaceServer*, or, by installing a network diode between the two, would sort the same effect, trading off security for usability. Similarly, we observe that applying a fix

to `CVE-2022-26` on the GSM module, would cut reachability in the attack graph with the *Train Network*. Furthermore, placing components that interface with external networks such as the *Interface Server* on a demilitarised zone (DMZ) would increase the difficulty to compromise *Zone Controller* and *GSM Base Module* at *Operation* layer, and thus the capability of carry out safety-critical attacks.

7.5 Summary & Conclusions

In this chapter, we have applied Cassandra to analyse the impact of adversarial threats targeting a representative Communication-Based Train Control System. We have uncovered a number of potential safety-critical threat scenarios and their related attack paths. The results presented in this chapter were published as a use case in [CL23].

Despite the inherent complexity of the CBTC system analysed in this chapter, it was designed and developed for the sole purpose of our research. While we have built it starting from the EU [Uni] specifications, it is possible that some real-world aspects might have been simplified or overlooked.

In the next chapter, we demonstrate the applicability of Cassandra on a more realistic system. In particular, we study the safe operation of a realistic industrial control system testbed which mimics a concrete power system. The analysis is aimed at discovering safety-critical attack paths and applicable security controls.

Chapter 8

Security aware hazard analysis for smart grids: a case study

As we witness Advanced Persistent Threat (APT) groups targeting the energy sector and developing malware such as BlackEnergy [Mit22], Crashoveride [Inc22], and Pipedream [Dra22], it becomes crucial to evaluate the safety consequences of attacks on power systems.

In this chapter, we analyse the safe operation of a smart grid system in adversarial conditions. The objective of the analysis is the generation of attack paths that lead to safety violations. The analysis of safety-critical attack paths allows defenders to identify safety-critical assets, e.g., components that are crucial for successful attacks (i.e., common nodes in attack paths), and focus hardening efforts towards these components.

Although incorporating security elements into the safety analysis of smart grids has been previously explored [FMS+17], in [CL20], to the best of our knowledge, we are the first to leverage the integration with the architecture of the power grid and generate attack paths as sequences of privileges that an attacker must obtain to cause an accident.

The functional and architectural models elaborated for this use case are based on those of the *Electric Power and Intelligent Control (EPIC)* test-bed [AKM18]. Behavioural models for the components were inspired by [SSD+17] and reviewed in collaboration with colleagues of the

Electrical Engineering Department at Imperial College.

This chapter is organised as follows. In Section 8.1, we provide an overview of the EPIC, identify system-level hazards, and model its safety control structure. In Section 8.2, we narrow our focus on a section of the power system, the microgrid. We identify unsafe control actions and model the behaviour of components in the microgrid. In this section, we also apply STRIDE to the safety model and synthesise a behavioural model for the adversary. Thus, in Section 8.3, we proceed with the enumeration of threat scenarios affecting the microgrid. In Section, 8.4, we generate the attack paths leading to the threat scenarios found and discuss the results in Section 8.5. Finally, in Section 8.6 we present the conclusions of this chapter.

Part of the content of this chapter was published in the proceedings of 2022 IEEE International Conference on Communications, Control, and Computing Technologies for Smart Grids (SmartGridComm) [CHC+22].

8.1 Testbed architecture and safety model

Smart grids are an example of emerging cyber-physical systems where the loss of safety as a result of successful attacks can lead to significant consequences. The Electric Power and Intelligent Control (EPIC) testbed is a smart grid infrastructure comprising *four* sections: power generation, transmission, microgrid and smart-home [AKM18]. The latter also includes a secure water treatment (SWaT) testbed (Figure 8.1). The physical process is controlled through a SCADA workstation, which is connected to a main (master) programmable logic controller (PLC) which, in turn, coordinates four PLCs, each of them responsible for a different section of the testbed. Each of the four PLCs interacts with the physical process through a group of Intelligent Electric Devices (IED). These latter measure the dynamics of the process and control it by opening and closing circuit breakers. The microgrid section also includes a Photovoltaic Array (PVArray), a battery energy storage, and two generators. These latter supply power when the microgrid is disconnected from the main transmission line and the batteries cannot meet the energy demand. Applying our analysis to EPIC allows us to model

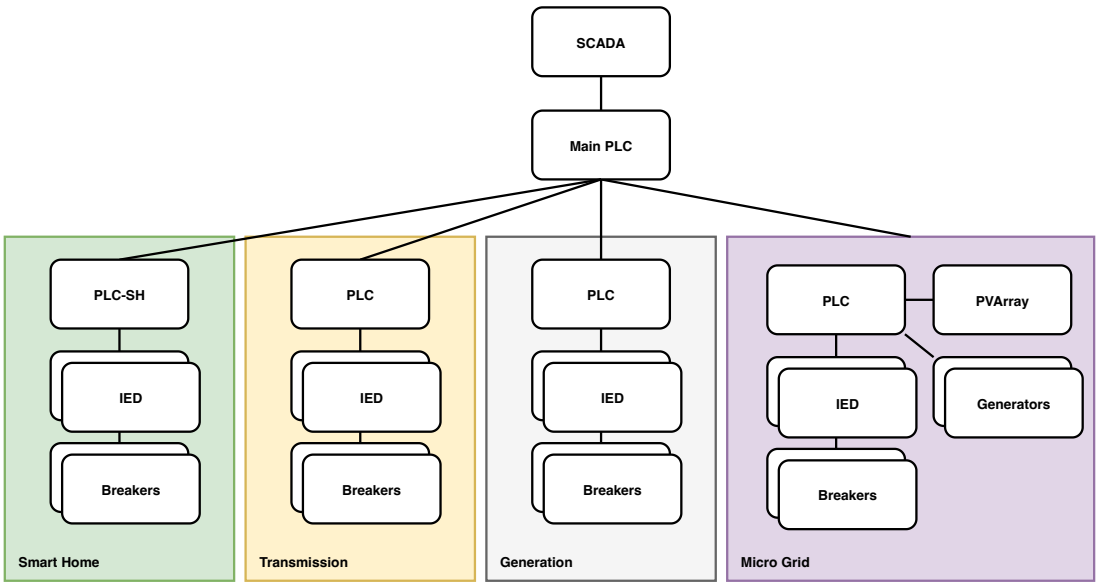


Figure 8.1: Electric Power and Intelligent Control (Overview)

realistic safety and security aspects of a smart grid. In addition, the attack paths generated can also be validated on the actual testbed.

Electrical layout The diagram in Figure 8.2 shows the electrical layout of EPIC with respect to the *four* loads: *critical* and *non-critical* loads, the *motor*$-G3$, and the output towards a water treatment plant (an external system). The critical load (shown in red in Figure 8.2) is located in the Smart Home (SH) section. Continuous power supply to the critical load (e.g.,

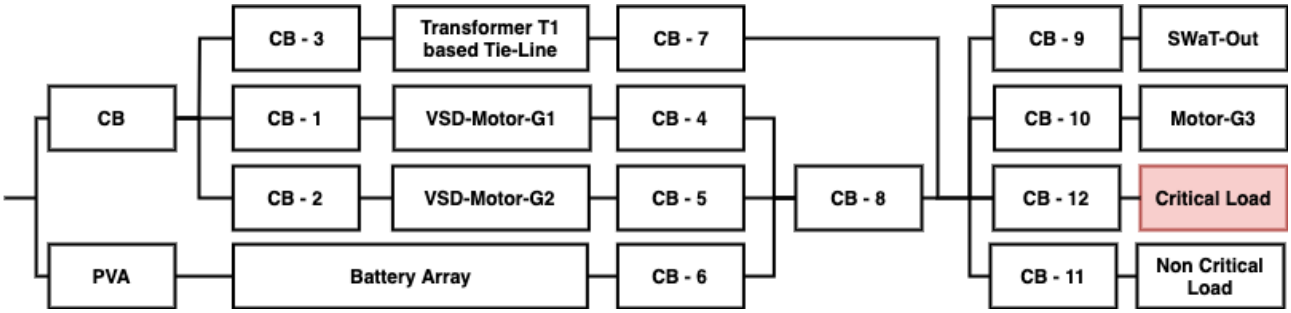


Figure 8.2: Power dependencies of the main loads in the EPIC testbed.

the intensive care unit ward of a hospital) is essential to maintain system safety. In particular, there are *four* parallel paths (Figure 8.2) supplying power to the critical load [AKM18]. *Three* of these paths are located in the microgrid (MG) section:

- $P_1 = CB, CB-1, CB-4, CB-8, CB-12$

- $P_2 = CB, CB-2, CB-5, CB-8, CB-12$
- $P_3 = CB-6, CB-8, CB-12$.

For MG to function in islanded mode, the power to the critical load must be provided by one of the two generators or by the battery array. Paths P_1 or P_2 are considered *active* if the power is supplied by the first or the second generator respectively. At the same time, P_3 is active if the load is powered through the battery. Note that for P_3 to be active, the battery array needs to have enough stored energy to satisfy the demand.

A $fourth$ path:

- $P_4 = CB, CB-3, CB-7, CB-12$

powers the load through a Transmission Network (TN) [AKM18]. The circuit breaker CB is located in the *generation* section of EPIC and it is controlled by a PLC ($gPLC$). Similarly, circuit breakers in the TN ($CB-3, CB-7$ and $CB-8$) are regulated by $tPLC$. Lastly, PLC located in the MG ($mPLC$) controls $CB-4$, $CB-5$ and the smart-home ($sPLC$) controls $CB-12$.

Safety model We employ STAMP to model the safety aspects of the testbed. First, we leverage expert knowledge to identify system-level hazards and relevant losses and the accidents that they can cause. We consider the following losses: (L_1) Loss of life, (L_2) Loss of safety of the operating environment, (L_3) Major damages to equipment, and (L_4) Economic losses.

Table 8.1: Table of accidents - Electric Power Grid

Acc	**Description**	Loss(es)
A_1	Interruption of power supply to *critical* load.	L_1
A_2	Nuisance tripping.	L_1, L_2
A_3	Physical damages to motors.	L_1, L_2, L_3, L_4
A_4	Physical damages to batteries.	L_2, L_3
A_5	Non synchronised bridge closure	L_2, L_3
A_6	Physical damages to IED.	L_3
A_7	Interruption of power supply to *non-critical* load.	L_4

Thus, the following accidents are considered. A_1, and A_2 lead, respectively, to L_1 (e.g., deriving from the power being cut to the critical load) and to L_1 and L_2. A_3 leads to losses L_1, L_2, L_3 and L_4. A_4 leads to losses L_2 and L_3, while A_5 leads to L_2 and L_3. Lastly, A_6 and A_7 lead to L_3 and L_4 respectively. $A_1 - A_5$ are the accidents with the highest losses as they threaten human lives and the safety of the operating environment. We summarise the accidents considered in Table 8.1.

The safety analysis proceeds with the investigation of system hazards that can lead to accidents $A_1 - A_7$. The hazards found are then refined to identify the responsible controllers. The objective of this step is the definition of safety requirements to enforce responsible controllers to avoid the hazards. For brevity, we focus on the hazards that can lead to accidents A_1, A_2 and A_7.

Following the analysis of electrical dependencies of the critical load (Figure 8.2), we find a total of 7 hazards that lead to A_1, A_2 and A_7 (Table 8.2). The safe control structure is shown in

Table 8.2: Hazards leading to accidents A_1, A_2 (Electric Power Grid).

Hazard	Description
H_1	Unexpected configuration of breakers
H_2	Loss of phase synchronisation.
H_3	Loss of view of Physical Process.
H_4	Failure to engage battery
H_5	Failure to engage photovoltaic array
H_6	Failure to alert non critical load
H_7	Reduced supply to non critical load

Figure 8.3 and consists of a total of 34 controllers, sensors and actuators. The control structure focuses on the control flow in the EMS and does not show the physical details of the process such as the flow of energy between power subsystems. Electrical dependencies in the physical process are shown in Figure 8.2. The EMS is supervised by a human operator who controls the functioning of the microgrid through the SCADA workstation, which in turn, controls the main PLC and the photovoltaic array (PVA). The main PLC acts as the coordinator for $four$ lower-level controllers, one for each stage of the EMS system. Each of these PLCs controls part of the EMS through dedicated Intelligent Electric Devices (IEDs). The IEDs are low-level controllers that measure frequency and voltage through an Advanced Measuring Interface

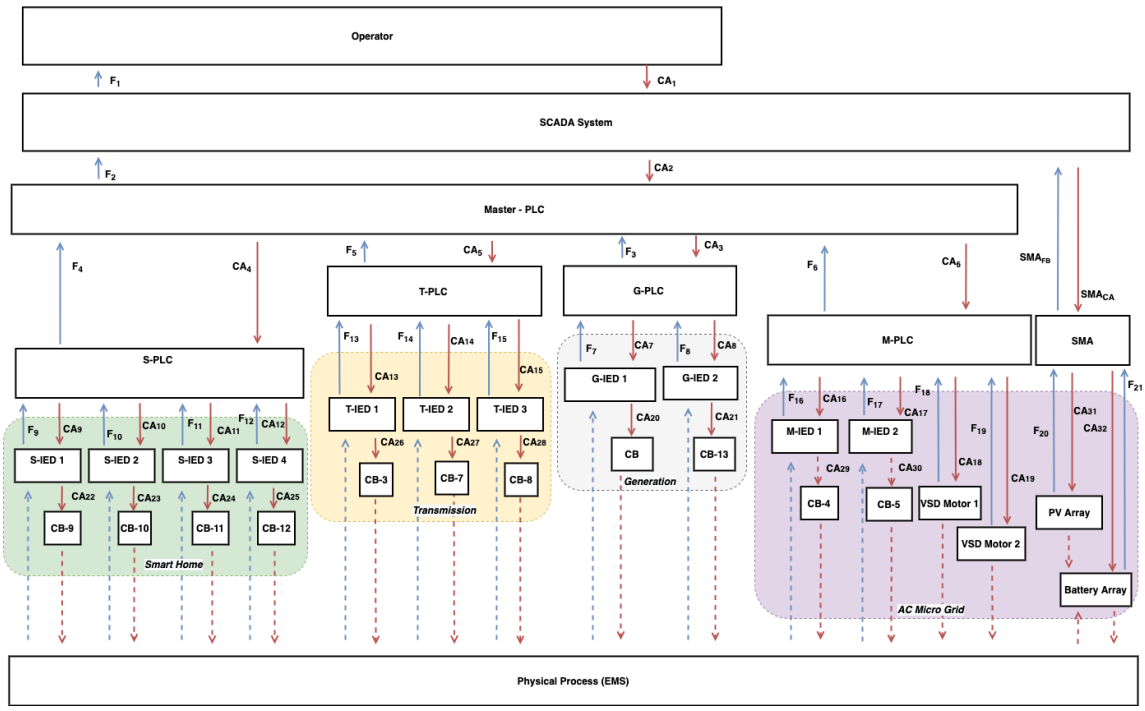


Figure 8.3: STAMP model of EPIC

(AMI) and control the physical network by opening and closing the 12 Circuit Breakers (CB). The IED can open a circuit breaker following an instruction received from the commanding PLC or adapt to frequency deviations. Circuit breakers $CB-1$, $CB-2$ and $CB-6$ are electrical safety interlocks, are not controlled by operators and are opened when an unbalance is detected. At the same time, CB, $CB-4$, $CB-5$, $CB-7$, $CB-8$ and $CB-12$ are controlled by operators, but they can also open automatically, as a result of nuisance tripping to preserve operational safety. An SMA inverter is controlled by the SCADA workstation and regulates the photovoltaic array in the microgrid. For simplicity, we do not show the AMI in the SCS shown in Figure 8.3. The process models of SCADA, PLCs and IEDs contain the variables and the control algorithms needed to regulate their respective stages of the EMS.

8.2 Analysis of the microgrid section

In this section, we provide a more detailed description of the processes involved in the control of the microgrid (MG) section of the EPIC testbed. The safe control structure of the MG is shown in Figure 8.4. For simplicity, we represent only one generator instead of the two 2 that

characterise the MG in EPIC.

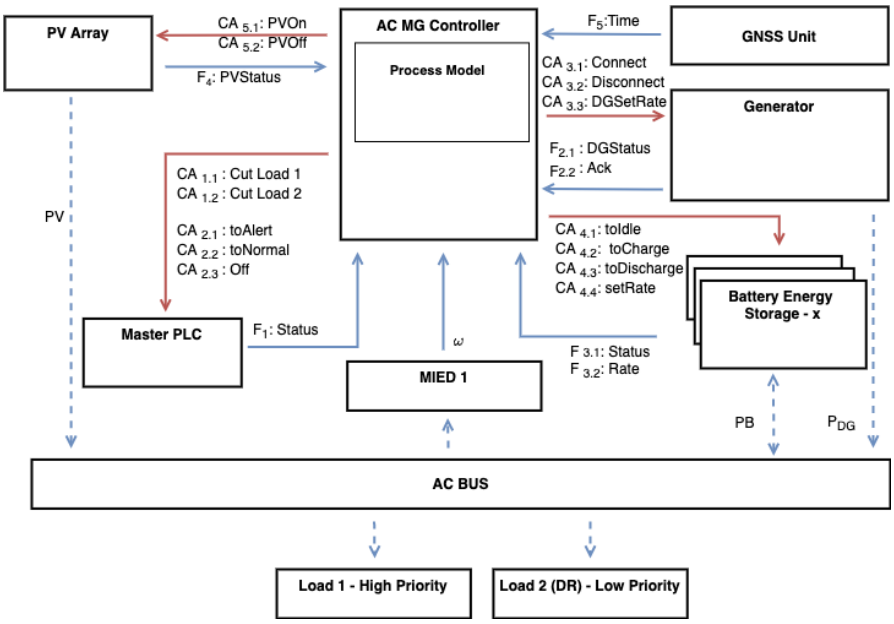


Figure 8.4: Safe Control Structure Microgrid

The safe control structure is a tuple (C, D, K) where C denotes the set of components (controllers, sensors, actuators and physical processes), D the set of *control actions* and *feedback*, and K denotes a set of *functional channels* [CL20]. Inside each controller, the variables of the *process model* retain the representation of the physical process and are used by the *control algorithm*. Given the absence of detailed documentation on the structure of the microgrid sector of the EPIC testbed and, most importantly, on the behaviour of its local controller, we have held a set of meetings with our colleagues in the EEE department at Imperial College to build the following realistic behavioural model, given our knowledge of the EPIC testbed. The behaviour of the components in the model was largely inspired by [SSD+17].

We assume that the MG controller relies on a set of local variables to store information, including the current frequency of the alternating current (AC) bus, the status of $BT-x$ and its current rate, the current rate of the generator, the current status of the loads and photovoltaic array. Bus frequency ω is defined *within limits* if its measurements are within boundaries specified by system designers; otherwise, the grid is unstable. Within these operational limits, a *low* frequency implies that power generation must be increased before the frequency drops further, whilst a *high* frequency implies an overproduction. An additional Boolean variable is

true if the phase of the microgrid is synchronised with the main grid. A vector of two Boolean variables summarises the status of the two loads. Each element is (*true* if the respective load is consuming power and *false* otherwise). When the frequency read on the AC bus drops below its *low* limit, the local (MG) controller keeps it within safe boundaries by engaging (in order) the photovoltaic array, the available batteries and, at last, the generator at an increasing rate. When the energy sources combined cannot satisfy the demand (frequency remains low), the local controller alerts the *Master PLC* to reduce the demand of *Load* 2. In the unfortunate case of the frequency remaining *low* even after that *Load* 2 was alerted, the controller requests the disconnection of *Load* 2 from the MG to preserve *Load* 1. The controller is eventually forced to request a detachment of *Load* 1 to preserve the safety of the infrastructure if the frequency does not stabilise after the disconnection of *Load* 2. We assume that, in the absence of an adversary, the generator is designed to sustain alone the combined demand of the two loads. The micro-grid controller uses control actions $CA_{1.1}$ and $CA_{1.2}$ to request the emergency opening of the breakers that support Load 1 and Load 2, respectively. At the same time, phase and voltage are read from *M-IED*1 (ω). Alerts to *(Load* 2*)* are sent to the *Master PLC* through control actions $CA_{2.1}, CA_{2.2}$, and $CA_{2.3}$. *Master PLC* returns its status through feedback F_1. For improved clarity in the diagram, we have hidden the circuit breakers CB-4 and CB-5 as well as the phase and voltage meter in the ACBus component and *M-IED*2. PV Array is switched on/off through control actions $CA_{5.1}$ and $CA_{5.2}$ and feeds back its status on F_4. The micro-grid controller controls the i-th battery energy storage (BES) subsystem through control actions $CA_{4.i.1}$, $CA_{4.i.2}$, $CA_{4.i.3}$ and $CA_{4.i.4}$ to respectively set it to *idle*, *charge mode*, *discharge mode* or to change the charging/discharging rate. The current status and rate of the i-th BES are fed back on $F_{3.i.1}$ and $CA_{3.i.2}$. The generator is engaged through commands on $CA_{3.1}$ and disengaged through commands on $CA_{3.2}$. The generator rate commands are issued on $CA_{3.3}$. On the other hand, feedback from the generator is exchanged on $F_{3.1}$ and $F_{3.2}$. We make the additional assumption that the MG is also equipped with a GNSS module for time-based synchronisation with the grid. Although this feature is not present in EPIC, we convened with our colleagues in the power system department that it is a feature common to MG scenarios. The GNSS module exchanges information with the MG controller through the feedback F_5.

We continue the application of STPA/ STPA-Sec and identify the Unsafe Control Actions (UCA). In this scenario, an example of UCA is the closure of a circuit breaker when the phases on either side differ. The unsafe action results in major damage to the infrastructure and can also harm nearby operators.

Table 8.3: Context table (CT) of the AC MG Controller. PE: Provided Early. PL: Provided Late.

CA	Frequency	Synch	DG	BT-1 State	BT-1 Rate	Status L2	Alert L2	**Not Provided**	**Provided**	**PE**	**PL**
$CA_{1.1}$	-	-	-	-	-	-		-	H_1	-	-
$CA_{1.2}$	Low	-	On (Max)	-	-	On (Reduced)	On	H_2	-	-	-
$CA_{1.2}$	-	-	-	-	-	On	Off	-	H_7	-	-
$CA_{2.1}$	Low	-	On (Max)	-	-	On	Off	H_2	-	-	-

In Table 8.3, we report an example of the context table for a subset of control actions of the microgrid controller. The context of unsafe applications of control actions is defined as a function of the process model variables previously discussed. The unsafe control actions are summarised in Table 8.4.

Table 8.4: Table of Unsafe Control Actions MicroGrid

#	**Description**	**Hazard**
UCA_1	$CA_{1.1}$ is always hazardous when provided and results in hazard. In fact, $CA_{1.1}$ causes the emergency shutdown of the critical load and is designed to be used only when an extremely low frequency on the grid can damage the critical load.	H_1
UCA_2	$CA_{1.2}$ is hazardous (H_2) if it is *not provided* when the frequency is *low*, the generator is *working at highest rate*, and the Load 2 is consuming at *reduce rate* having already being alerted. Not providing $CA_{1.2}$ in its context results in major damages to the microgrid and also in the shutdown of the critical load for insufficient power.	H_2
UCA_3	$CA_{1.2}$ is hazardous (H_7) if it is *provided* when the frequency is *low*, the generator is *working at highest rate*, and the Load 2 has *not been alerted* first. The correct procedure would be to alert Load 2 before issuing $CA_{1.2}$. In fact, providing $CA_{1.2}$ also results in a hazard.	H_7
UCA_4	$CA_{2.1}$ is hazardous (H_6) if it is not provided when the frequency is *low*, the generator, the generator is *working at highest rate*, and the Load 2 is consuming at *normal rate*.	H_6

In the next sections, we use Cassandra to identify safety-critical threat scenarios and derive related attack paths.

8.2.1 Safe behavioural model

We have formalised the behavioural model of the AC micro-grid using Network Timed Automata (NTA) using a full-knowledge approach (Section 4.2). For simplicity, we chose not to model aspects related to the synchronisation with the main grid; therefore, our behavioural model does not include information to synchronise to the main grid or the GNSS module. The

resulting behavioural model of the micro-grid is the parallel composition of 9 timed automata (TA), one for each component of the control structure (we have deployed 1 generator and 2 Battery Energy Systems (BES)). An additional timed automaton models the behaviour of the channel over which control actions and feedback are exchanged. In this section, we describe the automata modelling the behaviours of the MG controller, the communication channel, and one observer.

