DOI: http://dx.doi.org/10.14236/ewic/ICS2015.4

A Model-based Approach to Interdependency between Safety and Security in ICS

Tingting Li
Institute for Security Science and Technology
Imperial College London
London, UK
tingting.li@imperial.ac.uk

Chris Hankin
Institute for Security Science and Technology
Imperial College London
London, UK
c.hankin@imperial.ac.uk

Wide use of modern ICT technologies brings not only communication efficiency, but also security vulnerabilities into industrial control systems. Traditional physically-isolated systems are now required to take cyber security into consideration, which might also lead to system failures. However, integrating security and safety analysis has always been a challenging issue and the various interdependencies between them make it even more difficult, because they might mutually enhance, or undermine. The paper proposes an integrating framework to (i) formalise the desired and undesired properties to be safe(unsafe) or secure(insecure), including the dependencies between them, (ii) evaluate if a query state reaches a safe(unsafe) or secure(insecure) state, and further quantify how safe or secure the state is. In this way, we can accurately capture the benign and harmful relations between safety and security, particularly detecting and measuring conflicting impacts on them. Finally, this framework is implemented by answer set programming to enable automatic evaluation, which is demonstrated by a case study on pipeline transportation.

Keywords: safety; security; answer set programming; interdependency between safety and security

1. INTRODUCTION

Both safety and security are paramount criteria to evaluate the reliability of an industrial control system (ICS). Safety aims to protect systems from *accidental* failures and hazards, such as fire, flood, vehicle crashes, chemical explosions. Security, however, targets for *intentional* cyber attacks originated from malicious sources, which in particular deals with integrity, availability and confidentially of a system. A referential framework is proposed by Pièrre-Cambacédès and Chaudet (2010) to distinguish security and safety in terms of origin and consequence. Specifically, safety addresses accidents that may damage the environment, while security concerns about malicious attacks that may harm both the system and the environment.

Traditionally, ICS were physically isolated. To increase interconnectivity and enable remote control, modern ICS are not closed systems any more, which makes them vulnerable to external cyber attacks. Thus it becomes important to consider safety and security of ICS in an integrated framework. Therefore, a number of researchers suggest that it is necessary to combine security and safety requirements (Bloomfield et al. 2013; Kriaa et al. 2015; Novak et al. 2007). However, it has never been a trivial task, because of various interdependencies between them (Pièrre-Cambacédès and Bouissou 2010): (i)*Conditional dependency*: safety relies on the fulfillment of security requirements or vice-versa, (ii)*Mutual reinforcement*: safety enhances security, or vice-versa, (iii)*Antagonism*: safety and security requirements conflict with each other, and (iv)*Independence*.

In this paper, we propose an integrated mechanism to evaluate if a particular scenario of a system is safe(secure) or not, and more importantly the mechanism allows for the various interdependencies between safety and security. We first need a way to formally represent the scenarios. We describe these scenarios by *states* of a system. A state consists of states of components (e.g. sensors, actuators, control units), system configurations (e.g. network connectivity), consequences of cyber attacks (e.g. a connection is blocked) and countermeasures from security and safety. Such state-based modeling enables us to not only analyse how states of components/subsystems contribute to the overall safety of a system, but also capture the outcomes of cyber attacks. Rather than addressing broader types of consequences of cyber attacks, we focus on the ones which can sabotage the operational aspects

of ICS. This motivation exactly aligns with the need of integrating security and safety analysis for ICS, because in terms of the traditional goals of security, the integrity and confidentiality are less important than availability in this case (Langer 2012).

The main contributions of this paper are threefold: (i) *state formulae* are defined to declaratively formalise the desired and undesired properties of secure and safe states. The conditional dependency between them is also taken into account, by which a security property could be a condition to reach a safe state, or vice-versa. (ii) we provide both a qualitative and a quantitative evaluation mechanisms, in which the former is able to verify if a query state reaches a safe(secure) or unsafe(insecure) state subject to specified state formulae, while the latter further quantifies the safety and security of the state; (iii) based on the two evaluation mechanisms, we are able to accurately analyse the four different types of interdependencies between security and safety. In particular, we can detect *conflicting states* (i.e. states that are safe but not secure, or secure but not safe), and also identify and measure *conflicting effects* of a countermeasure (implemented or to implement) on them, both of which are rarely addressed by existing work. Moreover, the proposed framework is fully implemented for automatic evaluation by a declarative logic programming paradigm – *answer set programming*, which is then demonstrated by a case study on pipeline transportation.

We employ a hypothetical case study about controlling a pipeline transportation by a simplified industrial control system to demonstrate the proposed evaluation approach. The case study is originally presented by Kriaa et al. (2014) to illustrate the conditional dependency between safety and security. We also use it for the same purpose, but more importantly we extend the case study with an extra new scenario about side-effects of patching vulnerabilities to detect conflicting impact on security and safety.

Fig.1(a) depicts the basic configuration of the pipeline. It deploys two *sensors* (S1 and S2) at different parts of the pipe to monitor the pressure. There is a *pump* to accelerate the speed of fluid and a *valve* to obstruct fluid. The pump is controlled by a *Remote Terminal Unit (RTU)* (labelled as RTU1) while the valve is controlled by RTU2. Both RTUs receive commands from a *Master Control Centre*. The primary safety requirement is that both the pump and the valve must be switched off when a sensor collects an abnormal reading. This example helps us to form interesting states of a system by multiple states of components, consequence of cyber attacks and effect of countermeasures. More details will be discussed in Section 5.2.

We start with the specification of state formulae in Section 2, which is followed by two evaluation mechanisms in Section 3. Various interdependencies between safety and security are discussed in Section 4. An implementation by answer set programming is presented, with a case study in Section 5. The paper concludes with discussing related work in Section 6 and future directions in Section 7.

2. FORMALISATION OF STATE EVALUATION

In this section, we first give a formal representation of the *states* of a system. As explained in the preceding section, a state consists of states of components and subsystems, outcome of cyber attacks, impact of security and safety countermeasures. A state is formally expressed by a set of ground atoms. Each atom captures an aspect or a component of a system. Variables in atoms can be instantiated by values from corresponding categories. All atoms for describing states of a system are from a finite set $\mathcal{A}$.

Definition 1 (States) *Let S be a particular state of a system, expressed by a set of ground atomic formula (or ground atoms) from a finite alphabet $\mathcal{A}$: $S = \{at | at \in \mathcal{A}\}$, or $S \in 2^{\mathcal{A}}$.*

As $\mathcal{A}$ is bounded, all the states can be represented by $2^{\mathcal{A}}$, which is thus finite. States label different views of a system we want to evaluate. We can further select some states of particular interest for security and safety. These states are captured by *state formulae*. A state formula ϕ serves as an indicator to verify if a state S is safe(unsafe) or secure(insecure). A formula is expressed by a combination of: (i) ground atoms from $\mathcal{A}$, (ii) negated ground atoms from $\neg\mathcal{A}$ and (iii) three-valued functions $f^x[\![X]\!]$ [1].

Definition 2 (State Formulae) *: Let ϕ be a state formula, $\mathcal{A}$ a finite alphabet, $f^x[\![X]\!]$ three-valued functions:*

$$\phi = \left\{ at \;\middle|\; \begin{array}{ll} at \in \mathcal{A} & \vee \\ at \in \neg\mathcal{A} & \vee \\ at = f^x[\![X]\!] & \vee \\ at = f^{\neg x}[\![X]\!] & \vee \end{array} \right\}$$

*Sets of formulae are defined for specific purposes: Φ_a is a set of **safe** state formulae, Ψ_a is a set of **unsafe** state formulae, and $\Phi_a \cap \Psi_a = \emptyset$. Φ_e is a set of **secure** state formulae, while Ψ_e is a set of **insecure** state formulae, and $\Phi_e \cap \Psi_e = \emptyset$.*

When describing a state being safe (resp. secure) or unsafe (resp. insecure), one may express which properties are expected to be satisfied by the state, and which properties are not expected. Therefore, a state formula is comprised of ground atoms (for

[1] these three-valued evaluation functions for absolute security and safety are defined in Section3.1 shortly.