In Figure 8.5, we show the behavioural model of the AC MG Controller in a system with the number of BES (NB), set to 2. The controller checks that the frequency on the AC bus is

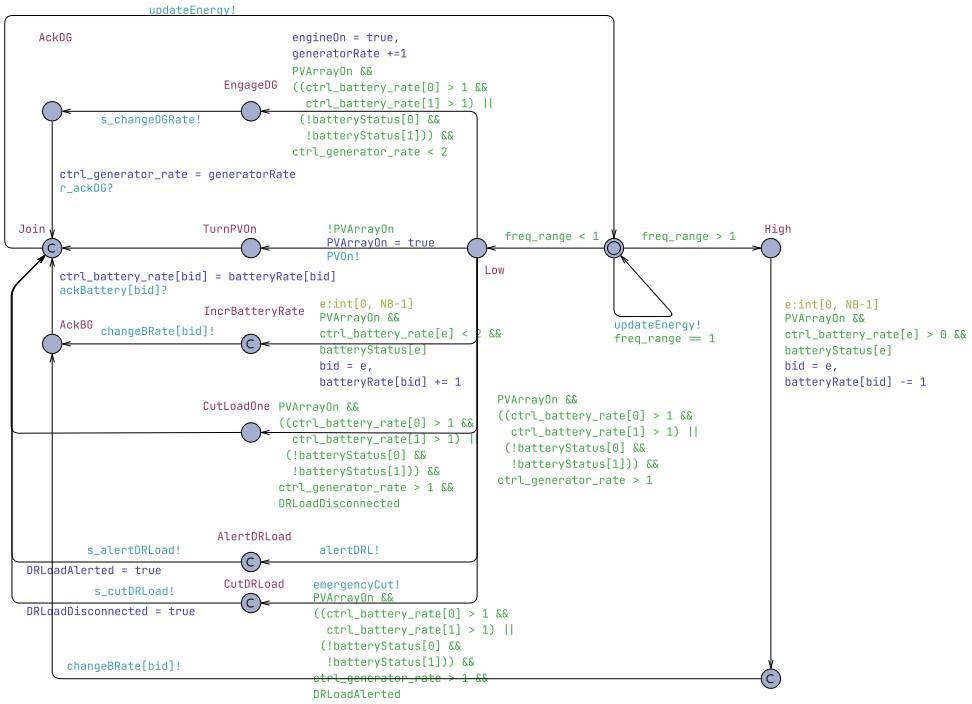


Figure 8.5: Behaviour of AC MG Controller

within normal parameters. If a low-frequency interrupt is detected, it activates, in order, the photovoltaic array (if it was offline), the batteries at an increasing rate and, ultimately, the generator also at an increasing rate. If the frequency remains low, the controller sends an alert to Load 2 through the master PLC and, eventually cuts it out. On the other hand, if the frequency is too high, the controller reduces the rate of the batteries incrementally. In this model, we make the ideal assumption that the generator starts contributing to the power balance as soon as it is engaged while, in reality, there can a delay before it reaches full generation capacity. Note that for simplicity, the controller does not implement the behaviour of decreasing/switching off

the generator when the frequency is too high. Nevertheless, the state machine representing the generator exposes the interface to switch off/on and increase/decrease the rate and we will see how this can be exploited by an adversary.

In Figure 8.6 we show the timed automaton that implements the channel over which messages are exchanged between controllers. Previously (Chapter 4), we have mentioned how an explicit

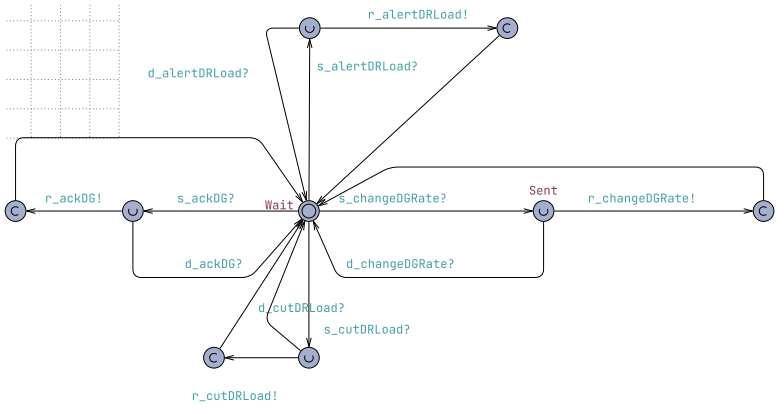


Figure 8.6: Behaviour of channel in the AC MG

implementation of the channel is essential to decouple the *send* and *receive* actions in an NTA. Such decoupling allows us to simulate an attacker able to perform Denial of Service and Man-in-The-Middle attacks on exchanged messages. In normal conditions, the channel acts as a relay and sends a synchronisation signal to the receiver, signalling to copy the value from the shared buffer. In adversarial conditions, between the send and the receive, the attacker can *disrupt* the message (channel does not notify receiver) or can change the value of the message before the receiver is notified.

Finally, we use an observer to monitor unsafe control actions of the type *not provided* or *provided too early/too late* (Figure 8.7).

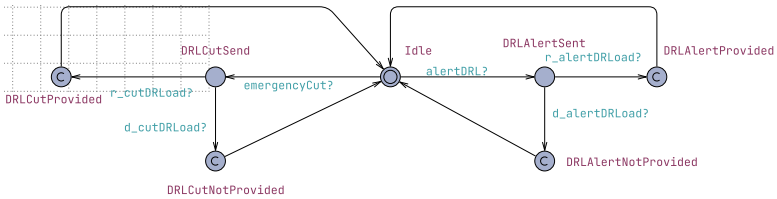


Figure 8.7: Behaviour of monitor for AC MG

Verification We use model checking to verify that, in non-adversarial conditions, the behaviour of the system is safe. In particular, we start from the unsafe control actions identified in 8.2 and leverage the mapping established earlier (Chapter 4) to identify, in the behavioural model the states where they are applied. Thus we use our toolchain to verify the reachability of the unsafe states. As an example, we define the following three reachability statements to verify UCA_1, UCA_2, and UCA_4:

UCA_1 - $\phi_1 := E <> MGController.CutLoadOne$

Explanation: Is there a configuration where the MGController requests the emergency shutdown of Load 1?

UCA_2 - $\phi_2 := E <> Monitor.DRLoadCutNP$

Explanation: Is there a configuration where the demand/response load (Load 2) should be shut down and the action is not provided?

UCA_4 - $\phi_3 := E <> Monitor.DRLoadANP$

Explanation: Is there a situation where the demand/response load (Load 2) should be alerted and the action is not provided?

In non-adversarial conditions, we expect the verification of ϕ_1, ϕ_2 and ϕ_3 to not be *not reachable*.

8.2.2 Attacker synthesis

We identify the STRIDE threats to the safe control structure. We define the system's trust boundaries and reason about how STRIDE threats can affect individual control actions and feedback signals. In particular, for the use case, we consider *Spoofing*, *Tampering*, *Denial of Service* and *Information Disclosure* threats against control actions and feedback but do not consider *Repudiation* and *Elevation of Privileges* as they do not *directly* impact safety. In our micro-grid example, we consider Spoofing, Tampering, Denial of Service and Unauthorised Read threats against $CA_{1.2}$, $CA_{5.1}$ and $CA_{11.3}$, and Spoofing of $CA_{12.2}$. The full list of the

threats considered is shown in Table 8.5. We use these threats to construct a state machine

Table 8.5: Threats MG

#	Control Structure	Type	Value
t_1	$CA_{1.2}$: Cut L2	Spoof	$[0-1]$
t_2	$CA_{1.2}$: Cut L2	DoS	-
t_3	$CA_{1.2}$: Cut L2	Read	-
t_4	$CA_{1.2}$: Cut L2	Tamper	-
t_5	$CA_{11.3}$: SetDGRate	Spoof	$[0-2]$
t_6	$CA_{11.3}$: SetDGRate	DoS	-
t_7	$CA_{11.3}$: SetDGRate	Read	-
t_8	$CA_{11.3}$: SetDGRate	Tamper	-
t_9	$CA_{5.1}$: Alert L2	Spoof	$[0-1]$
t_{10}	$CA_{5.1}$: Alert L2	DoS	-
t_{11}	$CA_{5.1}$: Alert L2	Read	-
t_{12}	$CA_{5.1}$: Alert L2	Tamper	-
t_{13}	$CA_{12.2}$: ackDG	Spoof	-

that synthesises a behavioural model of the attacker. In our example, this state machine is shown in Figure 8.8.

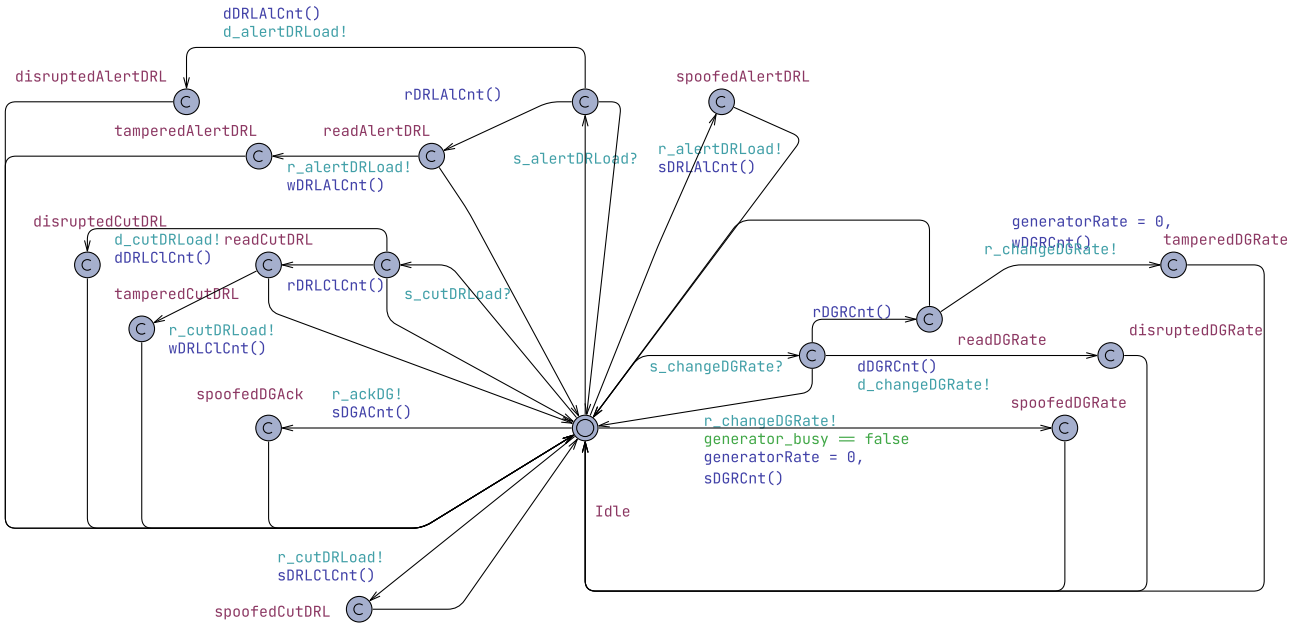


Figure 8.8: Behaviour of attacker

8.3 Discovery & Enumeration of threat scenarios

We construct the *instrumented model* of the system as outlined earlier in Chapter 5. Formally verifying ϕ_1, ϕ_2 and ϕ_3 on the instrumented model allows us to explore all the possible combinations of threats and component behaviours, including their cascading effects. During the first step of Cassandra, i.e., identification and enumeration of threat scenarios, we uncover safety-critical attacks (Chapter 5). We report the outcome of the verification in Table 8.6 for

Table 8.6: Threat scenarios found.

Hazard	Spec	T	TF	Attacks	CT
H_1	ϕ_1	3	3	$\{t_7, t_8\}$, $\{t_{13}\}$, $\{t_5\}$	$453.69s$
H_2	ϕ_2	11	3	$\{t_7, t_8, t_2\}$, $\{t_{13}, t_2\}$, $\{t_5, t_2\}$	$681.45s$
H_6	ϕ_3	3	3	$\{t_7, t_8, t_{10}\}$, $\{t_{13}, t_{10}\}$, $\{t_5, t_{10}\}$	$318.76s$

$\bar{t} = 300s$. For each ϕ, we also report the total number of scenarios found (T), the number of scenarios that are left after the elimination of redundancies (TF), and the cumulative time (CT) taken to verify each ϕ and those that our custom algorithm derived from it for enumeration. The attack enumeration finds 9 attacks potentially leading to the hazards H_1, H_2, H_6 and H_7. Among these, H_1, H_2 and H_6 potentially lead to the accident A_1. Scenarios $\{t_7, t_8\}$ and $\{t_5\}$, are straightforward, e.g. spoof or intercept communication between MG controller and generator. The adversary can use these threats to change the configuration of the generator to produce less energy than absorbed by the critical load, thus creating a situation of *low frequency* that leads the MG controller to cut the critical load. Similarly, in $\{t_{13}\}$ the MG controller is tricked into thinking that the generator is already operating at maximum rate.

The scenarios found from verifying ϕ_2 and ϕ_3 are more elaborated. By first targeting the generator, the adversary creates a frequency imbalance, which allows them to successfully trigger the emergency control action *Cut (Alert) Load 2*. This is due to the fact that we have assumed the generator can sustain, alone, the combined maximum demand of $Load_1$ and $Load_2$. By removing any assumption on the generator's production capacity we can reduce the scenarios to $\{t_2\}$ for ϕ_2 and $\{t_{10}\}$ for ϕ_3.

In Figure 8.9 we show the attack tree for hazard H_2. Whilst we consider H_2 to only be triggered

by UCA_2, it is possible to have more than one UCA concurring to cause the same hazard. From the verification of ϕ_2 we have identified *three* threat scenarios (TSc-2, TSc-2 and TSc-3) leading to UCA_2.

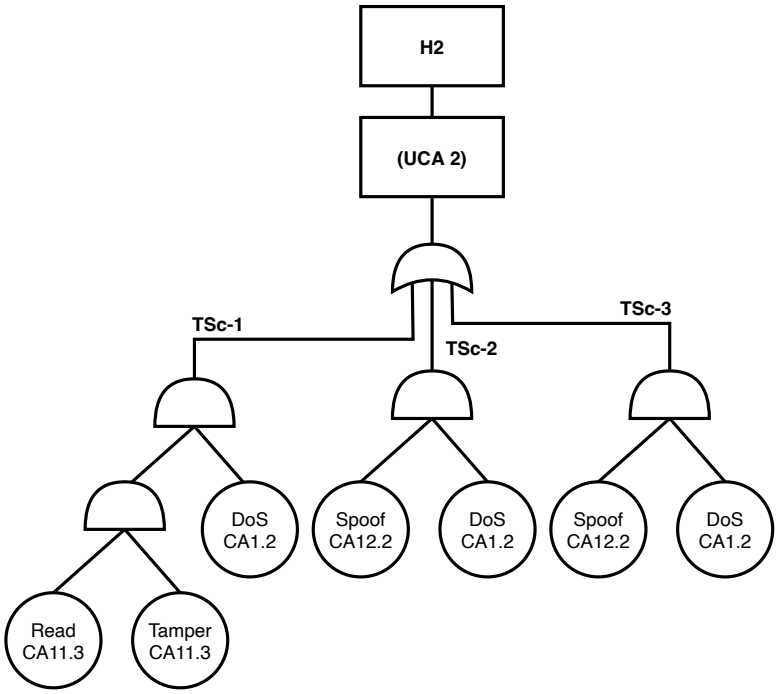


Figure 8.9: Threat scenarios tree

8.4 Analysis of safety-critical attack paths

The architecture of the micro-grid treated in this use-case is shown in Figure 8.10. This is a subset of the architecture of the EPIC testbed, with similar hardware, software and vulnerabilities. Specifically, the MG controller is implemented as a Programmable Logic Controller and it is located in the micro-grid control network. The SCADA workstation is a Windows host vulnerable to *EthernalBlue* (`CVE-2017-0144`), while the PLC controllers run a version of Dropbear SSH that is exposed to remote code execution (RCE) vulnerabilities (`CVE-2016-7406`, `CVE-2016-7407`, `CVE-2016-7408`, and `CVE-2016-7409`). PLCs also expose a network service that enables an attacker with access to the same network to upload and run code without authorisation (`CVE-2012-6068`). Microsoft Windows computers in the system also run an SMB server vulnerable to `CVE-2017-0267`, `CVE-2017-0268`, and `CVE-2017-0269` [AKZM20]. The latter allows the adversary to breach the confidentiality and the availability of messages.

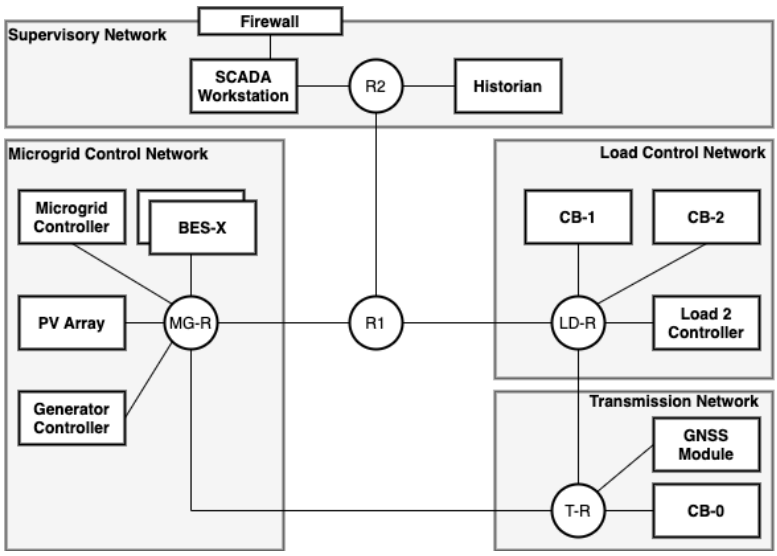


Figure 8.10: Network topology of micro-grid.

We placed the controller of Load 2 in the Load Control Network, along with the network-enabled circuit breakers for emergency detach of Load 2 (CB-2) and Load 1 (CB-1). Similarly, the Transmission Network contains the network-enabled circuit breaker for the connection to the main grid (CB-0) and the GNSS module for accurate time sampling. As in EPIC, the control networks are connected in a High-availability Seamless Redundancy (HSR) topology. Finally, a SCADA workstation and historian server monitor the operations from the Supervisory Network. The SCADA workstation is equipped with a second network for secure connection with other private networks and, as in [CHC+22] runs a Remote Desktop Protool (RDP) service to enable remote monitoring. For simplicity, in the STAMP model in Figure 8.3, we ignore eventual controls from the SCADA workstation.

8.4.1 Safety-critical attack paths

In the second step of Cassandra, we leverage the integration with the architectural model of the CPS to identify the privileges the attacker needs to acquire to execute a threat scenario. For simplicity, we consider two types of privileges: *network access* and *code execution*. At this stage, we assume that the architectural model of the CPS (Figure 8.10) is available. This model shows how components are connected (network topology) and provides high-level details on the implementation of system components such as manufacturer, model, software run, network

address and open ports. Usual traceability requirements adopted during the development of safety-critical systems explain the map between the architectural and information models.

We analyse the architecture of the system in conjunction with its control model and execute a mapping between the privileges (architecture) and threats (safety model). For the micro-grid scenario, we consider the subset of privileges in Table 8.7.

Table 8.7: Privileges AC MG architecture

#	**Privilege**
P_1	$execCode(mgCtrl, root)$
P_2	$execCode(pvArray, user)$
P_3	$execCode(pvArray, root)$
P_4	$execCode(scadaWorkstation, user)$
P_5	$execCode(scadaWorkstation, root)$
P_6	$execCode(historian, user)$
P_7	$execCode(historian, root)$
P_8	$netAccess(scadaWorkstation, \_, \_)$
P_9	$netAccess(mgCtrlNet, \_)$
P_{10}	$netAccess(LoadCtrlNet, \_, \_)$
P_{11}	$netAccess(TransmissionNet, \_, \_)$

To compute the safety set of H_2, i.e., the set of privileges that need to be protected to avoid H_2, we need to identify the privileges that enable the executions of the threats identified in the previous step. From Figure 8.9, we have that the threat scenarios combine, in different ways, threats t_2, t_5, t_7, t_8, and t_{13} (Table 8.5). From the architecture of the micro-grid (Figure 8.10) we build the following mapping between threats and the privileges required:

$$t_2 \leftarrow P_9 \vee P_{10} \vee P_1$$

$$t_5 \leftarrow P_9 \vee P_{10} \vee P_{11}$$

$$t_7 \leftarrow P_4 \vee P_5 \vee P_6 \vee P_7 \vee P_8 \vee P_9 \vee P_{10} \vee P_{11} \vee P_{12} \vee P_1$$

$$t_8 \leftarrow P_1 \vee P_9 \vee P_{10}$$

$$t_{13} \leftarrow P_9 \vee P_{10} \vee P_{11}$$

For example, t_2 (Denial of Service of Cut Load) can be performed by acquiring network access to the MG Control Network, network access to the Load Control Network or (root) access to the micro-grid PLC (e.g., modified firmware).

Paths generation We rely on MulVAL to generate the attack graph of the system. We assume that a security assessment has been performed to evaluate the vulnerabilities and weaknesses of the different components. The resulting graph includes safety-critical attack paths, in the whole architecture, leading to the privileges in the safety set. An analysis of the discovered paths allows us to determine architectural-level security requirements such as network allocation, patching of specific vulnerabilities and host isolation, capable of preventing the threat scenarios.

We have discussed the vulnerabilities that affect the components of our system. We have defined a trust boundary above the SCADA workstation and consider connections coming from the external private network as potentially dangerous. Therefore, the SCADA network is at the perimeter of our analysis and constitutes the initial target of a potential safety-critical attack. In

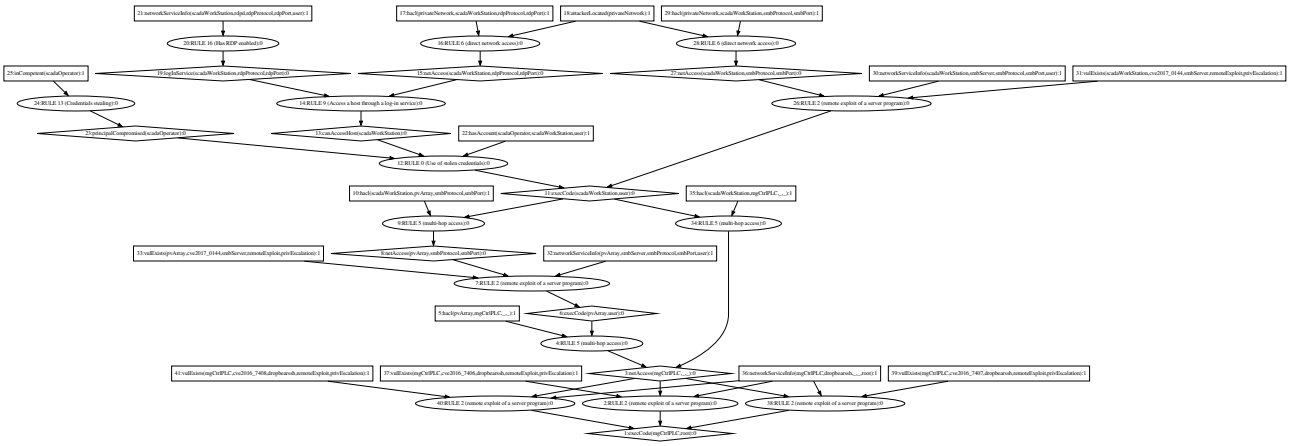


Figure 8.11: Attack graph to root privileges in micro-grid controller

Figure 8.11 we show the generated attack paths from the perimeter to privilege P_2 (privileged code execution on the micro-grid controller), which we represent using the MulVAL syntax [OGA+05]. In the case of the AC micro-grid, the attack graph in Figure 8.11 shows that an attacker can access the SCADA workstation in two ways: re-use stolen RDP credentials - if they have any - or exploit EthernaBlue. Once they have reached the SCADA workstation, they have

obtained privilege P_8 and can already execute the threat $t7$. From there the attacker has direct access to the micro-grid PLC, which allows firmware updates without authentication (CVE-2012-6068). This gives the attacker privileged code execution on the device (P_2, Table 8.7). When at the PLC, the attacker can also obtain a direct interface to the control network. This gives them privileges P_9, P_{10} and P_{11} and, therefore, the ability to execute a safety-critical attack (Figure 8.9). Note that MulVAL also finds an alternative path to the micro-grid controller where the attacker exploits the photovoltaic array controller before attacking the PLC. This attack path is less efficient than the other already found and does not provide any advantage.