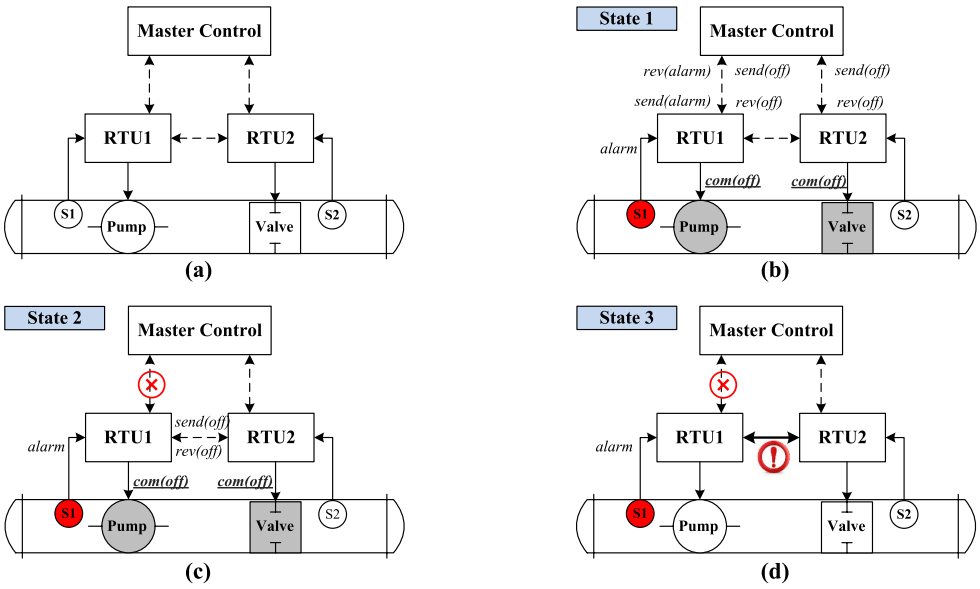

***Figure 1: (a)**: basic configuration. **(b)**: primary protection. **(c)**: reflex action recovers failure of primary protection due to jammed connections. **(d)**: reflex action also fails due to accidental blocking of a security patch.*

desired properties), and also negated ground atoms (for undesired properties). A state formula might also include three-valued evaluation functions, which are used to evaluate absolute safety and security. These embedded functions in a state formula allows for the conditional dependency between safety and security. A safe state requires some security properties to be satisfied, so we could have a nested security evaluation function $f^e[\![X]\!]$ as part of the safe state formula. For simplicity, we assume that safe and unsafe state sets are disjoint, and it also applies to secure and insecure states. In this paper, we adopt the convention that all safety-related notations are labeled with "a", whereas all security-related ones are labeled with "e". For instance, a safe set is Φ_a and a secure set is Φ_e. We then continue to present how a state matches a state formula, and in turn to decide whether the state is safe (secure) or not.

Definition 3 (State Evaluation) *A state S, formed by atoms from alphabet $\mathcal{A}$, satisfies a state formula ϕ, denoted by $S \models \phi$, such that: $S \models \phi \Leftarrow \forall at \in \phi \cdot S \models at$, where*

$$S \models at \Leftarrow \begin{cases} at \in \mathcal{A}, & at \in S & \vee \\ at \in \neg\mathcal{A}, & at \notin S & \vee \\ at = f^x[\![X]\!], & f^x[\![X]\!](S) = \mathtt{t} & \vee \\ at = f^{\neg x}[\![X]\!], & f^x[\![X]\!](S) = \mathtt{f} & \vee \end{cases}$$

Intuitively, to satisfy a state formula ϕ, all positive atoms are required to present whereas all negated atoms absent in a state S. The nested functions in ϕ are required to be *true* (t) when $f^x[\![X]\!]$ is given, or false (f) when $f^{\neg x}[\![X]\!]$ is given in ϕ. An example of state evaluation is shown in Fig.2. We have a safe set Φ_a with one formula ϕ_1, and a secure set Φ_e with one formula ϕ_2. In the example we leave the unsafe set Ψ_a and insecure set Ψ_e empty. The safe formula ϕ_1 states that it is safe to keep the pump and the valve running, when the system is *secure* subject to the security evaluation function $f^e[\![\Phi_e, \Psi_e]\!]$). As specified by the only secure formula ϕ_2, the system is secure if all the three connections amongst *mtu*, *rtu1* and *rtu2* are not jammed due to cyber attacks. Thus, the safety of a state depends on the security of the state. A query state S is given to describe a scenario and evaluated.

Example: given a secure state set Φ_e, a safe state set Φ_a, an insecure state set Ψ_e and an unsafe state set Ψ_a as below:
$\Phi_a = \{\phi_1\}$ $\quad\quad$ $\Psi_a = \emptyset$
$\Phi_e = \{\phi_2\}$ $\quad\quad$ $\Psi_e = \emptyset$

ϕ_1 = { *pump(on), valve(on),* $f^e[\![\Phi_e, \Psi_e]\!]$ }
ϕ_2 = { ¬*jammed(mtu, rtu1),* ¬*jammed(mtu, rtu2),* ¬*jammed(rtu1, rtu2)* }
A query state is given as below:
S = {*pump(on), valve(on), sensor(alarm)*}
According to Def. 3, S satisfies ϕ_1 and ϕ_2.

***Figure 2:** Example: evaluation with nested functions*

3. EVALUATION FOR SAFETY AND SECURITY

In this section, we provide two different ways to evaluate safety and security. The first one qualifies a state to be *absolutely* safe or secure in terms of the specified formulae. The evaluation is supported by two three-valued functions $f^a[\![\Phi_a, \Psi_a]\!](S)$ and $f^e[\![\Phi_e, \Psi_e]\!](S)$ over a query state S. More details are discussed in Section 3.1. A quantitative evaluation is provided in Section 3.2, by which we can obtain a

metric (namely safety and security degree) to assess how safe or secure a state is.

3.1. Absolute Safety and Security

A safe state set Φ_a and an unsafe state set Ψ_a are specified to enumerate states that are considered safe and unsafe respectively. A three-valued function is defined: $f^a[\![\Phi_a, \Psi_a]\!](S) \rightarrow \{\mathtt{t}, \mathtt{f}, \mathtt{u}\}$, to evaluate a query state S by the state sets Φ_a and Ψ_a, where $\mathtt{t}$ corresponds to *safe*, $\mathtt{f}$ indicates *unsafe*, and for the sake of completeness, $\mathtt{u}$ is assigned to all other indeterminate states. The interpretation of the evaluation mechanism is : (i) a state is absolutely *safe* if the state satisfies *at least* one formula in the safe set Φ_a, and matches *none* of the formulae specified in the unsafe set Ψ_a; (ii) a state is absolutely *unsafe* if the state satisfies *at least* one formula in the unsafe set Ψ_a, regardless of its satisfaction of the safe set; (iii) a state is considered as *unspecified* if we fail to verify that it is safe or unsafe subject to the specified formulae.

$$f^a[\![\Phi_a, \Psi_a]\!](S) = \begin{cases} \mathtt{t} & \exists \phi \in \Phi_a, S \models \phi \quad \wedge \\ & \nexists \psi \in \Psi_a, S \models \psi \\ \mathtt{f} & \exists \psi \in \Psi_a, S \models \psi \\ \mathtt{u} & \text{otherwise} \end{cases} \tag{1}$$

As we made the assumption that Φ_a and Ψ_a are disjoint, the function always returns an unique value and it is impossible to verify a state to be both *safe* and *unsafe*. We would argue that such an assumption is necessary for this paper, as we focus on the inter-relationships between security and safety, rather than the intra-formulae relationships within safety set or security set. In the implementation Section 5, we have certain constraint rules to avoid such conflicting formulaes.

Similarly, $f^e[\![\Phi_e, \Psi_e]\!](S)$ is defined to verify if a state is absolutely secure, or insecure or unknown, subject to a secure state set Φ_e and an insecure state set Ψ_e:

$$f^e[\![\Phi_e, \Psi_e]\!](S) = \begin{cases} \mathtt{t} & \exists \phi \in \Phi_e, S \models \phi \quad \wedge \\ & \nexists \psi \in \Psi_e, S \models \psi \\ \mathtt{f} & \exists \psi \in \Psi_e, S \models \psi \\ \mathtt{u} & \text{otherwise} \end{cases} \tag{2}$$

3.2. Safety and Security Degree

Absolute evaluation provides an easy way to capture dependency between safety and security and qualify states. However, it may not be adequate to accurately express the mutual effects between security and safety. A question worth thinking about is whether we can gain more safety whilst we gain more security, or whether we have to sacrifice some security to gain more safety? Before we try to address these questions, we first need a way to represent quantitatively how much we gain or sacrifice in terms of security and safety. Therefore, in this section, we introduce the notion of *security degree* and *safety degree*, and how they can be calculated against a query state S.

A safe state set Φ_a and an unsafe state set Ψ_a are given to evaluate a state S is *safe* to certain degree $\Delta a = a - \tilde{a}$, and Δa is the *safety degree* of S. The two components a and $\tilde{a}$ are obtained by a function $f^a[\Phi_a, \Psi_a](S) \rightarrow (a, \tilde{a})$, where:

$$\begin{cases} a = |\Phi'_a|\,, & \forall \phi \in \Phi'_a, S \models \phi \quad \wedge \\ & \operatorname{argmax}\{|\Phi'_a| : \Phi'_a \subseteq \Phi_a\} \\ \tilde{a} = |\Psi'_a|\,, & \forall \psi \in \Psi'_a, S \models \psi \quad \wedge \\ & \operatorname{argmax}\{|\Psi'_a| : \Psi'_a \subseteq \Psi_a\} \end{cases} \tag{3}$$

From the definition, the safety degree is calculated by the maximal number a of safe formulae the state satisfies, minus the maximal number $\tilde{a}$ of unsafe formulae the state satisfies. Thus, a state would have higher safety degree if it supports more safe formulae and less unsafe formulae.