8.5 Discussion

The application of Cassandra to the AC microgrid enables us to discover several threat scenarios that affect the system design. Starting from there, we can build assurance case fragments using an attack tree representation, that explains which combinations of threats, interleaved with system behaviour, pose a risk for the CPS and how they have been mitigated. We combined the discovered threat scenarios with the architectural model of the system to identify the set of privileges that are critical with respect to safety. Finally, we produced an attack graph from the system to visualise the attack paths that can lead to the application of the threat scenario identified.

For comparison, we have performed a manual application of our integrated safety and security analysis on the broader energy management system (EMS) (Figure 8.11). Results were reported in [CHC+22]. For the micro-grid section, the automatic application of Cassandra brought the discovery of interesting threat scenarios, given our assumptions on the attacker model. For example, the manipulation of the generator followed by a denial of service of alert messages can lead to dangerous phase unbalance even if the generator was designed to sustain large loads (i.e., critical and non-critical combined) in principle. The analysis of attack paths in Cassandra identified the same safety-critical attack paths that were manually identified in [CHC+22]

In Figure 8.12, we show a summary of the safety-critical attack paths identified for the broader EMS. Labels for control actions and feedback refer to the STAMP model in Figure 8.3). [CHC+22]. We observe that the safety-critical attack paths to the micro-grid PLC discov-

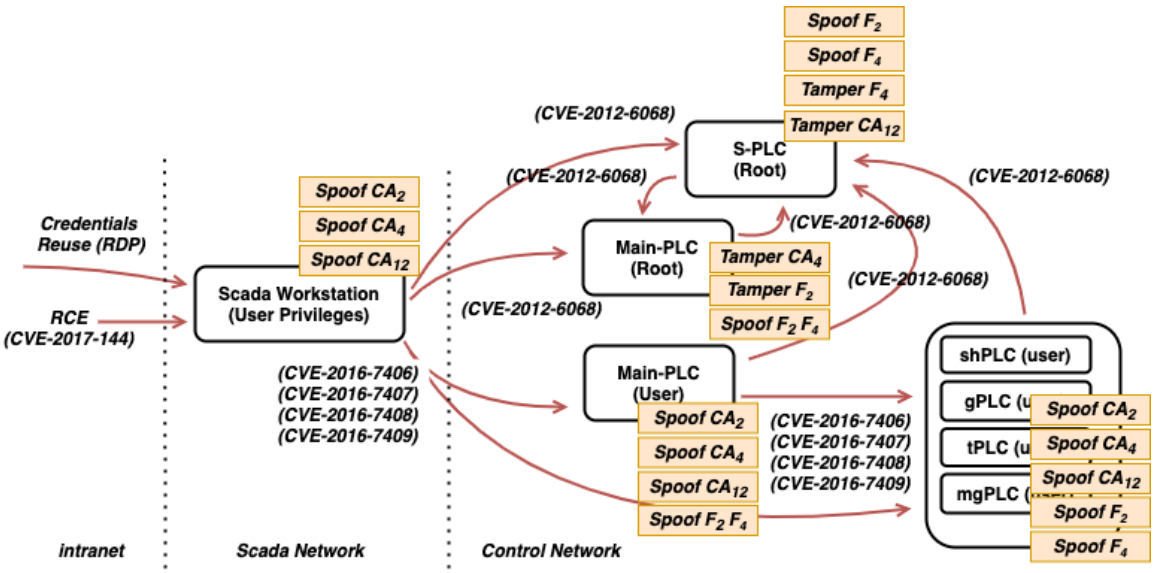


Figure 8.12: Attack graph derived via manual analysis STPA-Sec

ered automatically by Cassandra (Figure 8.11) are also among the attack paths shown in Figure 8.12.

By analysing the graph, the first security requirement that we can produce as a result of this analysis is the installation of a network diode between the SCADA workstation and the control networks. This would provide a physical layer of protection to the environment without interfering with the operating requirements (if the SCADA workstation has only monitoring purposes). A second security requirement consists of mitigating $CVE-2017-144$ (EthernaBlue) on the SCADA workstation. Such action, together with stricter policies on operator accesses as an additional requirement, mitigates the threats in the threat scenario tree (Figure 8.9). An additional - and perhaps obvious - security requirement would be the mitigation of $CVE-2012-6068$. Unfortunately, this vulnerability affects a PLC, and it might not be possible to correct it due to safety constraints, e.g., if the mitigation is not approved by certification authorities.

8.6 Summary & Conclusions

In this chapter, we have applied Cassandra to analyse the impact of STRIDE threats targeting a power system. The objective of the analysis was the identification of safety-critical attack paths. The functional and architectural models used for this use case are based on those of the Electric Power and Intelligent Control (EPIC) test-bed [AKM18]. We have identified a number of potential safety-critical threat scenarios and uncovered the safety-critical attack paths leading to their exploitation. Our results help security analysts in determining weaknesses and vulnerabilities that should be addressed first to preserve safety. Results obtained for the use case presented in this chapter were partially published in [CHC+22].

For the micro-grid section, the novel approach introduced by our methodology led to the discovery of interesting threat scenarios. For example, a manipulation of the generator followed by a denial of service of alert messages can trigger dangerous phase unbalance when the micro-grid is operating in islanded mode. In particular, the hazard can also be caused in the case of generators appositely designed to sustain large loads (i.e., critical and non-critical combined).

In the next chapter, we show the application of Cassandra to analyse the safe operation of a Flight Management System. We will elaborate on the automatic enumeration of threat scenarios under a *partial knowledge* behavioural model. In addition, the use case presented in the next section proposes an application of Bayesian inference on uncovered attack paths aimed at evaluating the efficacy of different security control policies.

Chapter 9

Analysing the impact of attacks on the safety of flight management systems: a case study

With the increase of connectivity in modern aircraft, the attack surface available to adversaries also grows. Complex attacks combine the exploitation of vulnerabilities with the malicious use of legitimate functionality to induce cascading effects and alter system behaviour. In digital avionics systems, without appropriate safeguards, such attacks can lead to the violation of safety requirements.

The Flight Management System (FMS) is of utmost importance among the various systems integrated into modern aircraft, primarily due to its pivotal role in significantly reducing the crew workload associated with flight planning, navigation, and guidance. With the rapid advances in technology, FMS has become an indispensable component of both commercial and general aviation aircraft, significantly improving operational efficiency and flight safety [MSJ13]. Its functionality relies on seamless integration with numerous other systems and components within the aircraft, thereby increasing the complexity involved in conducting a comprehensive analysis of all potential paths that could lead to safety or security violations. Performing a thorough analysis of aircraft systems such as the FMS during the development phase of the aircraft fa-

cilitates the identification of potential architectural improvements and the implementation of necessary mitigations to maintain required levels of safety and security and, thus, development cost.

In this chapter, we apply Cassandra to analyse the safe operation of a Flight Management System (FMS) when the adversary tries to access safety-critical information by compromising the device used as the Electronic Flight Bag (EFB). We evaluate the probability of successful attacks in three different scenarios: EFB available on a pilot-owned device, EFB available on an airline-controlled device with limited connectivity, and EFB available on aircraft only. While the outcome of Cassandra may be intuitive in this case, the example allows us to show how Cassandra improves automation and integration of safety and security analysis for modern avionic architectures, where complexity hinders intuition and manual analysis is laborious and error-prone.

The avionics models analysed in this chapter were developed in collaboration with Airbus. The content of this chapter was presented at the AIAA/IEEE $42-nd$ Digital Avionics Systems Conference (DASC) [CSP+23] and was awarded the best paper award in the *Cybersecurity I* session.

9.1 Preliminaries

The Flight Management System (FMS) is part of the auto-flight system in conjunction with flight guidance and flight envelope management. The FMS is responsible for managing the navigation, data-link, lateral and vertical functions of an aircraft. It carries out safety-critical functions such as following routes and fuel management. It exchanges information with other parts of aircraft avionics such as Navigation Systems (NS), Flight Control Systems (FCS), as well as other sensors and actuators to carry out its functions. For simplicity, in this use case, we primarily focus on the safety of the operations of the *follow route* function, but our approach can be easily applied to the safety of other functions. In Figure 9.1, we show the network topology considered. It is divided into three distinct domains, each of them with its own characteristics

and security levels [OOA06]. The aircraft control domain (ACD), depicted in blue, is used by avionics components such as FMS, NS, sensors, etc. to exchange information. This network is safety-critical and contains redundant components as well as redundant buses. For simplicity, this redundancy is not shown explicitly in Figure 9.1. The airline service domain (ASD) is depicted on a green background, it contains the management, non-safety-critical, information systems used to communicate with airline services through wireless networks. The ASD also contains the Data Loader System (DLS), used by the pilots to upload flight plans from the Electronic Flight Bag (EFB). The latter is a Commercial off-the-shelf (COTS) portable device where the flight plans are stored and which also allows their consultations during the distinct phases of the flight [HCK+15]. Finally, components for passenger *infotainment* are located in the passenger information and entertainment services domain (PIES).

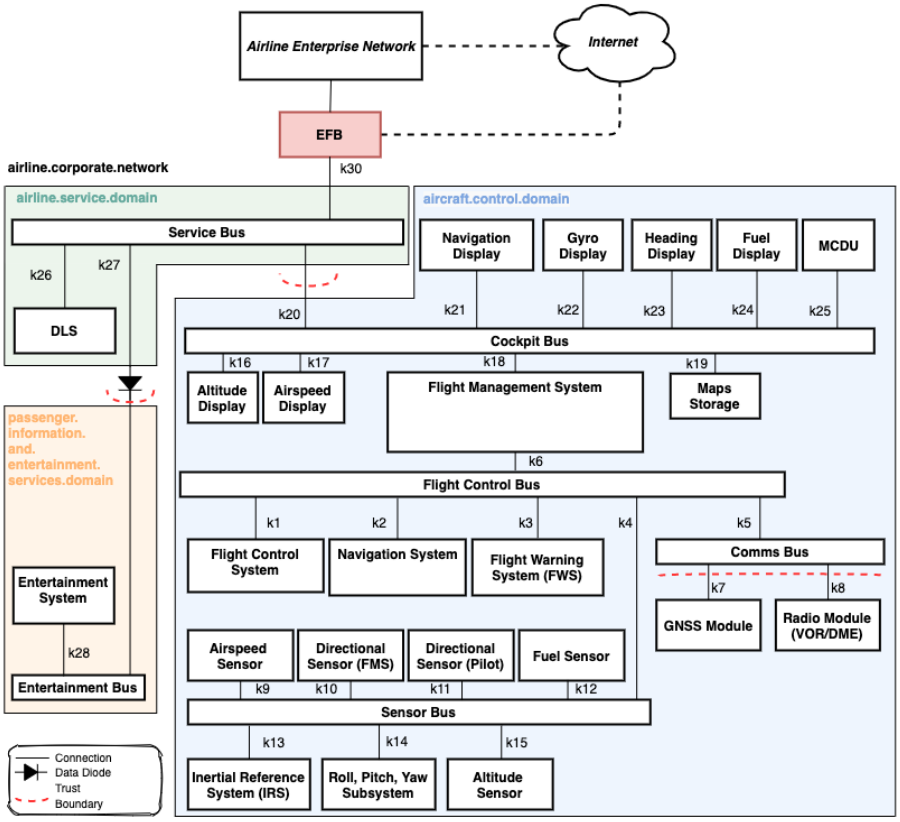


Figure 9.1: Aircraft Computer Network

In this use case, we analyse how the risk of safety-critical attacks changes, given the following three different levels of security controls on the Electronic Flight Bags (EFB) enforced by the operator:

(Scenario 1) EFB is provided by the airline and has access to the internet (e.g., available under a bring-your-own-device policy).

(Scenario 2) EFB is provided by the airline and cannot access the internet, but is enabled with network access towards airline-owned enterprise network.

(Scenario 3) EFB does not have access to the internet.

In particular, we assume that the EFB device is an iPad and that it is vulnerable to `CVE-2022-22620` (Remote Code Execution, RCE) and `CVE-2022-22675` (Local Privilege Escalation). In fact, the Federal Aviation Administration (FAA) approved iPad to be used as EFB in 2011 [CNN11]. In our scenario, we also assume that the map server hosts a software component vulnerable to `CVE-2017-0144` (RCE). Finally, we assume the presence of the following RCE vulnerabilities `Vuln-X`, `Vuln-Y`, and `Vuln-Z`, which affect the EFB proprietary software, the DLS, and the aircraft map storage, respectively.

9.1.1 Safety Model

Earlier in this thesis we have mentioned the *three* stages composing Cassandra (Figure 3.1). The first stage uses the safety model of the system to produce the *threat scenarios tree* which explains how high-level threats can lead to catastrophic consequences (❶). We leverage STAMP to model the safety of the system and STPA and STPA-Sec to discover the dynamics leading to accidents [Lev16, YP17] which is particularly relevant in this context. In fact, STPA and STPA-Sec are well-accepted methods for safety analysis and are applied across many different sectors [PCKDG22]. Furthermore, we use STRIDE [Sho08] as the threat modelling approach within STPA-Sec. The derivation of the interleaving sets of malicious actions and system behaviour is achieved through model checking. In doing so, we are able to also consider critical aspects relative to the time and context in which threats occur (e.g., flight phase, autopilot being enabled). In the second stage, we leverage the integration with the system architecture to identify the safety-critical attack paths. This operation is critical in identifying the privileges that the attacker needs to obtain to execute safety-critical threat scenarios and cause harm

(❷). Finally, in this use case we employ exact Bayesian inference on the attack graph to quantitatively compute the probability of success for the attack paths identified. We evaluate the effectiveness of security measures in reducing the probability of success of safety-critical attacks (❸).

9.1.2 Definition of the scope of the analysis

The first step of STPA/STPA-Sec consists of defining the scope of the analysis. At this stage, we identify *Losses*, *Accidents* and *Hazards*, as defined in [Lev16], that are relevant to the FMS. In particular, we consider the following subset of losses from [Spe20] the Acceptable Means of Compliance (AMC) for Large Aeroplanes[1] [Spe20]; Multiple Fatalities (L_1), Hull Loss (L_2), and Physical discomfort or a significant increase in workload of Flight Crew (L_3). According to [Spe20], L_1 and L_2 are classified as *Catastrophic* (allowable probability $< 10^{-9}$) while L_3 is classified as *Major* (allowable probability $< 10^{-5}$). For this use case, we consider the following reduced subset of events leading to L_1, L_2 and L_3: (A_1) Collision (from Loss of Separation), (A_2), Controlled Flight into Terrain (CFiT), (A_3) Unreliable Avionics Behaviour. We have

Table 9.1: Table of Hazards for Flight Management System

#	**Hazard**	**Accidents**
H_1	Aircraft violates minimum separation standards in flight	A_1, A_2
H_2	Aircraft does not maintain safe distance from terrain and other obstacles	A_2
H_3	Unable to follow route	A_3

identified in Table 9.1 the subset of hazards of concern for the *follow route* functionality of the FMS. If the FMS receives an unreliable position from the Navigation System (NS), it will compute commands that can lead to a loss of separation and, eventually, a collision if the aircraft is on autopilot (H_1). Conversely, commands issued by FMS based on an unreliable position or map data can lead to CFiT (H_2) when the aircraft is descending on autopilot. Finally, unreliable feedback from avionics or unreliable data on the databases can lead to errors in the computation of the flight path and an increase in the workload of the Flight Crew that is forced to compute corrections by alternate means (H_3).

[1] AMC 25.1309 – System Design and Analysis

9.1.3 Derivation of control structure

The second step of STPA/STPA-Sec is the derivation of the safe control structure which captures the dynamics of the interactions between system components [Lev16].

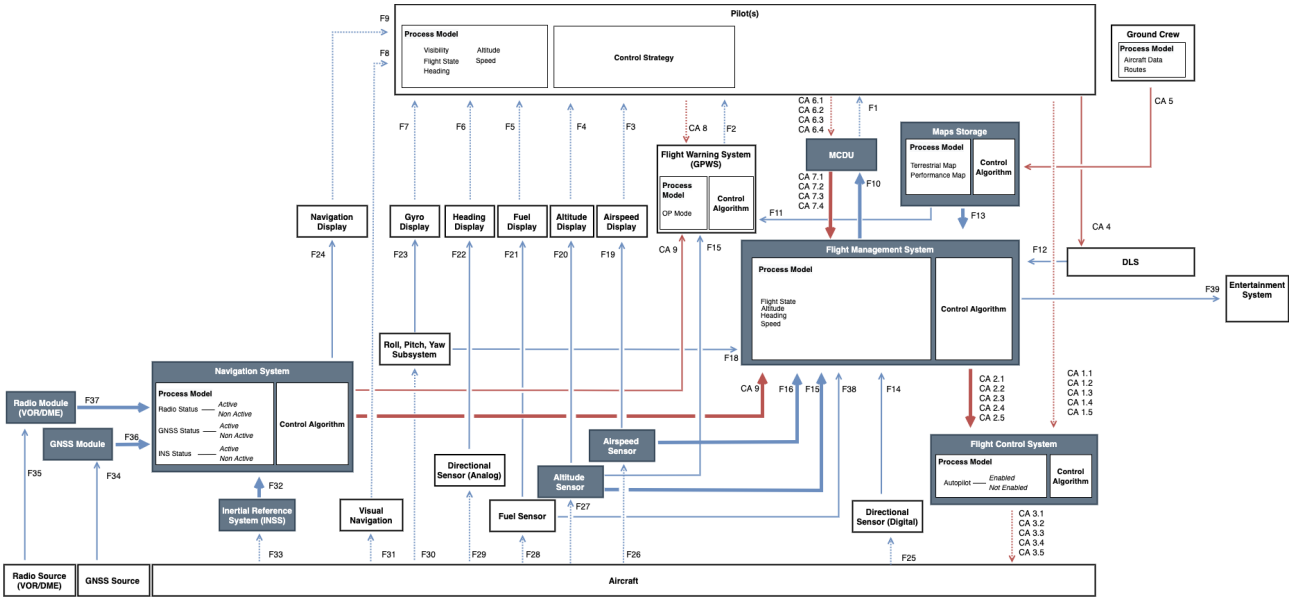


Figure 9.2: Safe Control Structure of Avionics - Highlighted elements are analysed in the next Section using Cassandra.

The SCS is a tuple (C, D, K) where C denotes the set of components (controllers, sensors, actuators and physical processes), D the set of *control actions* and *feedback*, and K a set of *functional channels* [CL20]. Within each controller, the variables of the *process model* retain the representation of the physical process and are used by the *control algorithm*. The SCS for our use case is shown in Figure 9.2. For simplicity, only a subset of components, control actions and feedback are analysed using Cassandra in the next sections. These have been highlighted in Figure 9.2 and are summarised in Table 9.2.

In this paper, we focus on the safe operation of the FMS which issues control actions Change Altitude ($CA_{2.1}$), Change Yaw ($CA_{2.2}$), Change Pitch ($CA_{2.3}$), Change Roll ($CA_{2.4}$), and Change Thrust ($CA_{2.5}$). We use CA_2 to refer to the entire output of the FMS, hence control actions $CA_{2.1}$ to $CA_{2.5}$. These actions are computed on the basis of the current state of the FMS from the input received from the Navigation System (NS) and other sensors such as airspeed, fuel, heading, altitude, etc. The NS computes CA_9 on the basis of the data received from the Inertial

Table 9.2: Control Actions and Feedback in Figure 9.2.

CA/FB	Description (Source Component)	CA/FB	Description (Source Component)
CA 1.1	Change Altitude (Climb/Descend) (Pilot)	CA 1.2	Change Yaw (Pilot)
CA 1.3	Change Pitch (Pilot)	CA 1.4	Change Roll (Pilot)
CA 1.5	Change Thrust (Decrease) (Pilot)	CA 2.1	Change Altitude (Climb/Descend) (FMS)
CA 2.2	Change Yaw (FMS)	CA 2.3	Change Pitch (FMS)
CA 2.4	Change Roll (FMS)	CA 2.5	Change Thrust (FMS)
CA 3.1	Change Altitude (Climb/Descend) (FCS)	CA 3.2	Change Yaw (FCS)
CA 3.3	Change Pitch (FCS)	CA 3.4	Change Roll (FCS)
CA 3.5	Change Thrust (FCS)	CA 6.1	Set Route (Pilot)
CA 6.2	Change Route (Pilot)	CA 6.3	Start Follow (Pilot)
CA 6.4	Stop Follow (Pilot)	CA 8	Toggle Alarm (Pilot)
CA 7.1	Set Route (MCDU)	CA 7.2	Change Route (MCDU)
CA 7.3	Start Follow (MCDU)	CA 7.4	Stop Follow (MCDU)
CA 5	Upload Data (Ground Crew)	CA 4	Upload Flight Plan (Pilot)
CA 15	Radio Signal (Radio Source)	CA 16	GNSS Position (GNSS Module)
CA 17	Radio Position (Radio Module)	CA 14	GNSS Signal (GNSS Source)
CA 9	Position (Navigation System)		
F1	Flight Data (MCDU)	F2	Alarm (FWS)
F3	Airspeed (Airspeed Display)	F4	Altitude (Altitude Display)
F5	Fuel (Fuel Display)	F6	Heading (Heading Display)
F7	Roll, Pitch, Yaw (Gyro Display)	F8	VNav (Visual Navigation)
F9	Position (Navigation Display)	F10	Flight Data (FMS)
F11	Alarm (FMS)	F12	Uploaded Data (DLS)
F13	Stored Maps (Map Storage)	F14	Heading (Dir Sensor FMS)
F15	Altitude (Altitude Sensor)	F16	Airspeed (Airspeed Sensor)
F17	Position (Navigation System)	F18	Roll, Pitch, Yaw (RPY Subsystem)
F19	Airspeed (Airspeed Sensor)	F20	Altitude (Altitude Sensor)
F21	Fuel Level (Fuel Sensor)	F22	Heading (Dir Sensor Pilot)
F23	Roll, Pitch, Yaw (RPY Subsystem)	F24	Position (Navigation System)
F25	IMU-1 Heading (Raw)	F26	Airspeed (Raw)
F27	DME Input (Raw)	F28	Fuel Level (Raw)
F29	IMU-2 Heading (Raw)	F30	Roll, Pitch, Yaw (Raw)
F31	Visual Navigation Input (Raw)	F32	Acceleration Forces (Raw)
F33	Position (IRS)	F34	CA 14
F35	CA 15	F36	CA 16
F37	CA 17	F38	Fuel Level (Fuel Sensor)

Reference System (INSS) (F_{32}), the Global Navigation Satellite System (GNSS) (F_{36}), and the radio (F_{37}) using the criteria outlined later in section 9.1.6.

9.1.4 Identification of Unsafe Control Actions

The third step of STPA/STPA-Sec is the identification of Unsafe Control Actions (UCA). For example, the *climb* command issued by the FMS is unsafe when its application places the aircraft in an unsafe situation, e.g., loss of separation, steep climb, etc. For simplicity, we report here only the following unsafe applications related to the control action *change altitude*,

the output of the FMS ($CA_{2.1}$):

UCA_1 : $CA_{2.1}$ (Change altitude) is applied (**climb**) when not required while **autopilot is active** $\leftarrow H_1, H_3$.

UCA_2 : $CA_{2.1}$ (Change altitude) is applied (**descend**) when not required while **autopilot is active** $\leftarrow H_2, H_3$

UCA_3 : $CA_{2.1}$ (Change altitude) is applied (**climb** or **descend**) when not required while **autopilot is not active** H_3

UCA_4 : $CA_{2.1}$ (Change altitude) is not applied (**climb**) when required while **autopilot is active** $\leftarrow H_2, H_3$.

UCA_5 : $CA_{2.1}$ (Change altitude) is not applied (**descend**) when required while **autopilot is active** $\leftarrow H_1, H_3$.

UCA_6 : $CA_{2.1}$ (Change altitude) is not applied (**climb** or **descend**) when required while **autopilot is not active** $\leftarrow H_3$.

UCA_7 : $CA_{2.1}$ (Change altitude) is applied (**climb** or **descend**) too early or too late while **autopilot is active** $\leftarrow H_1, H_2, H_3$.

UCA_8 : $CA_{2.1}$ (Change altitude) is applied (**climb** or **descend**) for too long or too short while **autopilot is active** $\leftarrow H_1, H_2, H_3$.

9.1.5 Identification of Threat Scenarios (STPA-Sec)

In the last step of STPA/STPA-Sec, we look for the causes behind the application of unsafe control actions in the whole control structure. In particular, STPA-Sec looks into threats as a possible cause, whereas traditional STPA focuses on faults. The identification of causal scenarios is traditionally performed manually, mainly relying on expert knowledge. This process is complex, time-consuming and error-prone as it requires the analyst to intersect the specific behaviour of individual system components with a high-level system view on how control actions and feedback propagate. In addition to this, STPA-Sec does not provide clear tools to define

the scope of the security analysis (e.g., trust boundaries) nor provides guidelines on the nature of threats to consider in this fourth step. We employ STRIDE [KMLS17] to identify threats that are applicable to individual elements of the control structure. Thus, proceed with the enumeration of threat scenarios *threat scenarios*.