The similar way is adopted to obtain security degree. A secure state set Φ_e and an insecure state set Ψ_e are given to evaluate a state S is *secure* to certain degree $\Delta e = e - \tilde{e}$, where:

$$\begin{cases} e = |\Phi'_e|\,, & \forall \phi \in \Phi'_e, S \models \phi \quad \wedge \\ & \operatorname{argmax}\{|\Phi'_e| : \Phi'_e \subseteq \Phi_e\} \\ \tilde{e} = |\Psi'_e|\,, & \forall \psi \in \Psi'_e, S \models \psi \quad \wedge \\ & \operatorname{argmax}\{|\Psi'_e| : \Psi'_e \subseteq \Psi_e\} \end{cases} \tag{4}$$

4. RELATIONS OF SAFETY AND SECURITY

So far we discussed separate evaluation of security and safety, and in this section we combine the two evaluations to accurately capture the interdependencies between safety and security. As discussed in Section 1, there are various relations between security and safety: *conditional dependency* (Section 4.1), *mutual reinforcement* (Section 4.2)and *antagonism* (Section 4.3). In the rest of this section, we discuss each of them in turn.

4.1. Conditional Dependency

When we define the concept of state formulae in Section 2, a component of a formula could be a nested evaluation function, such as the absolute evaluation functions (c.f. Section 3.1). It allows a safe state formula to require the fulfilment of a security condition, or vice-versa. Such conditional dependency might slightly increase the complexity of the evaluation procedure. We take safety evaluation $f^a[\![\Phi_a, \Psi_a]\!](S)$ as an example to illustrate the procedures in Algorithm 1.

To evaluate if a state is safe or not, we need to solve the function $f^a[\![\Phi_a, \Psi_a]\!](S)$. Firstly we look for any unsafe state formula that can be satisfied by the state S (line 2 to 10). When we check if the state S matches a formula ϕ, we firstly resolve

Algorithm 1 Evaluation of Conditional Dependency

Input: a query state S, a safe state set Φ_a, an unsafe state set Ψ_a, a secure state set Φ_e, an insecure state set Ψ_e, a subset $X \subseteq \Phi_e \cup \Psi_e$.
Compute: $f^a[\![\Phi_a, \Psi_a]\!](S) \rightarrow res \in \{\mathtt{t}, \mathtt{f}, \mathtt{u}\}$
Output: res

1: **procedure** EVALUATE($f^a[\![\Phi_a, \Psi_a]\!](S)$)
2: **for all** $\psi \in \Psi_a$ **do** ▷ check for any unsafe state
3: **if** $f^e[\![X]\!] \vee \neg f^{\neg e}[\![X]\!] \in \psi$ **then**
4: **evaluate** $f^e[\![X]\!](S) \rightarrow \{\mathtt{t}, \mathtt{f}, \mathtt{u}\}$
5: **end if**
6: **if** $S \models \psi$ **then** ▷ an unsafe state matches
7: $res \leftarrow \mathtt{f}$
8: **return** res
9: **end if**
10: **end for** ▷ no unsafe state found
11: **for all** $\phi \in \Phi_a$ **do** ▷ check for any safe state
12: **if** $f^e[\![X]\!] \vee \neg f^{\neg e}[\![X]\!] \in \phi$ **then**
13: **evaluate** $f^e[\![X]\!](S) \rightarrow \{\mathtt{t}, \mathtt{f}, \mathtt{u}\}$
14: **end if**
15: **if** $S \models \phi$ **then** ▷ a safe state matches
16: $res \leftarrow \mathtt{t}$
17: **return** res
18: **end if**
19: **end for**
20: $res \leftarrow \mathtt{u}$ ▷ indeterminate state
21: **return** res
22: **end procedure**

any nested functions $f^e[\![X]\!]$ or $f^{\neg e}[\![X]\!]$ enclosed in ϕ with regard to Def.3 (line 3 to 5). If any unsafe match is found (line 6), the whole procedure terminates and returns $\mathtt{f}$, i.e. the state is identified as unsafe, otherwise the whole procedure continues to check against safe states (as listed on line 11 to 19). Again, all nested functions are evaluated firstly and then if any safe state is found, the whole procedure terminates and returns $\mathtt{t}$, indicating the query state is safe. Finally, if the whole procedure fails to find any match in either unsafe or safe states, then the whole procedure returns $\mathtt{u}$ and terminates (line 20). Continuing with the example in Fig.2, we can illustrate absolute evaluation with conditional dependency. As the query S satisfies ϕ_1 and ϕ_2, we can determine that the state is absolutely safe in terms of the given formulae.

4.2. Mutual Reinforcement

With the help of security degree and safety degree introduced in Section 3.2, we can now accurately measure the mutual effects between security and safety. It would be beneficial to know how security and safety are affected if we make any modification to the system configuration, or implement new countermeasures. For instance, if a new security countermeasure is deployed, the security degree is very likely to increase, but the safety degree may decrease. We denote a variant of S by S'.