9.1.6 Safe behavioural model

The *safe behavioural model* (SBM) of the CPS models the propagation of cascading effects across the safe control structure using *partial knowledge* approach (Chapter 4). In this use-case, we encode input/output relationships of system components in the *safe behavioural model* (SBM) of the CPS using a partial knowledge approach (Section 4.3).

For simplicity, the network of timed automata (NTA) encoding the SBM includes only a subset of components and information flows in the safe control structure (those shown with a darker background in Figure 9.2). In particular, the NTA representing our use case includes 17 timed automata, including one for each of the components *altitude sensor*, *inertial reference system* (INSS), *flight management system* (FMS), *map storage*, *multi-function control and display unit* (MCDU), *flight control system* (FCS), and *navigation system*. We model the changes of states in the information flows with the following TAs *MCDU input*, *MCDU output*, *FMS pitch output*[2], *FMS Steering Output*[3], *GNSS position*, *radio position*, *inertial position*, *position (Navigation System Output)*, *map storage data*, and *altitude*. Timed automata in the NTA are synchronised with each other over *synchronisation channels*. In this Section, we outline some of the automata in our NTA as an example and comment on their structure.

The states of the timed automata represent different failure modes for each component or information flow (e.g., loss of availability/reliability of a component or information flow, alternate mode of functioning of a component, etc.). Transitions across locations express dependencies between the failure modes of elements in the SCS, e.g., how a loss of reliability/availability on the input of a controller affects the state of the controller, and/or how, an eventual change of

[2]For simplicity, this output aggregates changes of *pitch*, *yaw* and *roll*
[3]For simplicity, this output aggregates changes of *thrust* and *altitude*

state of the latter affects its output (section 4.3). For each component and information flow, we consider two basic failure modes: *unreliable* and *unavailable*. For complex components, such as the controller, we specify a broader set of failure modes which we assume to be provided by the design team.

In Figure 9.3 we show the timed automaton that represents the *behaviour* of a simplified flight control system (FCS). For simplicity, the FCS accepts two inputs from the flight management

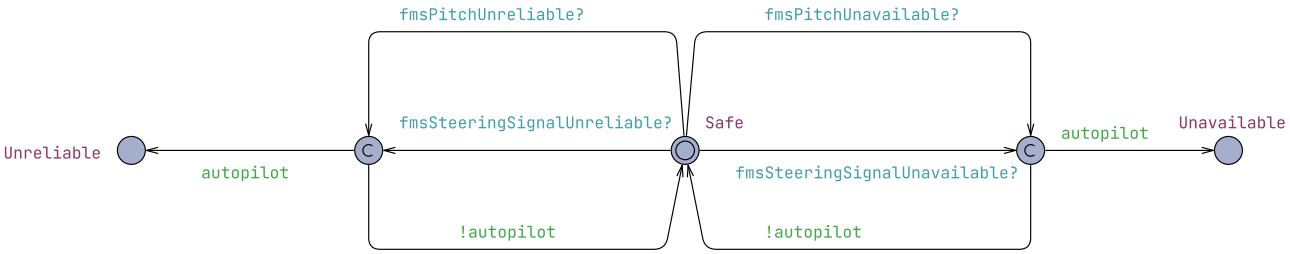


Figure 9.3: Flight Control System (TA)

system. *FMS Steering*, aggregates requests of changes of thrust, change of altitude and change of heading, while *FMS Pitch* aggregates pitch, yaw and roll. The FCS starts in a safe location where both its inputs are available and reliable, and so is its output. If one of the inputs becomes *unavailable* or *unreliable*, the output becomes *unavailable* or *unreliable* respectively - when the autopilot is engaged.

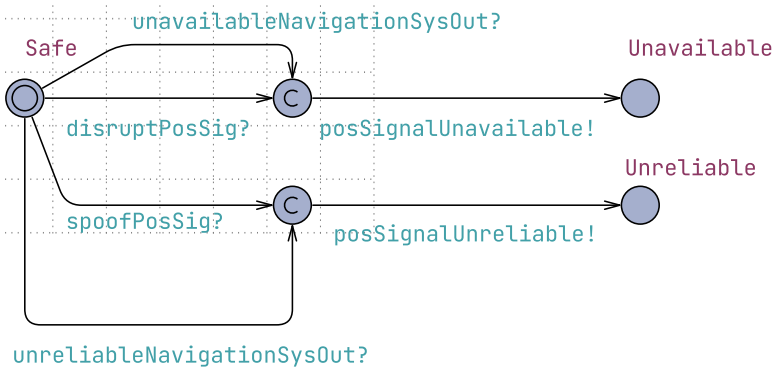


Figure 9.4: Position Signal (TA)

In Figure 9.4, we show the timed automaton that encodes the states of the position signal (information flow), which is an output of the NS and an input of the FMS. The position signal starts in a *Safe* location and becomes *unreliable* (or *unavailable*) as a result of an attack:

1. against the navigation system (NS) (threat to source) or

2. against the position itself (threat to information flow data).

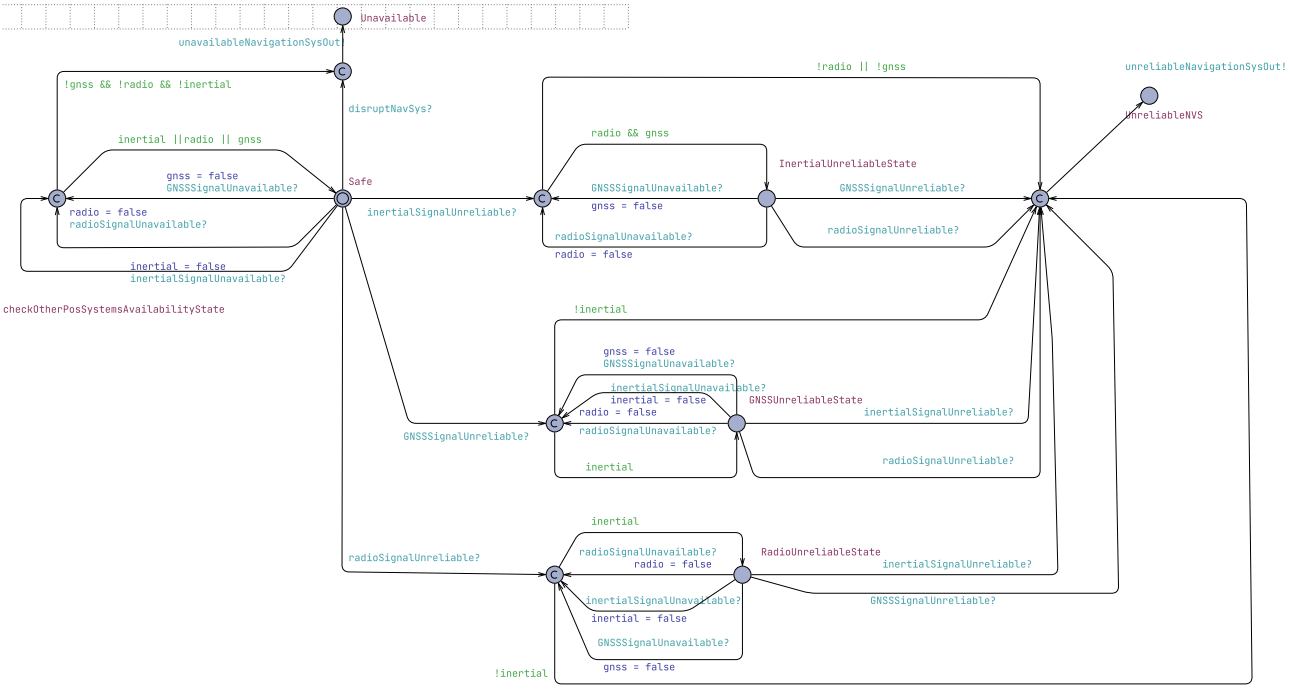


Figure 9.5: Navigation System (TA)

In Figure 9.5, we show the TA that encodes the behavioural model of the navigation system (NS) - a behaviour more complex than that of the FCS shown earlier. The NS uses a weighted voting algorithm of its three inputs to estimate the current position of the aircraft (output). The first input is provided by the GNSS service, which is considered the least reliable and vulnerable to spoofing attacks. The second input comes from the radio receiver. This is considered more precise and reliable than GNSS although its availability is not guaranteed for the entirety of the route. Although more difficult to attack than GNSS, the radio signal comes from an external source and cannot be considered secure. Finally, the third input comes from the inertial reference system (INSS) and is considered the most secure as it does not rely on external communications; however, its precision is low and degrades over time. The first output, *FMS Steering*, aggregates change of thrust, change of altitude and change of heading. Similarly, the second output, *FMS Pitch* aggregates pitch, yaw and roll. We assume that the FMS starts in a safe location where both its outputs are available and reliable.

The NS starts from a *Safe* location where we assume that the inputs are all *available* and *reliable* and so is its output. The output is computed according to the following rationale:

1. When the inputs are all available, but provide different values, the NS trusts the INSS above all.

2. A degraded mode allows the NS to trust GNSS and Radio over the INSS when GNSS and Radio agree.

3. Additional degraded modes allow the system to operate when one or two inputs are not available. In this case, the order of trust is INSS, radio, and GNSS. For example, if the INSS is unavailable (e.g., due to a fault), radio is trusted over GNSS in case of disagreement.

4. If all three inputs are unavailable, then the NS becomes unavailable.

5. the output of the NS becomes unreliable when multiple adverse events occur (e.g., radio and INSS are unavailable and GNSS provides unreliable values, or GNSS and radio start providing unreliable values as a result of a maliciously induced fault).

9.1.7 Threat Model

To formally verify the reachability of unsafe states in adversarial conditions, we build a synthetic model of the adversary's possible behaviour. We first define the trust boundaries and systematically model integrity and availability threats against the components, control actions and feedback in the SCS. For simplicity, we do not consider threats against *confidentiality* which (generally) do not impact safety. We assume that an attacker can conduct *tampering*, *spoofing*, and *denial of service* (DoS) attacks against the components, control actions and feedback of SCS (Figure 9.2) which are also included in the SBM. Overall, we consider a total of 34 threats (17 threats to availability and 17 to integrity). We use the threat model to build a synthetic attacker. This is a timed automaton which starts from a *Idle/Ready* location from where any threat can be launched in a non-deterministic fashion. When a threat is launched, the target enters a failure mode the effects of which propagate through the safe control structure.

9.1.8 Formal verification

The safe behavioural model captures how conditions such as *alternate modes of functioning*, but also loss of *availability* or *reliability* affect control actions and feedback, and how their effects propagate through the control structure. Earlier in this thesis we have proposed a correspondence between locations of TA representing control actions and the types of unsafe control actions defined in STPA (Section 4.3).

We leverage our proposed correspondence to define the following two statements to verify UCA_1, UCA_2, UCA_3, UCA_4, UCA_5, UCA_6, UCA_7, and UCA_8:

1. $\phi_1 := E <> fmssteering.Unreliable$

   Explanation: Is there a situation where *FMS Steering Output* becomes unreliable following an attack ($UCA_1 - UCA_8$)?

2. $\phi_2 := E <> fmssteering.Unavailable$

   Explanation: Is there a situation where *FMS Steering Output* becomes unreliable following an attack ($UCA_4 - UCA_8$)?

9.2 Discovery & Enumeration of threat scenarios

The formal verification of ϕ_1 and ϕ_2 on the instrumented model allows us to explore all the possible combinations of threats that, alone or through cascading effects, lead to safety violations. In particular, we use our toolchain to enumerate threat scenarios starting from ϕ_1 and ϕ_2. This enables us to uncover a set of scenarios that is complete with respect to the threats, component behaviours and bounds imposed on the model checking. We show the outcome of

Table 9.3: Threat scenarios found.

Hazard	Spec	T	CT
H_1	ϕ_1	12	$30.92s$
H_1	ϕ_2	9	$31.387s$

the verification of ϕ_1 and ϕ_2 ($\bar{t} = 300s$) leading to the hazard H_1 in Table 9.3. For each ϕ we

also report the total number scenarios found (T), and the cumulative time (CT) taken to verify each ϕ. Cassandra finds 12 threat scenarios potentially leading to the hazards H_1 through ϕ_1, while 9 additional scenarios leading to H_1 are found through the verification of ϕ_2.

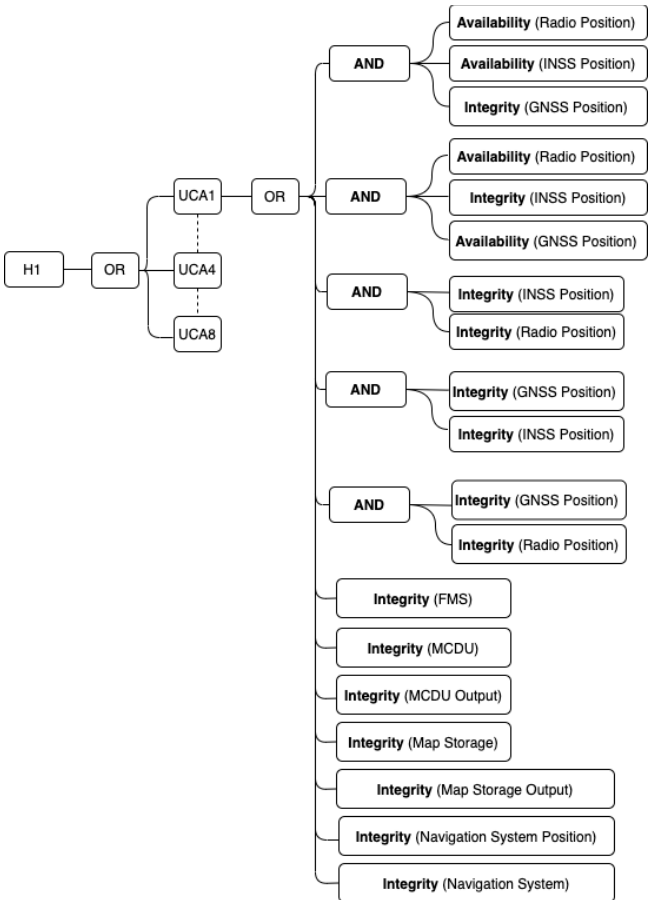


Figure 9.6: Threat scenarios tree (UCA_1)

Figure 9.6 shows an example of the threat scenario tree derived for H_1 through the verification of ϕ_1. Interestingly, while the last four scenarios are straightforward as the threats directly affect the FMS or its input, scenarios $1-5$, show all the combinations of attacks on the INSS, GNSS and Radio subsystems that can affect the decisions taken by the FMS while the aircraft is on autopilot.

9.3 Analysis of safety-critical attack paths

We employ MulVAL [OGA+05] to derive the safety-critical attack paths, i.e., the sequences of privileges an attacker needs to obtain to execute a threat scenario and thus cause a haz-

ard. The attack graph of the CPS shows the paths leading to a target threat scenario. For brevity, we focus on two types of privileges within the architecture: *network access* and *code execution*. This step of Cassandra relies on two assumptions (Section 6.2.1). First, we assume that the architecture of the aircraft is known. Then, we assume that a security assessment has been carried out to identify the vulnerabilities and weaknesses that affect the different components, and the privileges such vulnerabilities would grant to an adversary following a successful exploitation. We leverage the usual traceability requirements - normally adopted within the development life-cycle of safety-critical systems - to link threats against elements of the safe control structure to components in the architecture, i.e., which privileges enable which threats. In Table 9.4, we show the subset of threats we have considered in our proposed use case. In essence, an adversary that acquires network access to the most critical domain

Table 9.4: Threats and privileges in the use case. Spoofing and Tampering threats affect the integrity whereas DoS affects availability. NA: Network access. Exec: Code execution.

Privilege	**Threat**
$NA(GNSSNet)$ $NA(control.domain)$	$\{(integrity, GNSSPosition)\}$
$NA(internet)$ $NA(GNSSNet)$	$\{(availability, GNSSPosition)\}$
$NA(control.domain)$	$\{(integrity, INSSPosition)\}$
$NA(control.domain)$	$\{(availability, INSSPosition)\}$
$NA(control.domain)$	$\{(integrity, RadioPosition)\}$
$NA(control.domain)$	$\{(availability, RadioPosition)\}$
$NA(control.domain)$	$\{(integrity, Position(NavSysOut))\}$
$NA(control.domain)$. $Exec(FMS, root)$ $Exec(DLSr, root)$	$\{(integrity, mapDataOut)\}$
$NA(control.domain)$. $Exec(FMS, root)$	$\{(integrity, fmsSteeringOut)\}$
$NA(control.domain)$	$\{(availability, fmsSteeringOut)\}$
$NA(control.domain)$	$\{(integrity, mcduOut)\}$

on the aircraft (*aircraft.control.domain*) poses a threat to the integrity and the availability of most - or all - safety-relevant control actions and feedback previously defined in the safe control structure. Among these, for simplicity, we only consider *GNSS Position*, *Radio Position*, *INSS Position*, *NavSysOut*, *mapDataOut*, *fmsSteeringOut*, and *mcduOut*. Similarly, an adversary who gains network visibility of the GNSS receiver (e.g., via strong satellite signal) poses a threat to the integrity and/or the availability of the *GNSS Position* feedback. Finally, code

execution privileges on the *DLSLoader* and *FMS* are a threat to the integrity of *mapDataOut* and *fmsSteeringOut*.

9.3.1 Generation of the attack graph

We generate the attack paths leading to the execution of the threat scenarios relevant to UCA_1 in *three* distinct cases:

(T_1) Electronic Flight Bag (EFB) is provided by the airline and has access to the internet, or is available under a bring-your-own-device (BYOD) policy.

(T_2) EFB is provided by the airline and cannot access the internet but is enabled with network access towards an airline-owned enterprise network.

(T_3) EFB does not have access to the internet.

To perform this analysis we run MulVAL providing, as input, *three* variations of the aircraft architecture, each encoding a different case. The attack graph generation process (Section 6.2.1) uses the target scenarios as an additional input; we use threat scenarios $TS-1$ to $TS-12$ for this. In the first scenario (EFB with connectivity to the internet), the attack graph produced has 136 vertices and 155 edges, representing all the possible known attack paths that enable the execution of the attack (Figure 9.7). With the same settings, the attack graph produced when EFB has limited connectivity (no internet access) is slightly smaller (126 vertices and 143 edges) (Figure 9.8). Finally, no attack paths are found when the EFB has no connectivity at all and at least two among GNSS, INSS and Radio are operating correctly. This is expected as we are not considering physical attacks on the EFB. However, if we run MulVAL while INSS and Radio are down (e.g., due to a failure), MulVAL correctly finds that GNSS spoofing is a valid path leading to UCA_1. We have reported the attack graph generated for the two scenarios in Figures 9.7 and 9.8 for reference.

In the first configuration (Figure 9.7), we have found two main routes to TS-1 and TS-2; both require access to the Electronic Flight Bag as an intermediate step. The most straightforward

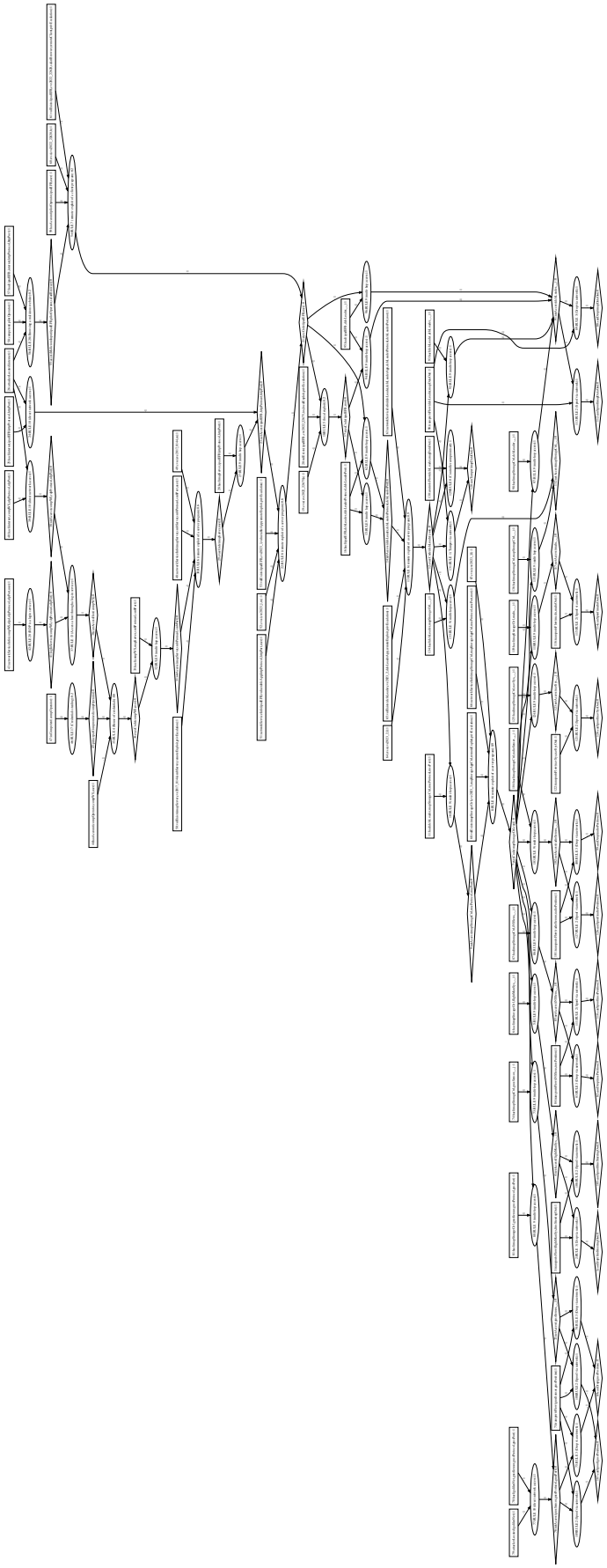


Figure 9.7: Attack Graph - (Scenario 1).

path towards the EFB exploits `CVE-2022-22620`, which grants code execution privileges on the device immediately. The second route requires the attacker to access the *enterprise network*; we assume this is possible through the reuse of valid credentials. When inside the enterprise network, they gain network visibility of the *mapServer*, which we have assumed to be affected by `CVE-2017-0144`. The exploitation of `CVE-2017-0144` grants code execution privileges on the *mapServer* and is a precondition to the exploitation of `Vuln-X`. We have assumed that `Vuln-X` is a vulnerability that affects the proprietary map application on the EFB and, when exploited, grants code execution privileges on the device.

Thus, the attacker exploits `CVE-2022-22675` to escalate privileges on the EFB and obtain *privileged* code execution rights on the device. After that, they can inject code on the DLS (`Vuln-Y`), which grants them visibility of the rest of the avionics. Access to the *mapStorage* unit is possible through additional lateral movement (`Vuln-Z`). An additional path (GNSS Spoofing) allows the attacker who has access to a powerful transmitter to spoof the GNSS position of the plane. We model the scenario as *network access* to the GNSS receiver, over GNSS. Code execution on the DLS as well as on the *mapStorage*, automatically gives the attacker the *ability* of executing TS-2. Similarly, code execution on *mapStorage* together with *network access* to the *GNSS*, automatically grant the attacker with the *capability* of execution TS-1

In the second case (Figure 9.8), the EFB cannot directly access the internet, thus the adversary cannot easily exploit `CVE-2022-22620`. Instead, the only viable path is the one through the *enterprise network*. Finally, we do not find any feasible attack path in the third scenario since we bring the EFB offline.

In all configurations, the analysis of the attack paths shows that the EFB is the only attack vector the adversary can use to access the onboard avionics. The main difference between the first two configurations is that in the first one, the attacker can directly access the EFB by exploiting `CVE-2022-22620`. A summary of the scenario 1 is provided by the BAG in Figure 9.9.