Definition 4 (Mutual Reinforcement) *Given an improved state S' of S, $S \subset S'$. $S' \gg S$ expresses S' reinforces safety without undermining security, or vice-versa:*

- $\Delta a' > \Delta a \ \wedge \ \Delta e' \geqslant \Delta e$, ***or***
- $\Delta e' > \Delta e \ \wedge \ \Delta a' \geqslant \Delta$

where $\Delta a'$, Δa, Δe and $\Delta e'$ are calculated by metric evaluation function (c.f. Eq.3 and 4) over S and S'.

Therefore, $S' \gg S$ indicates that the security is improved without undermining safety, or the safety is enhanced without sacrificing security.

4.3. Antagonism

Conflicting requirements between security and safety are always important to address for industrial control systems. We need to guarantee that increasing security (resp. safety) of a system is not damaging the safety (resp. security) of the system. However, in most cases, we may have to find the most optimal trade-off between security and safety. Here in this paper, as we combine both safety and security requirements into the process of evaluation, we are able to detect the existence of conflicting situations between safety and security: (i) the first one is based on the absolute evaluation, to detect *conflicting states*, as discussed in Section 4.3.1, and (ii) the second one employs the metric evaluation, to find *conflicting effects* as presented in Section 4.3.2.

4.3.1. Finding Conflicting States

Both of the two absolute evaluation functions $f^e[\![\Phi_e, \Psi_e]\!](S)$ and $f^a[\![\Phi_a, \Psi_a]\!](S)$ produce results from a three-valued set $\{\mathtt{t}, \mathtt{f}, \mathtt{u}\}$. Each value indicates different characteristics of a state, i.e. $\mathtt{t}$ for safe(resp. secure), $\mathtt{f}$ for unsafe(resp. insecure) and $\mathtt{u}$ for unknown. Once we combine the results from both functions, we can obtain 9 different combinations of truth values. The conflicting states are indicated by $[\mathtt{t}, \mathtt{f}]$ and $[\mathtt{f}, \mathtt{t}]$.

Definition 5 (Conflicting States) *: A state S is identified as a conflicting state with regards to sets of state formulae Φ_a, Ψ_a, Φ_e and Ψ_e, iff:*

- $f^e[\![\Phi_e, \Psi_e]\!](S) = \mathtt{t} \wedge f^a[\![\Phi_a, \Psi_a]\!](S) = \mathtt{f}$, ***or***
- $f^e[\![\Phi_e, \Psi_e]\!](S) = \mathtt{f} \wedge f^a[\![\Phi_a, \Psi_a]\!](S) = \mathtt{t}$.

4.3.2. Finding Conflicting Effects

This detection mechanism is based on security degree and safety degree, as defined in Section 3.2. It addresses the earlier requirements about system modifications and new countermeasures may bring conflicting effects to security and safety. We use S' to denote a variant of the original state S, and calculate their corresponding security and safety degree for comparison. The objective is to find out

the modifications by which only security or safety increases but the other decreases.

Definition 6 (Conflicting Effects) *: Given an improved state S' of S, $S \subset S'$. S' has conflicting effects to security and safety with regards to sets of state formulae Φ_a, Ψ_a, Φ_e, Ψ_e, and its original form S iff:*

- $\Delta a > \Delta a' \ \wedge \ \Delta e < \Delta e'$*,* ***or***
- $\Delta a < \Delta a' \ \wedge \ \Delta e > \Delta e'$

where $\Delta a'$, Δa, Δe and $\Delta e'$ are calculated by metric evaluation function $f^a[\Phi_a, \Psi_a](S)$ and $f^e[\Phi_e, \Psi_e](S)$(c.f. Eq.3 and 4). Such relation is denoted by $S \top S'$.

We can make various modifications to a state S, including network configurations and new security or safety countermeasures. However, not all of them are beneficial for both security and safety simultaneously. We provides a way to measure the effects, and to detect conflicting effects by comparing the measurements. We believe it would help with evaluating the effects of implementing a countermeasure and making optimal decisions on the trade-off between safety and security.

5. IMPLEMENTATION BY ASP

Answer Set Programming (ASP) is a method of declarative logic programming under the answer set semantics. Thanks to its declarative paradigm, it has the advantage of describing the expectations and constraints of solutions rather than designing an algorithm to find solutions to a problem. There are various languages and efficient solvers for ASP. For this paper, we use the language $AnsProlog$ (Baral 2003) and the solver CLINGO (Gebser et al. 2011), which are currently the most widely used ones.