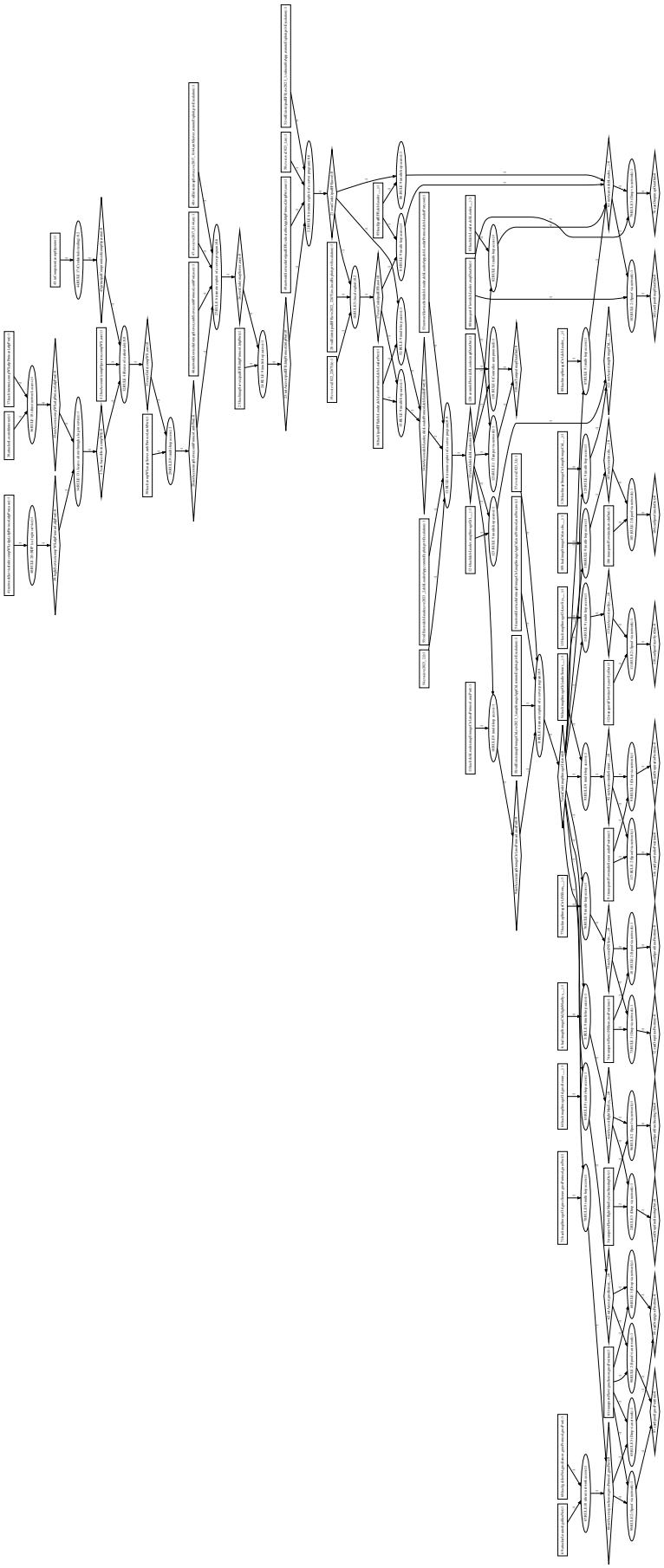


Figure 9.8: Attack Graph - (Scenario 2).

9.3.2 Likelihood of Threat Scenarios

We employ Bayesian Attack Graphs (BAG) [LM05] to compute the marginal probabilities of an adversary being successful in a threat scenario. The graph produced by MulVAL is a tripartite directed graph which also admits cycles. In order to apply Bayesian inference, we need to extract a BAG from the graph produced by MulVAL, we do so in the two steps previously outlined in Chapter 6. We start with the extraction of a Directed A-Cyclic Graph (DAG). Thus, we transform the DAG into a BAG. In essence, privileges in MulVAL are treated as privileges in the DAG, and MulVAL *derivation rules* are used to reconstruct the edges. Figure

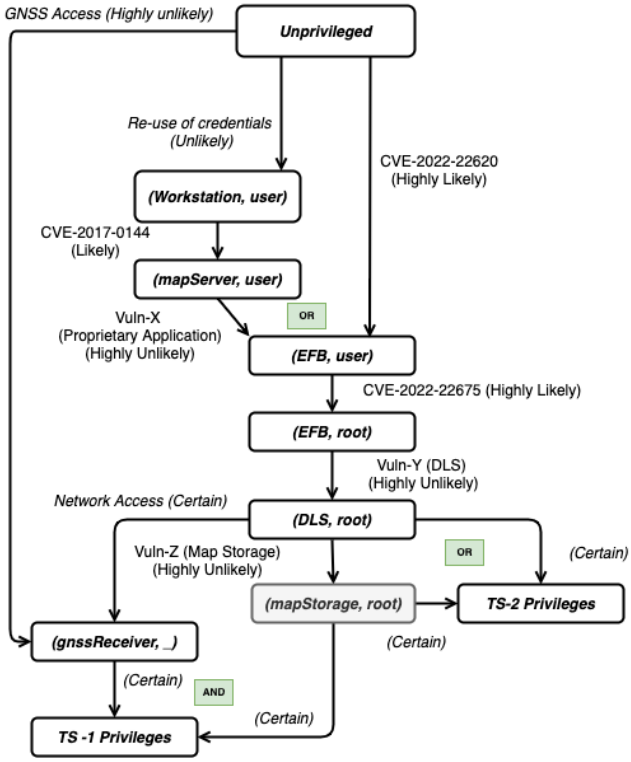


Figure 9.9: Attack Graph - (Configuration 1).

9.9 shows the attack graph produced for the case T_1, with targets $TS-1$ and $TS-2$

($TS-1$): *Integrity(GNSS Position)* AND *Availability(Radio Position* AND *Availability(INSS Position).*

($TS-2$): *Integrity(MapStorageOut)*

We use Bayesian inference as described in [MGSBL17] on the resulting DAG. Vertices are modelled using Bernoulli random variables with the probability of the vertex v_i being *compromised*

by the adversary being $P(V_i = 1) = p$ [MGSBL17]. BAG edges model dependencies between vertices. In this case, a dependency represents a vulnerability the attacker exploits to acquire a new privilege. The weight p_{ei} of the $i-th$ edge reflects the likelihood of the vulnerability i being exploited. In particular, we have used the following values for the inference *Highly-Unlikely* ($p = 0.1$), *Unlikely* ($p = 0.2$), *Likely* ($p = 0.5$), and *Highly-Likely* ($p = 0.8$).

When possible, we use the Common Vulnerability Scoring System (CVSS) as a reference to estimate the likelihood of exploitations for vulnerabilities, in line with the related work in this area [MGSBL17]. Vulnerabilities `Vuln-X`, `Vuln-Y`, and `Vuln-Z` affect software and devices that are not widely available to the public and, for this reason, may not have been assigned a CVE or a CVSS. Given their high specificity, we assume that each of them, although severe, is *highly unlikely* to be exploited. Finally, we consider a *reuse of credential unlikely* to be exploited; this reflects MulVAL [OGA+05] default configuration, when the victim user is *competent* (i.e., they have some degree of security awareness, use multi-factor authentication, etc.). By acquiring *root* privileges on the DLS, the adversary automatically also acquires the privileges necessary to execute $TS-2$. Similarly, by becoming *root* on the map storage, they also acquire the privileges related to $TS-2$ and (part of) $TS-1$. We model such cases using a *Certain* ($p = 1$) edge (Figure 9.9).

Table 9.5 shows the marginal probabilities of the attacker successfully executing a threat scenario ($TS-1$ or $TS-2$) in the three different cases. Overall, we obtain that, in the first

Table 9.5: Marginal Probabilities

Configuration	TS	Marginal Probability
EFB Internet access	$TS-1$	$3 * 10^{-4}$
EFB Internet access	$TS-2$	$26 * 10^{-3}$
EFB Airline restricted access	$TS-1$	$< 10^{-5}$
EFB Airline restricted access	$TS-2$	$2 * 10^{-4}$
EFB disconnected	$TS-1$	$\leq 10^{-1}$
EFB disconnected	$TS-2$	NA

configuration the marginal probability of $TS-1$ is $3 * 10^{-4}$ while that of $TS-2$ is $26 * 10^{-3}$. This is coherent with $TS-1$ requiring more conditions to be fulfilled than $TS-2$. Marginal probabilities of the two threat scenarios are, respectively, $< 10^{-5}$ and $2 * 10^{-4}$ under the second configuration, when the EFB has limited connectivity. Finally, if connectivity on the EFB is

disabled, $TS-2$ cannot be achieved while $TS-1$ is achievable with a marginal probability of 0.1 if and only if the NS is working in alternate mode GNSS only (worst case). However, without access to the aircraft through the EFB, the adversary would not be able to directly cause a loss of availability of both radio and INSS systems. Hence, the adversary needs the two systems to have failed independently already for the attack to be successful.

9.4 Discussion

Using Cassandra we have evaluated the risk associated with *three* different policies regarding management and configuration of EFB. The application of the first step of Cassandra on a subset of our aircraft SCS led to the discovery of 23 threat scenarios, ordered sequences of threats that malicious actors can actuate to cause a hazard. We leveraged the integration of the aircraft control structure with its architecture to generate the attack paths leading to the privileges that the attacker needs to engage in 12 of the found threat scenarios. We have identified two main paths of compromise starting from outside the system perimeter when the EFB is equipped with internet connectivity. The first path reaches the ACD through the airline enterprise network which, for simplicity, we have represented as a flat network with two hosts, a workstation, and a map server. By reusing credentials [Mit22, Dra22] the adversary can access the workstation and then pivot towards the map server. Network visibility on the EFB enables them to exploit `Vuln-X` - which we have assumed. A second path leads the adversary from the outside directly to the EFB through the exploitation of `CVE-2022-620`. Assuming that `Vuln-X` cannot be exploited from the internet (e.g., traffic blocked), the attacker can exploit `CVE-2022-22675` to acquire root privileges on the EFB. Then, they can exploit `Vuln-Y` and `Vuln-Z` ASD and ACD. The edge on the far left of Figure 9.9 shows that the GNSS service is also reachable from outside the system perimeter. As expected, the attack graph generated when the EFB does not have internet access does not contain the edge `CVE-2022-620` Figure 9.9. The qualitative analysis of the attack graph shows that the EFB is a bottleneck in the attack graphs. It also shows that the feasibility of threat scenarios depends on the configuration of the aircraft architecture; by disconnecting the EFB from the internet and then from the airline network, we

increasingly limit the number of threat scenarios that can be exploited. Trading off usability - workload imposed on the crew - against the security gains requires a quantitative analysis of the attack graphs produced. We do so by reasoning over the probabilities using Bayesian inference in the third step of Cassandra. The analysis of the marginal probabilities of threat scenarios yields three key results. Firstly, we show that the third configuration (EFB isolated) is the one that carries the least risk, which is intuitive. However, it also increases the workload for the crew, as limited connectivity also reduces the availability of automation in uploading data to the EFB. Secondly, low-level security controls (e.g., fixing of vulnerabilities) applied on airline corporate networks and devices are critical to contain the risk of successful attacks targeting the ACD through the EFB. In our use case, we show that, through the containment of vulnerabilities `CVE-2022-22675` and `CVE-2017-144`, we can reduce the marginal probabilities of $TS-2$ below 10^{-5}. To this extent, it is possible to be compliant with certification while also providing the Flight Crew with automated processes. Thirdly, internet connectivity on devices that interact with the aircraft network (e.g., EFB tablet) is associated with a significant risk and is highly discouraged.

9.5 Summary & Conclusions

In this chapter, we have applied Cassandra to analyse the impact of attacks on the safe operations of a representative flight management system (FMS). The objective of this analysis was to gauge the effects of different security policies on the risk related to threats targeting flight control operations. In particular, we have evaluated three different security policies around the management of electronic flight bags.

To achieve said evaluation we have first executed Cassandra's enumeration of threat scenarios using a *partial knowledge* behavioural model to describe the behaviour of system components. By applying this approach, we have substantially reduced verification times with respect to the previous use cases, where we employed *full-knowledge* behavioural model. In earlier chapters, we have discussed the differences between these two approaches and the main advantages and

disadvantages of choosing one over the other (Chapter 4)

Starting from the threat scenarios identified, we computed safety-critical attack paths. Thus, we have leveraged exact Bayesian inference to quantify the risk in the presence of the three different security controls.

Our results help security analysts determine best security practices to reduce the risk brought by safety-critical threats, while at the same time guaranteeing the usability of the system. Results obtained for the use case presented in this chapter were published in [CSP+23].

This chapter presented the last use case analysed with Cassandra in this thesis. The next chapter presents the conclusions, a discussion on the potential applications of our approach to real-world contexts, and possible directions for future works.

Chapter 10

Conclusion

Cyber-physical systems pervade all aspects of daily life. As these systems become more interconnected and more network-enabled devices are interfaced with the OT, their attack surface also becomes wider. While the overall number of possible attacks is increasing, only some will impact the safety of the CPS. Identifying and mitigating safety-critical attacks requires a combined approach to safety and security analysis. Although several attempts have been made to integrate the two, no definitive methodology has emerged. A comprehensive methodology should, in the first place, build on accepted safety and security methodologies so that its results are acceptable and usable for creating assurance cases. In order to produce *valuable* results, the methodology should relate to a behavioural model of system components. Doing so enables the analysis of cascading effects, thus the identification of attacks whose impact also depends on the current state of the system. At the same time, the methodology should also integrate with the deployment context (architecture). Doing so enables the identification of specific vulnerabilities and weaknesses that facilitate safety-critical threats, which is crucial in identifying and prioritising mitigations. The adoption of a systematic approach is also critical to eliminate subjectivity, reduce human error, and encourage reproducibility of results. Lastly, automation can be leveraged to reduce the skills necessary to perform the analysis.

We have proposed Cassandra, a novel methodology that combines STPA-Sec with STRIDE and applies model-checking to identify safety-critical threat scenarios and the vulnerabilities

that facilitate them. Cassandra uses STRIDE to systematically build a model of the attacker and integrates with the architecture of the CPS to identify safety-critical attack paths. Equally importantly, we have also proposed an integrated toolchain to automate the analysis and demonstrated its use in realistic use cases with increasing complexity and mirroring of real-world conditions.

Like all systems, Cassandra has some limitations. These mainly stem from the capability of existing tools to cope with complexity, in particular for model checking and attack graph generation. A more detailed discussion of the limitations of our approach and possible directions for future work follows later in this chapter.

Overall, the impact of our work is threefold. Firstly, it enables the enumeration of safety-critical threat scenarios, thus how the safety of a critical system can be compromised by a deliberate adversary. Secondly, formal verification guarantees the soundness of these scenarios. Thus, once appropriate mitigations have been defined and deployed this can form part of a formal assurance process. A main challenge in integrating safety and security methodologies is that the concepts and system models are defined independently. We have defined rigorous mappings in order to relate the two and define an integrated methodology. Thirdly, by decoupling the formal verification from the attack graph analysis, we can evaluate the impact of vulnerabilities on safety without repeating the formal verification. A common limitation of traditional integrated approaches is that they do not distinguish between the adversarial actions and the attack paths an adversary needs to use to be able to perform them. This does not allow practical mitigations to be implemented. By clearly distinguishing and separating between the design level constructs and the system architecture (deployment) we enable the practical use of this methodology in existing systems. Additionally, the functional design and mitigations can be analysed separately. For example, during the case study *"Analysing the impact of attacks on the safety of flight management systems"* (Chapter 9), we have evaluated the impact of three different security policies, without repeating the formal verification.

This chapter is structured as follows. In Section 10.1, we summarise thesis achievements. We elaborate on the potential application of our approach to real-world contexts in Section 10.2.

Finally, in Section 10.3, we discuss the limitations of our strategy and possible directions for future work.

10.1 Summary of thesis achievements

The methodology we propose, Cassandra, is built on three main building blocks, namely, System Theoretic Process Analysis for security (STPA-Sec), model-checking, and attack path analysis. We have organised it in *three* stages: *threat scenarios discovery and enumeration*, *derivation of safety-critical attack paths*, and *path analysis* (Chapter 3).

To achieve a systematic and automatic discovery and enumeration of safety-critical threat scenarios, in stage one of Cassandra, we have combined STPA-Sec with STRIDE and formal verification in novel ways. The first step towards this combination is the definition of a formal description of the STAMP model (Section 2.6). In order to formally verify the existence of attacks leading to safety violations, we have introduced the following elements:

- A safe behavioural model which formally describes the propagation of the impact of threats within the CPS. Specifically, we consider two different types of models to be used alternatively depending on the amount of knowledge available about the behaviour of system components. A *full-knowledge* model is preferred when the internal behaviour of system components is known and formalised. On the other hand, a *partial-knowledge* approach is to be chosen when the internal behaviour of system components is unknown. This allows to handle a variety of scenarios and flexible integration with different types of process design. For both approaches introduced, we also designed the necessary mapping with STAMP and STPA (Chapter 4). The formal definition of a behavioural model which considers the impact of adversarial actions interleaved with system behaviours and, at the same time, maps with STAMP and STPA is a novel contribution of this thesis.

- A model of adversarial actions which mimics the behaviour of an adversary. Following the application of STRIDE on the STAMP model of the CPS, we have used the threats

found to build a behavioural representation of the adversary (Section 5.2). The systematic derivation of a synthetic attacker performed in Cassandra is also novel. In fact, existing approaches based on STPA do not generally integrate the threats identified in the security analysis and a formal verification process of their impact, taking into account cascading effects. We have introduced the concept of synthesising a behavioural representation of the adversary from the security analysis to achieve this and show how this can be done with different levels of sophistication, trading off between the completeness of the analysis and its computational complexity.

- A set of TCTL properties (or statements), which we use to formally verify the reachability of STPA unsafe control actions (UCA). These are systematically derived from UCA through further mappings that we have defined between the types of UCA expected by STPA and the safe behavioural models (for both *partial* and *full-knowledge* approaches, Chapter 4).

We have introduced the process of threat scenarios discovery and enumeration (Chapter 5), which enables Cassandra to identify and also enumerate threat scenarios given a model of the behaviour for CPS components, a synthetic adversary, and the safety properties. Novel contributions of our work with respect to this process include the verification of safety properties in adversarial settings, the definition of formal relations between the models, and the development of an algorithm which enumerates threat scenarios through the verification of reachability statements over multiple iterations. The process is supported by a tool, which we have designed and implemented as a JAVA program and uses UPPAAL [BDL+12] JAVA API (Section 5.3.4) for formal verification.

Furthermore, we have introduced the *Accident-Threat-Vulnerability* model (Chapter 6) which extends [MF14] with elements from the STAMP domain. By linking STAMP objects to vulnerabilities and weaknesses in the architectural model, we make an important contribution towards the realisation of a more complete integrated analysis. In fact, the *Accident-Threat-Vulnerability* model enables us to relate high-level hazards to individual low-level exploitations.

We have introduced the notion of minimal safety set (MSS), i.e., the minimum set of privileges

that should be defended to guarantee safety. The broad concept of finding minimal groups of objects (e.g., components) which must operate reliably to guarantee safety is often adopted in the literature related to safety. Nevertheless, the identification of such privileges, whose protection guarantees safety, is unprecedented and crucial to later prioritise mitigations. This research of MSS is also supported by a tool, which we have developed using Microsoft Z3 [MB08] (Section 6.3.1).

During the second stage of Cassandra, we use MulVal [OGA+05] to generate system-wide attack paths (i.e., from the assumed perimeter to privileges in the OT), which we use to search for potential mitigations by looking at common nodes across the paths leading to the privileges MSS. The way we identify the MSS on the system architecture, i.e., from the results of the first stage of Cassandra, via the *Accident-Threat-Vulnerability* model, along with the research of common privileges to protect across the safety-critical attack paths produced, constitutes an original contribution to the best of our knowledge. Our strategy innovates with respect to traditional integrated methodologies, which usually look for high-level mitigation on the control structure. This process of qualitative risk analysis is also supported by several algorithms which we have developed for the ingestion and manipulation of attack graphs (Chapter 6).

As part of our contributions, we have also extended MulVAL [OGA+05] with a set of facts, predicates, and rules to express threats. Our newly defined set of rules enables a swift integration between MulVAL and Cassandra, allowing the generation of attack paths leading to threats. This extension is essential to the generation of safety-critical attack paths, which include high-level elements such as threats and hazards as well as vulnerabilities and privileges from the specific deployment contexts. Cassandra relies on such paths to quantitatively estimate the risk of safety-critical attacks.

Finally, Cassandra integrates with existing tools for Bayesian inference [MGSBL17] to achieve quantitative risk analysis during its third stage. To facilitate this integration, as part of our contribution, we have designed an algorithm to build a Bayesian network starting from the attack graph produced by MulVAL. Our approach explicitly addresses eventual cycles in the attack graph and differs from other strategies in the literature in the choice of model parameters,

and the number of nodes in the produced Bayesian attack graph. The latter is substantially lower, following a pruning operation (Section 6.4.2).

The process of BAG extraction is also supported by a tool which implements the algorithm outlined in Section 6.4.2. Additional software was implemented to support the Bayesian inference via the program developed in [MGSBL17]. These include a wrapper for the original tool to enable the use through remote procedure calls and an ingestion algorithm, which we use to read the Bayesian attack graph produced by us and convert it into the formalism accepted by [MGSBL17].

10.2 Applications

The purpose of Cassandra is the identification of safety-critical attacks. Its application is intended specifically for cyber-physical systems where ensuring safety is the priority. We see three possible scenarios of application for our methodology.

When deployed within the development life-cycle of safety-critical systems, Cassandra can evaluate design choices at the functional, behavioural, and architectural levels. In Figure 10.1,

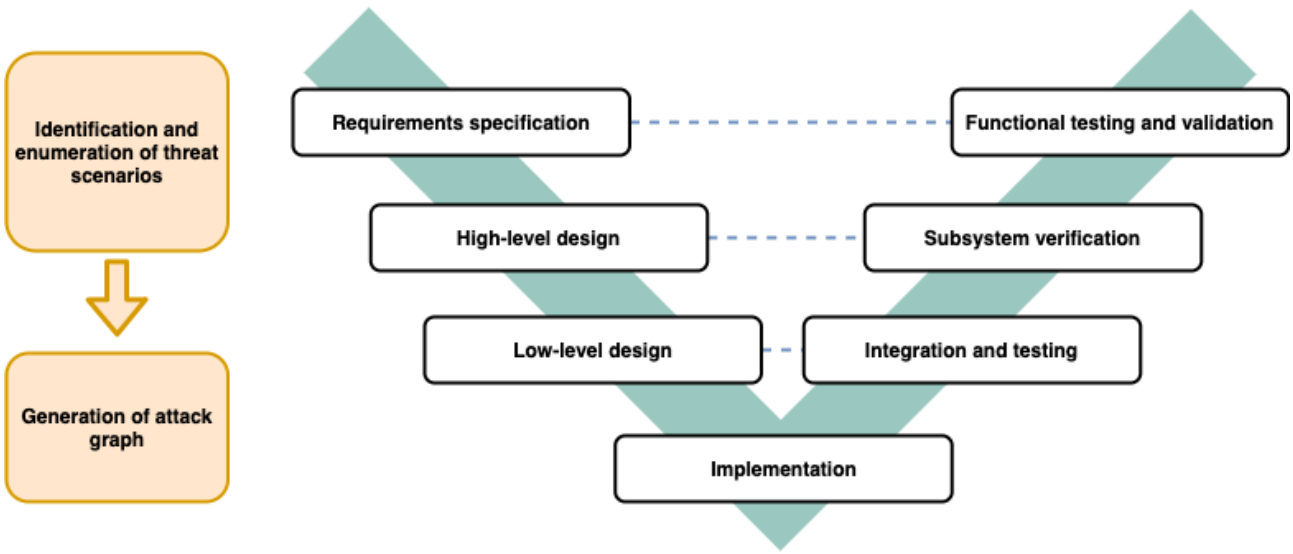


Figure 10.1: Integration of Cassandra in the V development process

we show a possible integration with the *V* model traditionally adopted for the development

of critical systems. Our process of *identification and enumeration of threat scenarios* can be applied along with the definition of functional and behavioural models (high-level design). When the architecture of the system is provided (low-level design), we can proceed with the *generation of the attack graph* and the *analysis of the paths found*. The latter is not shown in Figure for simplicity.

A second scenario of application of our methodology features its use to gauge the safety of a given design to attacks. Results can be of use to harden the architecture and apply further security controls such as firewall rules, network diodes, and physical protections aiming to safeguard privileges in the minimal safety set and reduce the risks associated with threat scenarios. Disconnecting hosts and rewiring connections, however, can have an impact on safety as well. It is important that the safety of the new topology is verified before applying the changes.

A third scenario sees a practical application of Cassandra in real-time as further aid during incident response operations. When an intrusion is detected at the enterprise level of the system, the security operation centre (SOC) can leverage information regarding critical attack paths to disconnect hosts or even rewire the infrastructure to safeguard privileges in the minimal safety set. Also, in this case, it is important that the safety of the new topology is verified before applying the changes.