Basic elements of a normal ASP program are atoms from a finite set of atoms $\mathcal{A}$. Each atom at has a *predicate* symbol applied to a number of *variables*, e.g. $\mathtt{p(X,Y)}$. These variables `X` and `Y` can be further grounded by specific constant values. The resulting ground atoms can be assigned either *true* or *false*. *Literals* are then atoms at or negated atoms. ASP uses *negation as failure* to compute the negation of an atom, i.e. $\mathbf{not}\ \mathtt{at}$ is true if there is no evidence to prove `at` in a program. Therefore, in this paper, the absence of at derives $\mathbf{not}\ a$. The language $AnsProlog$ follows the conventions that variables start with capitals, while values and predicates begin with lower-case letters.

A normal program P is a conjunction of rules r with the general form: $a :- b_1, ..., b_m, \mathrm{not}\ c_1, ..., \mathrm{not}\ c_n.$ where $r \in P$, and a, b_i and c_j are atoms. Intuitively, this means *if all atoms b_i are known/true and no atom c_j is known/true, then a must be known/true*. a is the head part of the rule, while b_i and $\mathrm{not}\ c_j$ are body parts of the rule. A rule is a *fact rule* if the body part of the rule is empty, or a *constraint rule* if the head part of the rule is empty, indicating that no solution satisfies the body. A syntactic suger *aggregates* is provided by the solver to count or limit the number of particular atoms. An aggregates statement is of the form: $lower\ \#count\{L_1, \ldots, L_n\}\ upper$, where $lower$ and $upper$ are the lower and upper bound of the number of literals L_i, i.e. $lower \leqslant n \leqslant upper$. Sometimes the keyword $count$ could be omitted.

The semantics of an ASP program is defined in terms of *answer sets*, i.e. assignments of true and false to all atoms in the program that satisfy the rules in a *minimal* and *consistent* fashion. A program may have zero or more answer sets. Each answer set is a solution to the problem specified in the program.

5.1. Evaluation by Answer Set Programming

5.1.1. State Evaluation

There are two basic atoms used to represent states and state formulae: (i) $\mathtt{state(S, atom)}$ denotes that an `atom` is part of the state `S`, and (ii) $\mathtt{match(S, f_n)}$ expresses a state `S` satisfies a formula $\mathtt{f_n}$. As introduced in Def.3, there are four different types of components to constitute a formula. For example, we have the following four formulae: $\phi_{f_1} = \{at_1\}$, $\phi_{f_2} = \{\neg at_1\}$, $\phi_{f_3} = \{at_2, f^x[\![\phi_{f_1}]\!]\}$ and $\phi_{f_4} = \{at_2, f^{\neg x}[\![\phi_{f_1}]\!]\}$. Then we encode them into ASP rules:

```
match(S,f1) :- state(S,at1).
match(S,f2) :- not state(S,at1).
match(S,f3) :- state(S,at2),match(S,f1).
match(S,f4) :- state(S,at2),not match(S,f1).
```

It is worth noting that $\mathtt{match(S, f3)}$ requires satisfaction of $\mathtt{match(S, f1)}$, which is formalised by $f^x[\![\phi_{f_1}]\!]$ in ϕ_{f_3}. We then assign these formulae into different formula sets with specific purpose:

```
   sec_set(S,f1) :- formula(f1).
 insec_set(S,f2) :- formula(f2).
  safe_set(S,f3) :- formula(f3).
usnafe_set(S,f4) :- formula(f4).
```

A query state S_1 is given by: $S_1 = \{at_1, at_2\}$, which can be translated into ASP facts as below:

```
stateID(s1).
state(s1, at1).   state(s1, at2).
```

Based on the ASP rules and facts above, the ASP solver produces an answer set including facts: $\mathtt{match(s1, f1)}$ and $\mathtt{match(s1, f3)}$, indicating that the state S_1 satisfies the formula ϕ_{f_1} and ϕ_{f_3}.

5.1.2. Absolute Evaluation
Absolution evaluation for safety and security is proposed as a qualitative means to decide if a query state is safe or not, secure or not, or indeterminate. Following the operational procedures outlined in Algorithm 1, we implement the following ASP program to achieve the evaluation of safety:

```
unsafe(S) :- match(S, F), unsafe_set(F),
             formula(F).
  safe(S) :- match(S, F), safe_set(F),
             not unsafe(S), formula(F).
 xsafe(S) :- not safe(S), not unsafe(S),
             stateID(S).
```

The unsafe(S), safe(S) and xsafe(S) correspond to the three values $\{\mathtt{t}, \mathtt{f}, \mathtt{u}\}$ produced by the safety evaluation $f^a[\![X]\!](S)$. Likewise, the results of security function $f^e[\![X]\!](S)$ are given by insecure(S), secure(S) and xsecure(S) respectively, and implemented in the same way.