The three use cases analysed in this thesis fall under the second scenario of application of our methodology. During the analysis of the Communication Based Train Control system (Chapter 7), we analysed the architecture and identified the minimal safety set, with the objective of selecting security controls. For the Electric Power Intelligent and Control use-case (Chapter 8) we have derived safety-critical attack paths and performed a qualitative analysis aimed at identifying the most relevant hosts in the network. Lastly, for the Flight Management System use-case (Chapter 9) we have derived safety-critical attack paths and performed a quantitative evaluation of the risk in the presence of three different security policies.

While applying Cassandra, the available knowledge on system behaviour should be evaluated and a *partial knowledge* model should be preferred when the behaviour of system components is uncertain, considered overly complex, or in the presence of other constraints, e.g., real-time

constraints during incident response applications. The identification of threat scenarios in the presence of a *full-knowledge* model is more complete (also considering the threat model and verification bounds), albeit slower than its counterpart. For the analysis of the Communication Based Train Control (Chapter 7) and Electric Power Intelligent and Control (Chapter 8) scenarios, we have assumed that the behavioural model of components is known. To this extent, we have proceeded with an application of Cassandra using a *full-knowledge* approach. On the other hand, we have assumed that our knowledge of the behaviour of the Flight Management System (Chapter 9) analysed in the third scenario was not accurate enough to be explicitly modelled. In this case, we went ahead with an application of Cassandra using a *partial-knowledge* approach.

10.3 Limitations & future work

The overarching objective of our research was the development of a systematic methodology for the automatic identification of threat scenarios which, at the same time, is also underpinned by a sound safety model. Aiming to enhance the practical usability of the scenarios found, our approach integrates with the context deployment to qualitatively assess and quantify risks.

During the development of Cassandra , we put a considerable amount of effort into the identification of a sound safety model which would suit our purposes, and in the development of formalism and mapping needed to support formal verification and enable automatic identification of cascading effects. We have adopted STAMP as the safety model, and its appliation is performed manually, as in all the related work. Although, in principle, it would be feasible to (partially) automate the identification of the control structure in STAMP by relying further on design elements of the system, this remains an item for future work. A limitation of Cassandra is related to the computational complexity of model-checking operations. Further optimisations to manage the complexity of the verification process are possible, for example, by synthesising a more specific model for attacker behaviour using more information from the attack graphs or by using more efficient strategies for formal verification.

To enable the generation of safety-critical attack paths, we have established relations between threats targeting elements of the STAMP model and the exploitation of vulnerabilities in the specific deployment context. We have focused on the identification of safety-critical attacks rather than on the selection of security controls. By doing so, we are able to determine candidate mitigations; however, mitigations may impact the safety and performance of the system. It is often desirable to perform aided or automated mitigation selection. To achieve this, *mapping back* from the potential security controls identified on the attack graph to functional element is needed. This circumstance was partially encountered during the analysis of the Flight Management System (Chapter 9) where we evaluated the effectiveness of different security policies. In this specific case, we have used three distinct architectural models, one implementing each security policy, to generate the attack graphs.

We have assumed that a mapping between STRIDE threats and the privileges necessary to perform them is available. In our use cases, we have manually identified these privileges in the system architecture. However, in real system deployments, the process can be partially automated by examining the permission required associated with the threats identified during the threat modelling stage. We leave the research on automating the identification of these privileges for future work.

During the development of Cassandra, we have prioritised research issues over the integration with frameworks and tools usually adopted in industries. Another direction for future work would be porting our approach to environments such as SysML or AADL. By doing so, we can achieve a tighter integration of Cassandra with Model-based systems engineering (MBSE). For example, taxonomies and ontology such as [PHNT19] would be valuable for porting to SysML. Such operation would also allow the use of existing frameworks such as SysML-Sec to carry out more advanced analysis., such as evaluating the verification of security controls.

Moreover, a further possible direction for future research would feature the implementation of Cassandra within a real-time monitor for CPS to orchestrate incident response operations. In this sense, when an intrusion is detected at the enterprise level of the system, information on safety-critical attack paths can be leveraged with the aim of safeguarding relevant privileges on

the OT.

Overall, additional possible directions for future work include further optimisations of the threat scenarios enumeration process, increasing the level of details of the relations established with the specific deployment context, and developing more advanced algorithms for the selection of security measures.

Appendix A

Safety-critical attack paths on communication-based train control (CBTC) system

In this section, we show in detail the attack graph generated by MulVAL for the Communication-Based Train Control (CBTC) system (Chapter 7). We show the complete graph in Figure 7.12. It counts 333 nodes, of these 170 are *and* nodes, 45 are *or* nodes, and 118 are basic facts (e.g., security conditions, network conditions, etc.).

For improved readability, we have partitioned the attack graph in *seven* colour-coded sections, which we show in the pages below in individual figures. Sections are for visual clarity and do not correspond to any semantic divisions within the graph.

We notice *four* clustered regions in the attack graph, one for each network area in the system. The first cluster (Figure A.2 top right corner) consists of a small subgraph which encodes the attack path leading to a GNSS spoofing attack. Since spoofing attacks do not allow to execute code on the target, the adversary cannot use lateral movements and the component is disconnected from the rest of the attack graph. The second cluster (also Figure A.2) shows the attack paths in the enterprise network. The adversary gains a *foothold* through the exploitation

of vulnerabilities `Vuln-03` (RCE) and `Vuln-04` (RCE). A series of lateral movements allows them to execute code, with different privileges on an internal application server, which they can use to pivot towards the operation network. The two clustered regions are highlighted by a grey background in Figure A.1.

A third cluster in the attack graph encodes attack paths in the operation network of the CBTC system. The adversary can access this area via the exploitation of `Vuln-15` (RCE), which grants execution privileges on the *Interface Server (Operation)*. In fact, from the *Interface Server (Operation)* the attacker gains network access to all devices in the operation network (p_8), and can execute lateral movements towards the he GSM-Base and the zone controllers, respectively, $p_2, p_4, p_8, p_{10}, p_{11}$, and p_{12}. For improved readability, we split the cluster over two graphical sections Figure A.1, highlighted by a blue and red background respectively. Individual figures for each region are provided in Figures A.3 and A.4.

Finally, the fourth cluster includes attack paths in the mobile (train) network, which is accessible via the exploitation of `Vuln-26` in our architecture. Due to the topology of the network the number of attack paths is quite high. To this extent, we have split its representations over four graphical sections (Figure A.1) highlighted by a yellow (Figure A.5), green (Figure A.6), light blue (Figure A.7) and purple (Figure A.8) background, respectively. On the mobile network, the adversary gains network visibility of onboard networked components and can obtain the safety-critical privileges $p_1, p_3, p_5, p_6, p_7, p_9, p_{13}$, and p_{14} through further lateral movements.

Clustered region	**Network area**	**Graphical region (Figure A.1)**
1	GNSS	Graphical region 1 (grey background)
2	Enterprise	Graphical region 1 (grey background)
3	Operations (zone control)	Graphical region 2 (blue background) Graphical region 3 (red background)
4	Mobile (train)	Graphical region 4 (yellow background) Graphical region 5 (green background) Graphical region 6 (light blue background) Graphical region 7 (purple background)

Table A.1: Structure of the attack graph

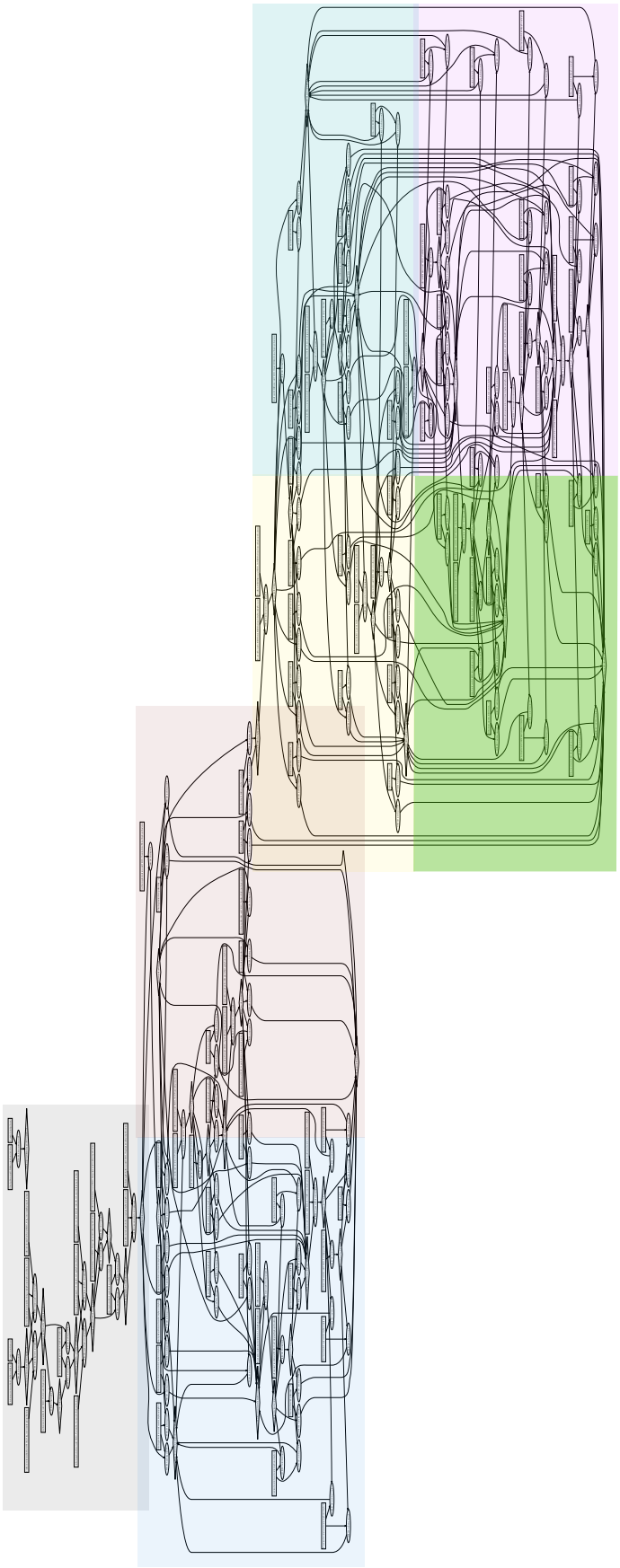


Figure A.1: Colours encode different subsections of the attack graph

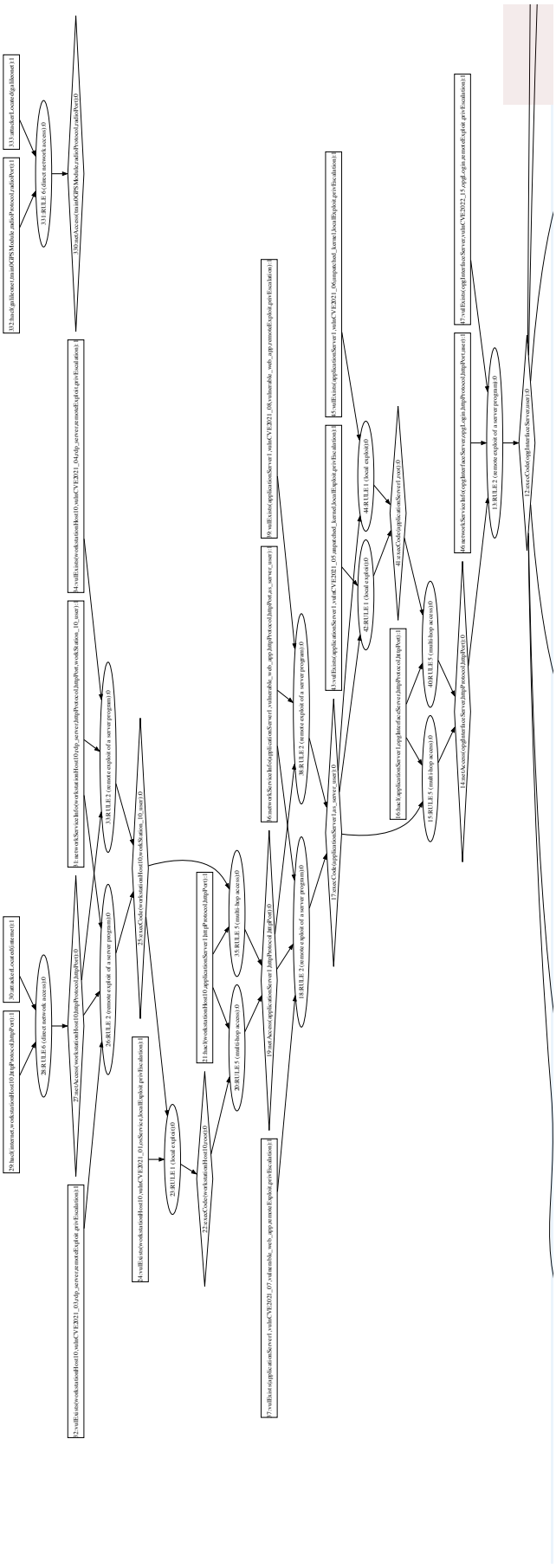


Figure A.2: Safety-critical attack paths - Section 1 (grey)

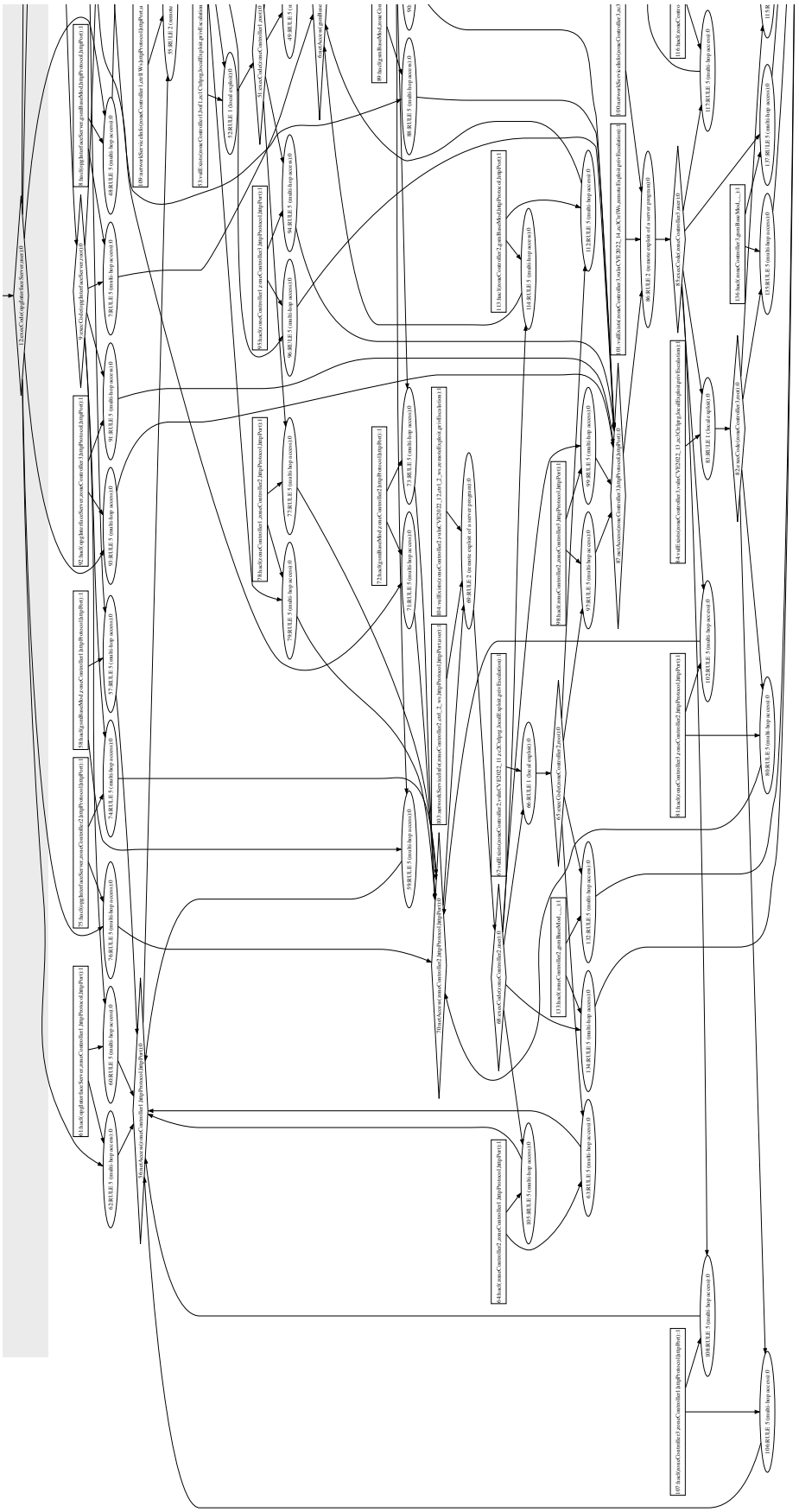


Figure A.3: Safety-critical attack paths - Section 2 (blue)

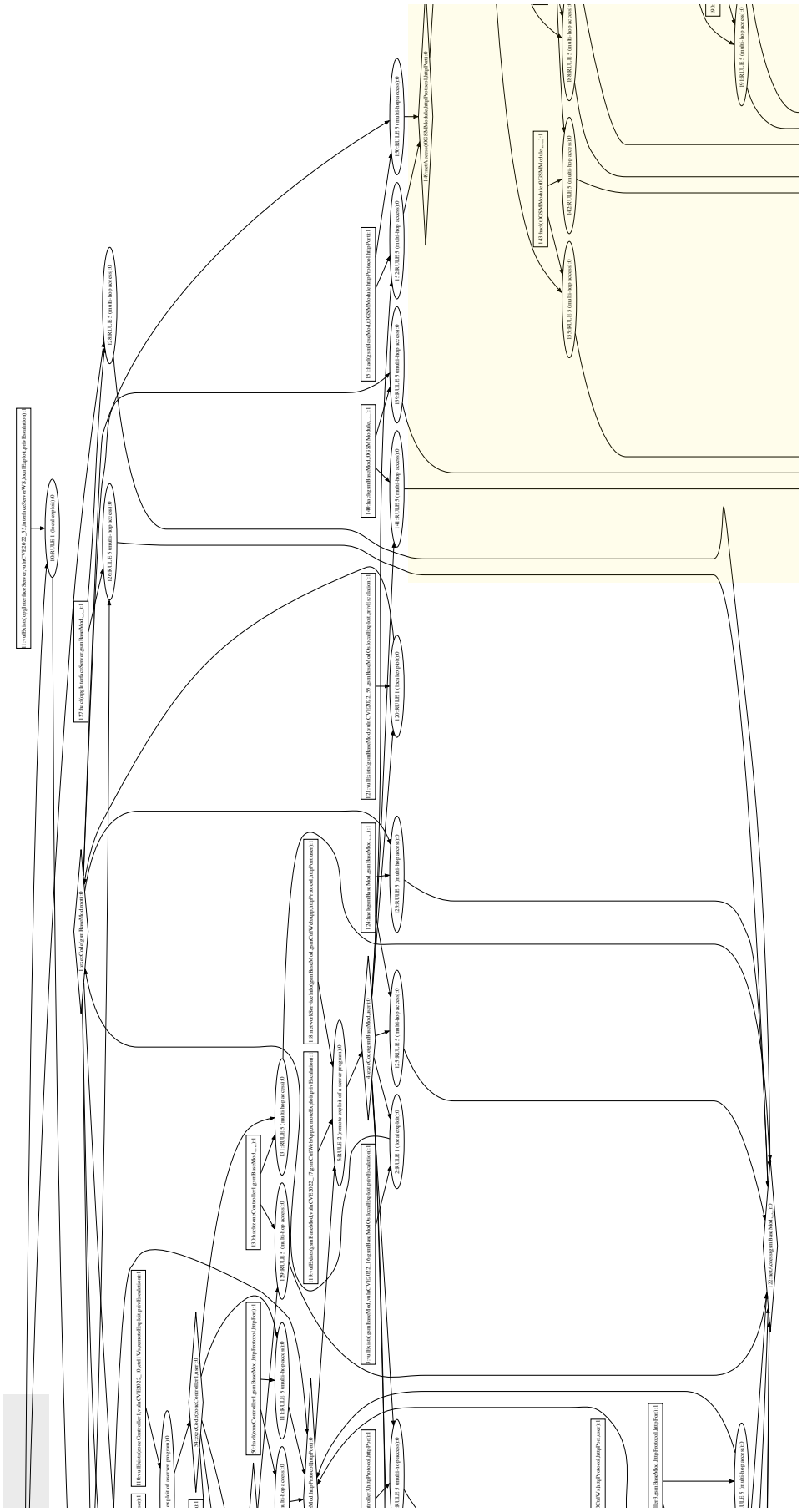


Figure A.4: Safety-critical attack paths - Section 3 (red)

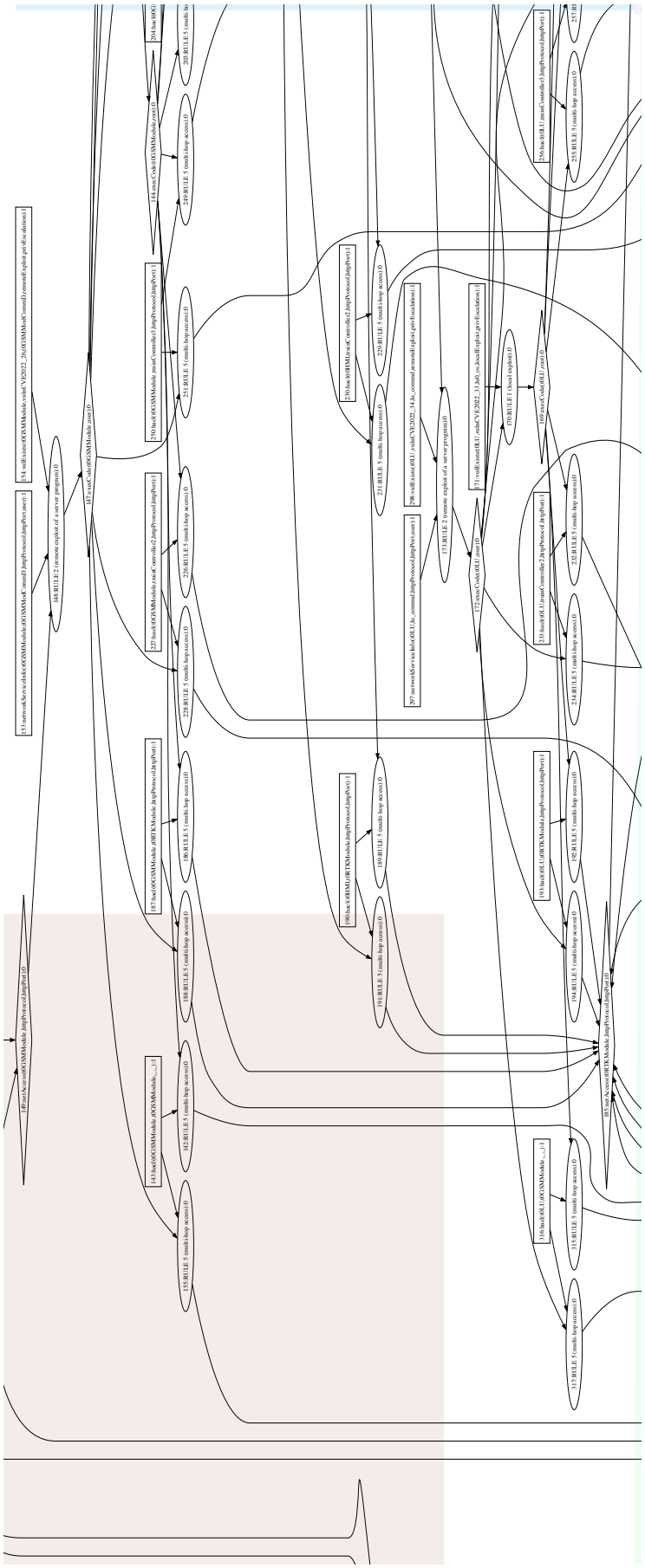


Figure A.5: Safety-critical attack paths - Section 4 (yellow)

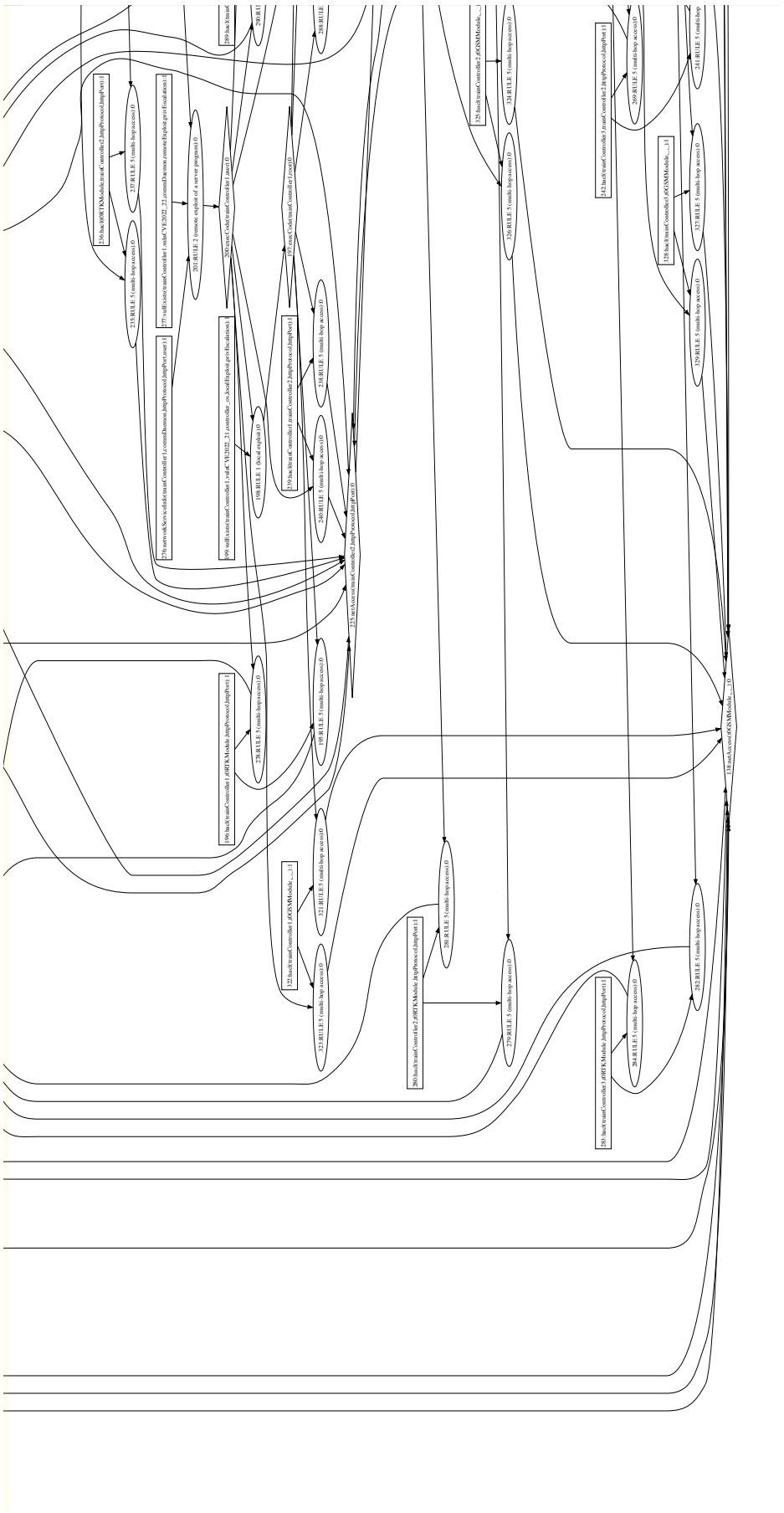


Figure A.6: Safety-critical attack paths - Section 5 (green)

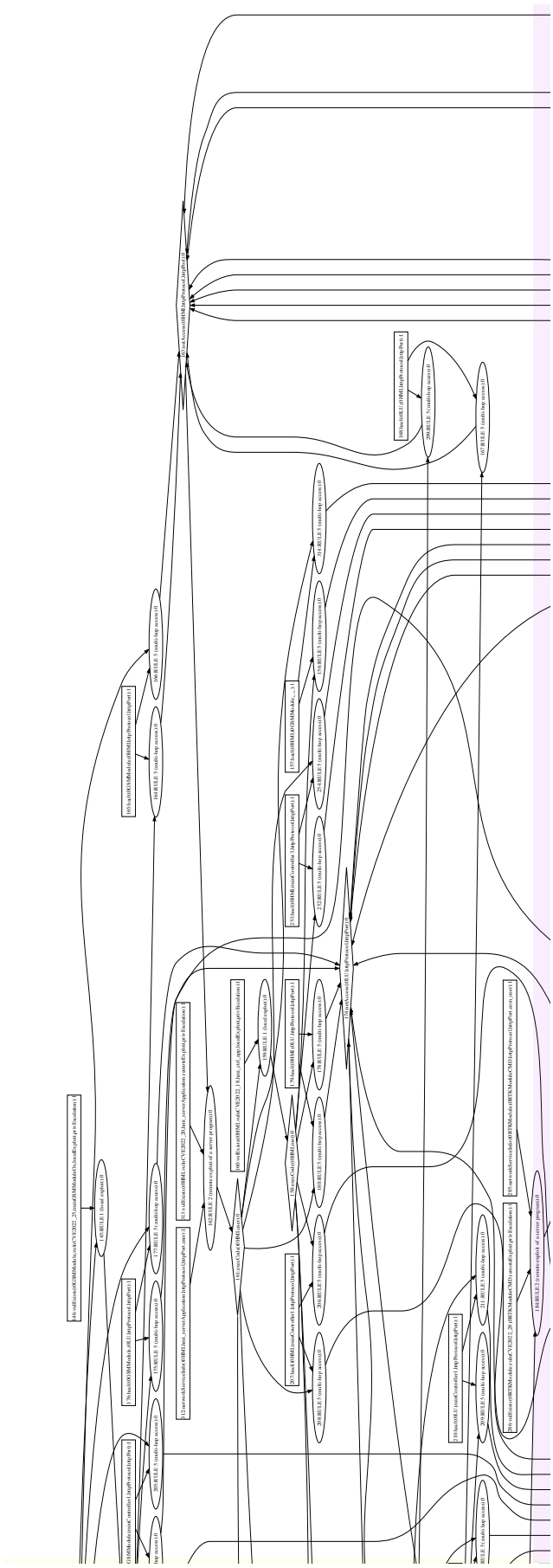


Figure A.7: Safety-critical attack paths - Section 6 (light blue)

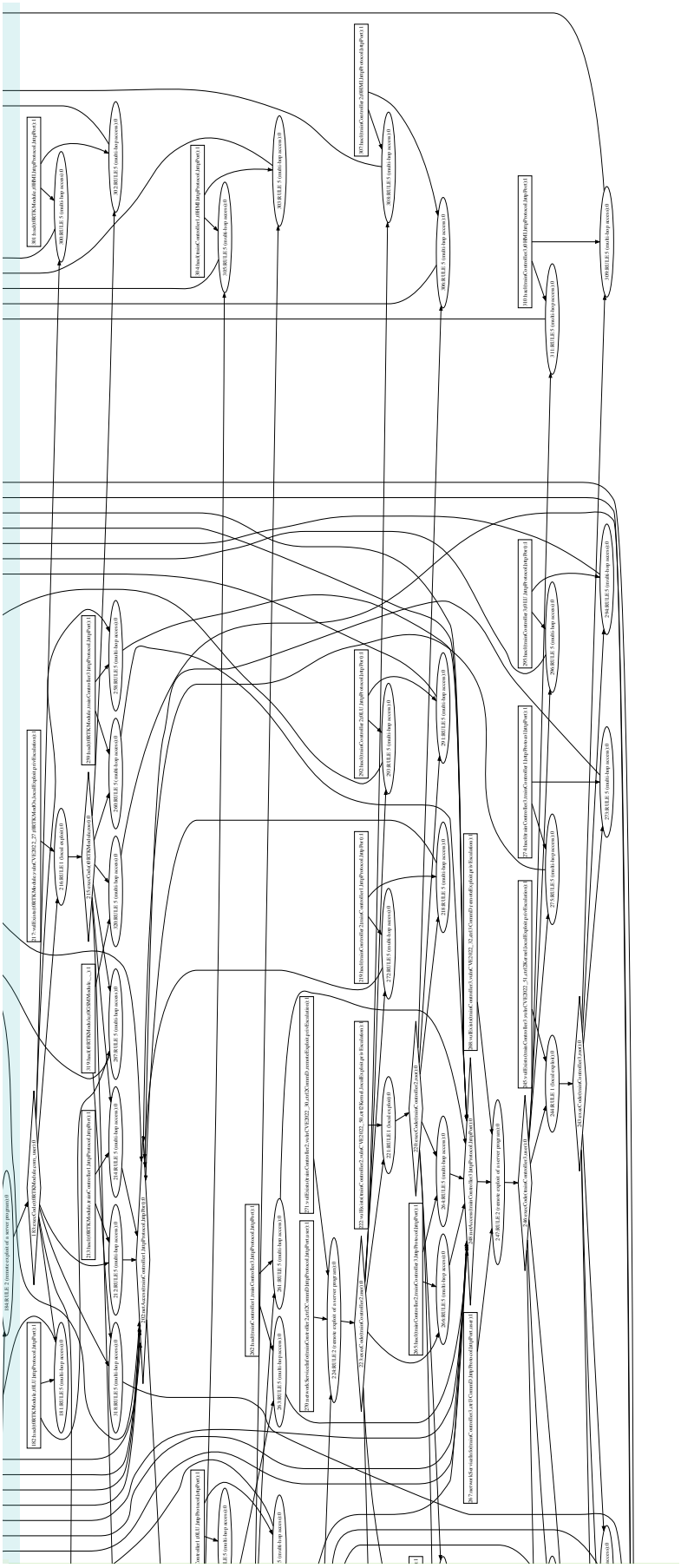


Figure A.8: Safety-critical attack paths - Section 7 (purple)

Bibliography

[AB05] Martín Abadi and Bruno Blanchet. Analyzing security protocols with secrecy types and logic programs. *Journal of the ACM (JACM)*, 52(1):102–146, 2005.

[ACD90] Rajeev Alur, Costas Courcoubetis, and David Dill. Model-checking for real-time systems. In *[1990] Proceedings. Fifth Annual IEEE Symposium on Logic in Computer Science*, pages 414–425. IEEE, 1990.

[AD90] Rajeev Alur and David Dill. Automata for modeling real-time systems. In *International colloquium on automata, languages, and programming*, pages 322–335. Springer, 1990.

[ADSW03] Christopher Alberts, Audrey Dorofee, James Stevens, and Carol Woody. Introduction to the octave approach. *Pittsburgh, PA, Carnegie Mellon University*, pages 72–74, 2003.

[AHH+13] Michael D Adams, Seth D Hitefield, Bruce Hoy, Michael C Fowler, and T Charles Clancy. Application of cybernetics and control theory for a new paradigm in cybersecurity. *arXiv preprint arXiv:1311.0257*, 2013.

[AHM19] Abdel-Latif Alshalalfah, Ghaith Bany Hamad, and Otmane Ait Mohamed. Towards system level security analysis of artificial pancreas via uppaal-smc. In *2019 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–5. IEEE, 2019.

[AJN12] Massimiliano Albanese, Sushil Jajodia, and Steven Noel. Time-efficient and cost-effective network hardening using attack graphs. In *IEEE/IFIP Interna-*

tional Conference on Dependable Systems and Networks (DSN 2012), pages 1–12. IEEE, 2012.

[AK86] Krzysztof R. Apt and Dexter Kozen. Limits for automatic verification of finite-state concurrent systems. *Inf. Process. Lett.*, 22(6):307–309, 1986.

[AKM18] Sridhar Adepu, Nandha Kumar Kandasamy, and Aditya Mathur. Epic: An electric power testbed for research and training in cyber physical systems security. In *Computer Security*, pages 37–52. Springer, 2018.

[AKZM20] Sridhar Adepu, Nandha Kumar Kandasamy, Jianying Zhou, and Aditya Mathur. Attacks on smart grid: Power supply interruption and malicious power generation. *International Journal of Information Security*, 19(2):189–211, 2020.

[AL15] Michael J Assante and Robert M Lee. The industrial control system cyber kill chain. *SANS Institute InfoSec Reading Room*, 1:24, 2015.

[ALS13] Philip Asare, John Lach, and John A Stankovic. Fstpa-i: A formal approach to hazard identification via system theoretic process analysis. In *Proceedings of the ACM/IEEE 4th International Conference on Cyber-Physical Systems*, pages 150–159, 2013.

[Apv08] Ludovic Apvrille. Ttool for diplodocus: An environment for design space exploration. In *Proceedings of the 8th international conference on New technologies in distributed systems*, pages 1–4, 2008.

[AR13] Ludovic Apvrille and Yves Roudier. Sysml-sec: A sysml environment for the design and development of secure embedded systems. *APCOSEC, Asia-Pacific Council on Systems Engineering*, pages 8–11, 2013.

[Ash61] W Ross Ashby. *An introduction to cybernetics*. Chapman & Hall Ltd, 1961.

[ATT21] MITRE ATT&CK. Mitre att&ck. *URL: https://attack. mitre. org*, 2021.

[Ave62] Catherine B Avery. The new century classical handbook. *(No Title)*, 1962.

[AW15] Asim Abdulkhaleq and Stefan Wagner. Integrated safety analysis using systems-theoretic process analysis and software model checking. In *Computer Safety, Reliability, and Security: 34th International Conference, SAFECOMP 2015, Delft, The Netherlands, September 23-25, 2015, Proceedings 34*, pages 121–134. Springer, 2015.

[AW16] Asim Abdulkhaleq and Stefan Wagner. A systematic and semi-automatic safety-based test case generation approach based on systems-theoretic process analysis. *arXiv preprint arXiv:1612.03103*, 2016.

[AWK02] Paul Ammann, Duminda Wijesekera, and Saket Kaushik. Scalable, graph-based network vulnerability analysis. In *Proceedings of the 9th ACM Conference on Computer and Communications Security*, pages 217–224, 2002.

[AWL15] Asim Abdulkhaleq, Stefan Wagner, and Nancy Leveson. A comprehensive safety engineering approach for software-intensive systems based on stpa. *Procedia Engineering*, 128:2–11, 2015.

[BBC19] BBC. Raspberry pi used to steal data from nasa lab, 2019.

[BBC21] BBC. Hacker tries to poison water supply of florida city, 2021.

[BCEF18] Georgios Bakirtzis, Bryan T Carter, Carl R Elks, and Cody H Fleming. A model-based approach to security analysis for cyber-physical systems. In *2018 Annual IEEE International Systems Conference (SysCon)*, pages 1–8. IEEE, 2018.

[BDL06a] Gerd Behrmann, Alexandre David, and Kim G Larsen. A tutorial on uppaal 4.0. *Department of computer science, Aalborg university*, 2006.

[BDL+06b] Gerd Behrmann, Alexandre David, Kim Guldstrand Larsen, John Håkansson, Paul Pettersson, Wang Yi, and Martijn Hendriks. Uppaal 4.0. *IEEE Computer Society*, 2006.

[BDL+12] Peter Bulychev, Alexandre David, Kim Gulstrand Larsen, Marius Mikučionis, Danny Bøgsted Poulsen, Axel Legay, and Zheng Wang. Uppaal-smc: Statistical model checking for priced timed automata. *arXiv preprint arXiv:1207.1272*, 2012.

[BH15] Anton Bretting and Mei Ha. Vehicle control unit security using open source autosar, 2015.

[BH21] Martín Barrère and Chris Hankin. Analysing mission-critical cyber-physical systems with and/or graphs and maxsat. *ACM Transactions on Cyber-Physical Systems*, 5(3):1–29, 2021.

[BM16] Dennis M Buede and William D Miller. *The engineering design of systems: models and methods*. John Wiley & Sons, 2016.

[BSC+19] Georgios Bakirtzis, Brandon J Simon, Aidan G Collins, Cody Harrison Fleming, and Carl R Elks. Data-driven vulnerability exploration for design phase system analysis. *IEEE Systems Journal*, 14(4):4864–4873, 2019.

[BSFE18] Georgios Bakirtzis, Brandon J Simon, Cody H Fleming, and Carl R Elks. Looking for a black cat in a dark room: Security visualization for cyber-physical system design and analysis. In *2018 IEEE Symposium on Visualization for Cyber Security (VizSec)*, pages 1–8. IEEE, 2018.

[BTB+19] Victor Bolbot, Gerasimos Theotokatos, Luminita Manuela Bujorianu, Evangelos Boulougouris, and Dracos Vassalos. Vulnerabilities and safety assurance methods in cyber-physical systems: A comprehensive review. *Reliability Engineering & System Safety*, 182:179–193, 2019.

[BW19] Peter Beaumont and Stephen Wolthusen. Micro-grid control security analysis: Analysis of current and emerging vulnerabilities. *Critical Infrastructure Security and Resilience: Theories, Methods, Tools and Technologies*, pages 159–184, 2019.

[BY03] Johan Bengtsson and Wang Yi. Timed automata: Semantics, algorithms and tools. In *Advanced Course on Petri Nets*, pages 87–124. Springer, 2003.

[CAL+11] Alvaro A Cárdenas, Saurabh Amin, Zong-Syun Lin, Yu-Lun Huang, Chi-Yen Huang, and Shankar Sastry. Attacks against process control systems: risk assessment, detection, and response. In *Proceedings of the 6th ACM symposium on information, computer and communications security*, pages 355–366. ACM, 2011.

[CAS08] Alvaro A Cardenas, Saurabh Amin, and Shankar Sastry. Secure control: Towards survivable cyber-physical systems. In *Distributed Computing Systems Workshops, 2008. ICDCS'08. 28th International Conference on*, pages 495–500. IEEE, 2008.

[CCQ17] Luis F Cómbita, Alvaro A Cárdenas, and Nicanor Quijano. Mitigation of sensor attacks on legacy industrial control systems. In *2017 IEEE 3rd Colombian Conference on Automatic Control (CCAC)*, pages 1–6. IEEE, 2017.

[CDP+17] Mathieu Comptier, David Déharbe, Julien Molinero Perez, Louis Mussat, Thibaut Pierre, and Denis Sabatier. Safety analysis of a cbtc system: a rigorous approach with event-b. In *International Conference on Reliability, Safety and Security of Railway Systems*, pages 148–159. Springer, 2017.

[CFP+20] Luca Maria Castiglione, Paolo Falcone, Alberto Petrillo, Simon Pietro Romano, and Stefania Santini. Cooperative intersection crossing over 5g. *IEEE/ACM Transactions on Networking*, 2020.

[CGM+12] Darren Cofer, Andrew Gacek, Steven Miller, Michael W Whalen, Brian LaValley, and Lui Sha. Compositional verification of architectural models. In *NASA Formal Methods Symposium*, pages 126–140. Springer, 2012.

[CHC+22] Luca Maria Castiglione, Zhongyuan Hau, Kenneth T Co, Luis Muñoz-González, Fei Teng, and Emil Lupu. Ha-grid: Security aware hazard analysis for

smart grids. In *2022 IEEE International Conference on Communications, Control, and Computing Technologies for Smart Grids (SmartGridComm)*, pages 446–452. IEEE, 2022.

[CL20] Luca Maria Castiglione and Emil C Lupu. Hazard driven threat modelling for cyber physical systems. In *Proceedings of the 2020 Joint Workshop on CPS&IoT Security and Privacy*, pages 13–24, 2020.

[CL23] Luca Maria Castiglione and Emil C. Lupu. Which attacks lead to hazards? combining safety and security analysis for cyber-physical systems. *IEEE Transactions on Dependable and Secure Computing*, pages 1–16, 2023.

[CLRS22] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. *Introduction to algorithms*. MIT press, 2022.

[CMT18] Z Berkay Celik, Patrick McDaniel, and Gang Tan. Soteria: Automated iot safety and security analysis. In *2018 {USENIX} Annual Technical Conference ({USENIX}{ATC} 18)*, pages 147–158, 2018.

[CNN11] CNN. Faa approves ipads for pilots electronic charts, 2011.

[CSP+23] Luca Maria Castiglione, Philipp Stassen, Cora Perner, Daniel Patrick Pereira, Gustavo de Carvalho Bertoli, and Emil C. Lupu. Don't panic! analysing the impact of attacks on the safety of flight management systems. In *2023 IEEE/AIAA 42nd Digital Avionics Systems Conference (DASC)*, pages 1–10, 2023.

[CT15] Frank Crawley and Brian Tyler. *HAZOP: Guide to best practice*. Elsevier, 2015.

[DAR21] DARPA. darpa.mil. `https://www.darpa.mil/program/cyber-assured-systems-engineering`, 2021. [Accessed 05-12-2023].

[DLL+15] Alexandre David, Kim G Larsen, Axel Legay, Marius Mikučionis, and Danny Bøgsted Poulsen. Uppaal smc tutorial. *International Journal on Software Tools for Technology Transfer*, 17(4):397–415, 2015.

[Dra22] Dragos. Chernovite's pipedream malware targeting industrial control systems (ics), 2022.

[DSCdMBH20] Nivio Paula De Souza, Cecília de Azevedo Castro César, Juliana de Melo Bezerra, and Celso Massaki Hirata. Extending stpa with stride to identify cybersecurity loss scenarios. *Journal of Information Security and Applications*, 55:102620, 2020.

[DV18] Alheri Longji Dakwat and Emilia Villani. System safety assessment based on stpa and model checking. *Safety science*, 109:130–143, 2018.

[E+15] Clifton A Ericson et al. *Hazard analysis techniques for system safety*. John Wiley & Sons, 2015.

[EGK+02] John Ellson, Emden Gansner, Lefteris Koutsofios, Stephen C North, and Gordon Woodhull. Graphviz—open source graph drawing tools. In *Graph Drawing: 9th International Symposium, GD 2001 Vienna, Austria, September 23–26, 2001 Revised Papers 9*, pages 483–484. Springer, 2002.

[EL99] Clifton A Ericson and Clifton Ll. Fault tree analysis. In *System Safety Conference, Orlando, Florida*, volume 1, pages 1–9, 1999.

[ESE22] ESET. Industroyer2. Industroyer reloaded. `https://www.welivesecurity.com/2022/04/12/industroyer2-industroyer-reloaded/`, 2022. [Online; accessed 10-April-2023].

[EUR18] EUROCAE. *ED-203A - Airworthiness Security Methods and Considerations*. EUROCAE, 2018.

[EVCC+22] Herson Esquivel-Vargas, John Henry Castellanos, Marco Caselli, Nils Ole Tippenhauer, and Andreas Peter. Identifying near-optimal single-shot attacks on

icss with limited process knowledge. In *International Conference on Applied Cryptography and Network Security*, pages 170–192. Springer, 2022.

[FGH06] Peter H Feiler, David P Gluch, and John J Hudak. The architecture analysis & design language (aadl): An introduction. Technical report, Carnegie-Mellon Univ Pittsburgh PA Software Engineering Inst, 2006.

[FJS+19] Song Fang, Karl Henrik Johansson, Mikael Skoglund, Henrik Sandberg, and Hideaki Ishii. Two-way coding in control systems under injection attacks: From attack detection to attack correction. *arXiv preprint arXiv:1901.05420*, 2019.

[FMS+17] Ivo Friedberg, Kieran McLaughlin, Paul Smith, David Laverty, and Sakir Sezer. Stpa-safesec: Safety and security analysis for cyber-physical systems. *Journal of information security and applications*, 34:183–196, 2017.

[Fou] The Apache Software Foundation. ActiveMQ. `https://activemq.apache.org/`. [Accessed 09-08-2023].

[Fow12] Martin Fowler. *Patterns of Enterprise Application Architecture: Pattern Enterpr Applica Arch.* Addison-Wesley, 2012.

[Gre] Greenbone. Openvas. [Online; accessed 10-April-2023].

[H+06] Matthew Hause et al. The sysml modelling language. In *Fifteenth European Systems Engineering Conference*, volume 9, pages 1–12, 2006.

[HBCS19] Giles Howard, Michael Butler, John Colley, and Vladimiro Sassone. A methodology for assuring the safety and security of critical infrastructure based on stpa and event-b. *International Journal of Critical Computer-Based Systems*, 9(1-2):56–75, 2019.

[HBS+02] Mark Hapner, Rich Burridge, Rahul Sharma, Joseph Fialli, and Kate Stout. Java message service. *Sun Microsystems Inc., Santa Clara, CA*, 9, 2002.

[HCK+15] Danielle Hiltunen, Stephanie G Chase, Andrew Kendra, Young Jin Jo, et al. Electronic flight bag (efb) 2015 industry survey. Technical report, John A. Volpe National Transportation Systems Center (US), 2015.

[HCMB18] Syed Hussain, Omar Chowdhury, Shagufta Mehnaz, and Elisa Bertino. Lteinspector: A systematic approach for adversarial testing of 4g lte. In *Network and Distributed Systems Security (NDSS) Symposium 2018*, 2018.

[HCS+21] Shengtuo Hu, Qi Alfred Chen, Jiachen Sun, Yiheng Feng, Z Morley Mao, and Henry X Liu. Automated discovery of denial-of-service vulnerabilities in connected vehicle protocols. In *USENIX Security Symposium*, 2021.

[Hel12] Philipp Helle. Automatic sysml-based safety analysis. In *Proceedings of the 5th International Workshop on Model Based Architecting and Construction of Embedded Systems*, pages 19–24, 2012.

[HMvdW+20] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, September 2020.

[HOM08] John Homer, Xinming Ou, and Miles A McQueen. From attack graphs to automated configuration management—an iterative approach. *Kansas State University Technical Report*, 2008.

[Hor51] Alfred Horn. On sentences which are true of direct unions of algebras1. *The Journal of Symbolic Logic*, 16(1):14–21, 1951.

[How04] Jeffrey Howard. Preserving system safety across the boundary between system integrator and software contractor. *SAE transactions*, pages 170–181, 2004.

[HTLC15] Adam Hahn, Roshan K Thomas, Ivan Lozano, and Alvaro Cardenas. A multi-layered and kill-chain based security analysis framework for cyber-physical systems. *International Journal of Critical Infrastructure Protection*, 11:39–50, 2015.

[HYLS20] Jia Huang, Jian-Xin You, Hu-Chen Liu, and Ming-Shun Song. Failure mode and effect analysis improvement: A systematic literature review and future research agenda. *Reliability Engineering & System Safety*, 199:106885, 2020.

[IAT22] IATA. Releases 2022 Airline Safety Performance — iata.org. `https://www.iata.org/en/pressroom/2023-releases/2023-03-07-01/#:~:text=The%20industry%202022%20fatality%20risk,(average%20of%2022%2C116%20years).`, 2022. [Accessed 06-12-2023].

[IEC18] IEC. Iec 60812:2018 — failure modes and effects analysis (fmea and fmeca). `https://webstore.iec.ch/publication/26359`, 2018. [Accessed 06-12-2023].

[Inc22] Dragos Inc. Crashoverride analysis of the threat to electric grid operations 2022, 2022.

[Int96] Sae International. *Guidelines and methods for conducting the safety assessment process on civil airborne systems and equipment*. SAE International, 1996.

[iso18] iso.org. Iso 26262-1:2018 - road vehicles – functional safety. `https://www.iso.org/standard/68383.html`, 2018. [Accessed 01-12-2023].

[JNO05] Sushil Jajodia, Steven Noel, and Brian O'berry. Topological analysis of network attack vulnerability. In *Managing cyber threats*, pages 247–266. Springer, 2005.

[Joh11] Chris Johnson. Cybersafety: Cybersecurity and safety-critical software engineering. In *Achieving Systems Safety: Proceedings of the Twentieth Safety-Critical Systems Symposium, Bristol, UK, 7-9th February 2012*, pages 85–95. Springer, 2011.

[KBC+14] Siwar Kriaa, Marc Bouissou, Frederic Colin, Yoran Halgand, and Ludovic Pietre-Cambacedes. Safety and security interactions modeling using the bdmp formalism: case study of a pipeline. In *International Conference on Computer Safety, Reliability, and Security*, pages 326–341. Springer, 2014.

[KBL19] Siwar Kriaa, Marc Bouissou, and Youssef Laarouchi. A new safety and security risk analysis framework for industrial control systems. *Proceedings of the Institution of Mechanical Engineers, Part O: Journal of risk and reliability*, 233(2):151–174, 2019.

[KGM+18] Sahand Hadizadeh Kafash, Jairo Giraldo, Carlos Murguia, Alvaro A Cardenas, and Justin Ruths. Constraining attacker capabilities through actuator saturation. In *2018 Annual American Control Conference (ACC)*, pages 986–991. IEEE, 2018.

[KLHP21] Hyungju Kim, Mary Ann Lundteigen, Andreas Hafver, and Frank Børre Pedersen. Utilization of risk priority number to systems-theoretic process analysis: A practical solution to manage a large number of unsafe control actions and loss scenarios. *Proceedings of the Institution of Mechanical Engineers, Part O: Journal of Risk and Reliability*, 235(1):92–107, 2021.

[KM21] Shaharyar Khan and Stuart E Madnick. Cybersafety: A system-theoretic approach to identify cyber-vulnerabilities & mitigation requirements in industrial control systems. *IEEE Transactions on Dependable and Secure Computing*, 2021.

[KMLS17] Rafiullah Khan, Kieran McLaughlin, David Laverty, and Sakir Sezer. Stride-based threat modeling for cyber-physical systems. In *2017 IEEE PES Innovative Smart Grid Technologies Conference Europe (ISGT-Europe)*, pages 1–6. IEEE, 2017.

[KMM18] Shaharyar Khan, Stuart Madnick, and Allen Moulton. Cyber-safety analysis of an industrial control system for chillers using stpa-sec. *MIT Sloan*, 2018.

[KNKS22] Rajesh Kumar, Bhavesh Narra, Rohan Kela, and Siddhant Singh. Afmt: Maintaining the safety-security of industrial control systems. *Computers in Industry*, 136:103584, 2022.

[KPCBH15] Siwar Kriaa, Ludovic Pietre-Cambacedes, Marc Bouissou, and Yoran Halgand. A survey of approaches combining safety and security for industrial control systems. *Reliability engineering & system safety*, 139:156–178, 2015.

[Lam05] Leslie Lamport. Real time is really simple. *Microsoft Research*, pages 2005–30, 2005.

[LAVM18] Qin Lin, Sridha Adepu, Sicco Verwer, and Aditya Mathur. Tabor: A graphical model-based approach for anomaly detection in industrial control systems. In *Proceedings of the 2018 on asia conference on computer and communications security*, pages 525–536, 2018.

[LCCZ19] Stefano Longari, Andrea Cannizzo, Michele Carminati, and Stefano Zanero. A secure-by-design framework for automotive on-board network risk analysis. In *2019 IEEE Vehicular Networking Conference (VNC)*, pages 1–8. IEEE, 2019.

[LDB20] Harjinder Singh Lallie, Kurt Debattista, and Jay Bal. A review of attack graph and attack tree visual syntax in cyber security. *Computer Science Review*, 35:100219, 2020.

[Lev86] Nancy G Leveson. Software safety: Why, what, and how. *ACM Computing Surveys (CSUR)*, 18(2):125–163, 1986.

[Lev15] Nancy G. Leveson. Is estimating probabilities the right goal for system safety? `http://sunnyday.mit.edu/papers/Making-Safety-Decisions.pdf`, 2015. [Accessed 03-12-2023].

[Lev16] Nancy G Leveson. *Engineering a safer world: Systems thinking applied to safety*. The MIT Press, 2016.

[LM05] Yu Liu and Hong Man. Network vulnerability assessment using bayesian networks. In *Data mining, intrusion detection, information assurance, and data networks security 2005*, volume 5812, pages 61–71. SPIE, 2005.

[LMMV20] Ruggero Lanotte, Massimo Merro, Andrei Munteanu, and Luca Viganò. A formal approach to physics-based attacks in cyber-physical systems. *ACM Transactions on Privacy and Security (TOPS)*, 23(1):1–41, 2020.

[LŠČ18] Elena Lisova, Irfan Šljivo, and Aida Čaušević. Safety and security co-analyses: A systematic literature review. *IEEE Systems Journal*, 13(3):2189–2200, 2018.

[LT18] Nancy G Leveson and John P Thomas. Stpa handbook. *Cambridge, MA, USA*, 2018.

[LV62] Robert E Lyons and Wouter Vanderkulk. The use of triple-modular redundancy to improve computer reliability. *IBM journal of research and development*, 6(2):200–209, 1962.

[Lyo08] Gordon Fyodor Lyon. *Nmap network scanning: The official Nmap project guide to network discovery and security scanning*. Insecure. Com LLC (US), 2008.

[Mat] Mathworks. Call matlab from python. `https://uk.mathworks.com/help/matlab/matlab-engine-for-python.html`. Accessed online 2023-11-15.

[MB08] Leonardo de Moura and Nikolaj Bjørner. Z3: An efficient smt solver. In *International conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 337–340. Springer, 2008.

[MBMS21] Soo-Jin Moon, Yves Bieri, Ruben Martins, and Vyas Sekar. Automatic discovery of evasion attacks against stateful firewalls. *CMU-CyLab-21-001*, 2021.

[MF14] Michael Muckin and Scott C Fitch. A threat-driven approach to cyber security. *Lockheed Martin Corporation*, 2014.

[MFL+13] Alexander M Melin, Erik M Ferragut, Jason A Laska, David L Fugate, and Roger Kisner. A mathematical framework for the analysis of cyber-resilient

control systems. In *2013 6th International Symposium on Resilient Control Systems (ISRCS)*, pages 13–18. IEEE, 2013.

[MGSBL17] Luis Muñoz-González, Daniele Sgandurra, Martín Barrère, and Emil C Lupu. Exact inference techniques for the analysis of bayesian attack graphs. *IEEE Transactions on Dependable and Secure Computing*, 16(2):231–244, 2017.

[MGSPL17] Luis Munoz-Gonzalez, Daniele Sgandurra, Andrea Paudice, and Emil C Lupu. Efficient attack graph analysis through approximate inference. *ACM Transactions on Privacy and Security (TOPS)*, 20(3):1–30, 2017.

[Mit22] Mitre. Blackenergy software s0089 mitre att&ck 2022, 2022.

[MIT23a] MITRE. Common attack pattern enumeration and classification, 2023.

[MIT23b] MITRE. CVE - CVE — cve.mitre.org. `https://cve.mitre.org/`, 2023. [Accessed 07-12-2023].

[MKNS20] Dimitrios Michail, Joris Kinable, Barak Naveh, and John V. Sichi. JGraphT–A Java Library for Graph Data Structures and Algorithms. *ACM Trans. Math. Softw.*, 46(2), May 2020.

[MLS+21] Baoluo Meng, Daniel Larraz, Kit Siu, Abha Moitra, John Interrante, William Smith, Saswata Paul, Daniel Prince, Heber Herencia-Zapana, M Fareed Arif, et al. Verdict: a language and framework for engineering cyber resilient and safe system. *Systems*, 9(1):18, 2021.

[MMSvM20] Isaac Matthews, John Mace, Sadegh Soudjani, and Aad van Moorsel. Cyclic bayesian attack graphs: a systematic computational approach. In *2020 IEEE 19th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*, pages 129–136. IEEE, 2020.

[MSB+15a] Georg Macher, Harald Sporer, Reinhard Berlach, Eric Armengaud, and Christian Kreiner. Sahara: A security-aware hazard and risk analysis method. In

2015 Design, Automation & Test in Europe Conference & Exhibition (DATE), pages 621–624, 2015.

[MSB+15b] Georg Macher, Harald Sporer, Reinhard Berlach, Eric Armengaud, and Christian Kreiner. Sahara: a security-aware hazard and risk analysis method. In *2015 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 621–624. IEEE, 2015.

[MSJ13] Ian Moir, Allan Seabridge, and Malcolm Jukes. Navigation syst. In *Civil Avionics Syst.*, pages 405–447. John Wiley & Sons, Ltd, August 2013.

[MSMY19] Logan O Mailloux, Martin Span, Robert F Mills, and William Young. A top down approach for eliciting systems security requirements for a notional autonomous space system. In *2019 IEEE International Systems Conference (SysCon)*, pages 1–7. IEEE, 2019.

[MSS98] John C Mitchell, Vitaly Shmatikov, and Ulrich Stern. Finite-state analysis of ssl 3.0. In *USENIX Security Symposium*, pages 201–216, 1998.

[Mur01] Kevin Murphy. The bayes net toolbox for matlab. *Computing science and statistics*, 33, 2001.

[Mur14] Kevin P Murphy. Bayes net toolbox for matlab. `https://github.com/bayesnet/bnt`, 2014.

[NIS] NIST. Nvd cvss v3. `https://nvd.nist.gov/vuln-metrics/cvss/v3-calculator`. [Online; accessed 14-November-2023].

[NM15] Arash Nourian and Stuart Madnick. A systems theoretic approach to the security threats in cyber physical systems applied to stuxnet. *IEEE Transactions on Dependable and Secure Computing*, 15(1):2–13, 2015.

[NPKS23] Stefano M Nicoletti, Marijn Peppelman, Christina Kolb, and Mariëlle Stoelinga. Model-based joint analysis of safety and security: Survey and identification of gaps. *Computer Science Review*, 50:100597, 2023.

[NRZ+14] Chris Newcombe, Tim Rath, Fan Zhang, Bogdan Munteanu, Marc Brooker, and Michael Deardeuff. Use of formal methods at amazon web services. `http://research.microsoft.com/en-us/um/people/lamport/tla/formal-methods-amazon.pdf`, 2014. [Online; Accessed 10-April-2023].

[NTP07] Thomas Novak, Albert Treytl, and Peter Palensky. Common approach to functional safety and system security in building automation and control systems. In *2007 IEEE Conference on Emerging Technologies and Factory Automation (EFTA 2007)*, pages 1141–1148. IEEE, 2007.

[OBM06] Xinming Ou, Wayne F Boyer, and Miles A McQueen. A scalable approach to attack graph generation. In *Proceedings of the 13th ACM conference on Computer and communications security*, pages 336–345, 2006.

[OGA+05] Xinming Ou, Sudhakar Govindavajhala, Andrew W Appel, et al. Mulval: A logic-based network security analyzer. In *USENIX security symposium*, volume 8, pages 113–128. Baltimore, MD, 2005.

[OOA06] Michael L Olive, Roy T Oishi, and Stephen Arentz. Commercial aircraft information security-an overview of arinc report 811. In *2006 ieee/aiaa 25TH Digital Avionics Systems Conference*, pages 1–12. IEEE, 2006.

[PC10] Ludovic Piètre-Cambacédès. *Des relations entre sûreté et sécurité*. PhD thesis, Télécom ParisTech, 2010.

[PCB10a] Ludovic Piètre-Cambacédès and Marc Bouissou. Beyond attack trees: dynamic security modeling with boolean logic driven markov processes (bdmp). In *2010 European Dependable Computing Conference*, pages 199–208. IEEE, 2010.

[PCB10b] Ludovic Piètre-Cambacédès and Marc Bouissou. Modeling safety and security interdependencies with bdmp (boolean logic driven markov processes). In *2010 IEEE International Conference on Systems, Man and Cybernetics*, pages 2852–2861. IEEE, 2010.

[PCB13] Ludovic Piètre-Cambacédès and Marc Bouissou. Cross-fertilization between safety and security engineering. *Reliability Engineering & System Safety*, 110:110–126, 2013.

[PCDB11] Ludovic Piètre-Cambacédès, Yann Deflesselle, and Marc Bouissou. Security modeling with bdmp: from theory to implementation. In *2011 Conference on Network and Information Systems Security*, pages 1–8. IEEE, 2011.

[PCKDG22] Riccardo Patriarca, Mikela Chatzimichailidou, Nektarios Karanikas, and Giulio Di Gravio. The past and present of system-theoretic accident model and processes (stamp) and its associated techniques: A scoping review. *Safety science*, 146:105566, 2022.

[PCQ15] Ludovic Pietre-Cambacedes and Edward L Quinn. Iec 62859: towards an international standard on the coordination between safety and cybersecurity for nuclear i&c systems. In *9th International Topical Meeting on Nuclear Plant Instrumentation, Control, and Human Machine Interface Technologies (NPIC&HMIT 2015)*, 2015.

[PCQH13] Ludovic Pietre-Cambacedes, Edward L Quinn, and Leroy Hardin. Cyber security of nuclear instrumentation & control systems: overview of the iec standardization activities. *IFAC Proceedings Volumes*, 46(9):2156–2160, 2013.

[PGM21] Christophe Ponsard, Jeremy Grandclaudon, and Philippe Massonet. A goal-driven approach for the joint deployment of safety and security standards for operators of essential services. *Journal of Software: Evolution and Process*, page e2338, 2021.

[PHNT19] Daniel Patrick Pereira, Celso Hirata, and Simin Nadjm-Tehrani. A stamp-based ontology approach to support safety and security analyses. *Journal of Information Security and Applications*, 47:302–319, 2019.

[PM99] Y Papadopoulos and J A McDermid. The potential for a generic approach to certification of safety critical systems in the transportation sector. *Reliability engineering & system safety*, 63(1):47–66, 1999.

[Pnu77] Amir Pnueli. The temporal logic of programs. In *18th Annual Symposium on Foundations of Computer Science (sfcs 1977)*, pages 46–57. ieee, 1977.

[PTD21] Ehsan Poorhadi, Elena Troubitysna, and György Dán. Formal modelling of the impact of cyber attacks on railway safety. In *International Conference on Computer Safety, Reliability, and Security*, pages 117–127. Springer, 2021.

[RA15] Yves Roudier and Ludovic Apvrille. Sysml-sec: A model driven approach for designing safe and secure systems. In *2015 3rd International Conference on Model-Driven Engineering and Software Development (MODELSWARD)*, pages 655–664. IEEE, 2015.

[RBD16] Antonio Roque, Kevin B Bush, and Christopher Degni. Security is about control: insights from cybernetics. In *Proceedings of the Symposium and Bootcamp on the Science of Security*, pages 17–24. ACM, 2016.

[RO13] Christian Raspotnig and Andreas Opdahl. Comparing risk identification techniques for safety and security requirements. *Journal of systems and software*, 86(4):1124–1151, 2013.

[Ros22] Andrew Ross. 'cyberpartisans' hack belarusian railway to disrupt russian buildup, 2022.

[RPG+21] Ron Ross, Victoria Pillitteri, Richard Graubart, Deborah Bodeau, and Rosalie McQuaid. *Developing cyber-resilient systems: a systems security engineering approach*. National Institute of Standards and Technology (U.S.), December 2021.

[RWW+09] Alastair Ruddle, David Ward, Benjamin Weyl, Sabir Idrees, Yves Roudier, Michael Friedewald, Timo Leimbach, Andreas Fuchs, Sigrid Gürgens, Olaf Hen-

niger, et al. Deliverable d2. 3: Security requirements for automotive on-board networks based on dark-side scenarios. *EVITA project*, 2009.

[SAJ22] Bastien Sultan, Ludovic Apvrille, and Philippe Jaillon. Safety, security and performance assessment of security countermeasures with sysml-sec. In *10th International Conference on Model-Driven Engineering and Software Development*, 2022.

[SCO+18] Nataliya Shevchenko, Timothy A Chick, Paige O'Riordan, Thomas P Scanlon, and Carol Woody. Threat modeling: a summary of available methods. Technical report, Carnegie Mellon University Software Engineering Institute Pittsburgh United ..., 2018.

[SHB06] M Swanson, J Hash, and P Bowen. *Guide for developing security plans for federal information systems.* NIST, 2006.

[Sho08] Adam Shostack. Experiences threat modeling at microsoft. *MODSEC@ MoDELS*, 2008, 2008.

[Sho14a] Adam Shostack. *Threat modeling: Designing for security.* John Wiley & Sons, 2014.

[Sho14b] Adam Shostack. *Threat modeling: Designing for security.* John Wiley & Sons, 2014.

[SKB+08] Kevin Stine, Richard Kissel, William Barker, Annabelle Lee, and Jim Fahlsing. Guide for mapping types of information and information systems to security categories: appendices. Technical report, National Institute of Standards and Technology, 2008.

[SKG21] Ioannis Stellios, Panayiotis Kotzanikolaou, and Christos Grigoriadis. Assessing iot enabled cyber-physical attack paths against critical systems. *Computers & Security*, 107:102316, 2021.

[SKN+21] Marielle Stoelinga, Christina Kolb, Stefano M Nicoletti, Carlos E Budde, and Ernst Moritz Hahn. The marriage between safety and cybersecurity: still practicing. In *International Symposium on Model Checking Software*, pages 3–21. Springer, 2021.

[Slo18] Joe Slowik. Anatomy of an attack: Detecting and defeating crashoverride. *VB2018, October*, 2018.

[SMGY18] Martin Trae Span, Logan O Mailloux, Michael R Grimaila, and William Bill Young. A systems security approach for requirements analysis of complex cyber-physical systems. In *2018 International Conference on Cyber Security and Protection of Digital Services (Cyber Security)*, pages 1–8. IEEE, 2018.

[SMMY18] Martin Span, Logan O Mailloux, Robert F Mills, and William Young. Conceptual systems security requirements analysis: Aerial refueling case study. *IEEE Access*, 6:46668–46682, 2018.

[Spe20] Certification Specifications. Acceptable means of compliance for large aeroplanes cs-25. *European Aviation Safety Agency, Amendment*, 24(10), 2020.

[SQR+19] Andrea Scarinci, Amanda Quilici, Danilo Ribeiro, Felipe Oliveira, Daniel Patrick, and Nancy G Leveson. Requirement generation for highly integrated aircraft systems through stpa: An application. *Journal of Aerospace Information Systems*, 16(1):9–21, 2019.

[SS11] Scott Shane and David E Sanger. Drone crash in iran reveals secret us surveillance effort. *The New York Times*, 7, 2011.

[SSD+17] Gayathri Sugumar, Rajasekar Selvamuthukumaran, Tomislav Dragicevic, Ulrik Nyman, Kim G Larsen, and Frede Blaabjerg. Formal validation of supervisory energy management systems for microgrids. In *IECON 2017-43rd Annual Conference of the IEEE Industrial Electronics Society*, pages 1154–1159. IEEE, 2017.

[SSW94] Konstantinos Sagonas, Terrance Swift, and David S Warren. Xsb as an efficient deductive database engine. *ACM SIGMOD Record*, 23(2):442–453, 1994.

[Sta07] Defence Standard. Standard 00-56 on safety management requirements for defence systems. *Ministry of Defence, Directorate of Standardisation, Kentigern House*, 65, 2007.

[TASJ18] André MH Teixeira, José Araújo, Henrik Sandberg, and Karl Henrik Johansson. Distributed sensor and actuator reconfiguration for fault-tolerant networked control systems. *IEEE Transactions on Control of Network Systems*, 5(4):1517–1528, 2018.

[Ten] Tenable. Nessus. `https://www.tenable.com/products/nessus`. [Online; accessed 10-April-2023].

[TI13] John P Thomas IV. *Extending and automating a systems-theoretic hazard analysis for requirements generation and analysis*. PhD thesis, Massachusetts Institute of Technology, 2013.

[TPSJ12] André Teixeira, Daniel Pérez, Henrik Sandberg, and Karl Henrik Johansson. Attack models and scenarios for networked control systems. In *Proceedings of the 1st international conference on High Confidence Networked Systems*, pages 55–64. ACM, 2012.

[TSSJ15] Andre Teixeira, Kin Cheong Sou, Henrik Sandberg, and Karl Henrik Johansson. Secure control systems: A quantitative risk management approach. *IEEE Control Systems*, 35(1):24–45, 2015.

[UM15] Tony UcedaVelez and Marco M Morana. *Risk Centric Threat Modeling: Process for Attack Simulation and Threat Analysis*. John Wiley & Sons, 2015.

[Uni] European Union. European railway traffic management system (ertms).

[USC17] David Umsonst, Henrik Sandberg, and Alvaro A Cárdenas. Security analysis of control system anomaly detectors. In *2017 American Control Conference (ACC)*, pages 5500–5506. IEEE, 2017.

[Var01] Moshe Y Vardi. Branching vs. linear time: Final showdown. In *International conference on tools and algorithms for the construction and analysis of systems*, pages 1–22. Springer, 2001.

[Wan04] Lambert Wanninger. Introduction to network rtk. *IAG Working Group*, 4(1):2003–2007, 2004.

[Wei11] Tim Weilkiens. *Systems engineering with SysML/UML: modeling, analysis, design*. Elsevier, 2011.

[WIL+08] Lingyu Wang, Tania Islam, Tao Long, Anoop Singhal, and Sushil Jajodia. An attack graph-based probabilistic security metric. In *IFIP Annual Conference on Data and Applications Security and Privacy*, pages 283–296, 2008.

[WT99] Alma Whitten and J Doug Tygar. Why johnny can't encrypt: A usability evaluation of pgp 5.0. In *USENIX security symposium*, volume 348, pages 169–184, 1999.

[YL13] William Young and Nancy Leveson. Systems thinking for safety and security. In *Proceedings of the 29th Annual Computer Security Applications Conference*, pages 1–8, 2013.

[YP17] William Young and Reed Porada. System-theoretic process analysis for security (stpa-sec): Cyber security and stpa. In *2017 STAMP Conference*, 2017.

[ZDLW21] Changxiao Zhao, Lei Dong, Hao Li, and Peng Wang. Safety assessment of the reconfigurable integrated modular avionics based on stpa. *International Journal of Aerospace Engineering*, 2021, 2021.

[Zen22] Kengo Zenitani. Attack graph analysis: an explanatory guide. *Computers & Security*, page 103081, 2022.