5.1.3. Security and Safety Degree
In this section we present the implementation of calculating safety degree and security degree.

```
safe_degree(S,D) :-
    D=#count{match(S,F):safe_set(F)}.
unsafe_degree(S,D) :-
    D=#count{match(S, F):unsafe_set(F)}.
safety(S, D) :- D=D1-D2,
    safe_degree(S, D1), unsafe_degree(S, D2),
```

safe_degree(S, D) and unsafe_degree(S, D) collect respectively the number (D) of safe and unsafe formulae matched by the state S. Thus the safety degree of S is calculated by subtracting the number of matched unsafe formulae from the number of matched safe formulae, which is finally carried by safety(S, D). The security degree is obtained in the same way, and encoded by security(S, D) .

5.1.4. Conflict Detection
Having implemented the two evaluation mechanisms, we can now detect conflicting states and conflicting effects (as defined in Def.5 and 6).

```
conflict_state(S) :- secure(S), unsafe(S).
conflict_state(S) :- safe(S), insecure(S).

conflict_effct(S1, S2) :-
    safety(S1, A1), safety(S2, A2),
    security(S1, E1), security(S2, E2),
    A1 > A2, E1 < E2,  stateID(S1;S2).
conflict_effct(S2, S1) :-
    safety(S1, A1),   safety(S2, A2),
    security(S1, E1),  security(S2, E2),
    A1 < A2, E1 > E2,  stateID(S1;S2).
```

The detection of conflicting states is to find states which are secure but not safe, or safe but not secure. The atom conflict_state(S) encodes such a state. Ideally, an *improved* state should outperform the original state in terms of both security and safety, rather than earning one of them by sacrificing the other. The atom conflict_effect(S1, S2) is derived to alarm the latter situations between S1 and S2.

5.2. Case Study: Pipeline Transportation

As we described in Section 1, we use an example about pipeline transportation (Fig.1) to illustrate the proposed evaluation approach. Particularly, possible conflicting impact of a proposed countermeasure on security and safety of a system can be detected and measured. We then focus on two response mechanisms to restore the safety.**Primary Protection** in Fig.1(b): (i) an alarm signal is received by RTU1, (ii) RTU1 forwards it to Master control centre, (iii) Master replies with off commands to both RTU1 and RTU2 to stop the pump and close the valve. Such primary protection heavily depends on reliable connections between Master and the RTUs. Therefore, *RTU Reflex Action* is also deployed to maintain the safety of the system, particularly when the connections between Master and RTUs are jammed by intentional cyber attacks.

RTU Reflex Action in Fig.1(c): (i) an alarm is received by RTU1, (ii) RTU1 immediately sends a command to stop the pump without waiting for instructions from the Master, (iii) RTU1 sends off commands to other RTUs without waiting for the Master. Thus, RTU2 closes the valve successfully.

In Fig.3, we list corresponding ASP rules for Primary Protection (line 1-5) and Reflex Action (line 6-8). The rule on line 3-5 states that successful delivery of a message (e.g. alarm, commands) depends on (i) an available connection between Host1 and Host2, state(S, connected(Host1, Host2)), and (ii) the connection is not attacked, **not** state(S, jammed(Host1, Host2)). The two conditions are also required between RTU1 and RTU2 to deploy reflex actions (line 7-8). Line 9 specifies that a connection is valid if there is no accidental blocking to it acci_block(Host1, Host2), which could be an unfortunate consequence of security patching (line 10). Such an accident is very likely to harm the safety of the pipeline by disabling reflex action. We will use this example to analyse the conflicting effects shortly. Finally, The two constrains rules on line 11-12 make sure it is impossible for a formula to be a safe(resp. secure) and an unsafe (resp. insecure) formula at the same time.

We summarise the safety and security requirements of the case as state formulae in Fig.4. Three *safety* formulae are given on line 1-3, and satisfying any of them keeps the system safe (as in the example

Primary Protection:

```
1  state(S, send(RTU, master, alarm)) :- state(S, receive(RTU, alarm)), rtu(RTU).
2    state(S, send(master, RTU, off)) :- state(S, receive(master, alarm)), rtu(RTU).
3   state(S, receive(Host2, Message)) :- state(S, send(Host1, Host2, Message)),
4                                        state(S, connected(Host1, Host2)),
5                                        not state(S, jammed(Host1, Host2)).
```

Reflex Actions: :

```
6  state(S,command(Actuator,off)):-state(S, receive(Rtu1, alarm)), control(Rtu1, Actuator).
7    state(S,send(Rtu1,Rtu2,off)):-state(S,receive(Rtu1,alarm)),state(S,connected(Rtu1,Rtu2)),
8                                  not state(S, jammed(Rtu1, Rtu2)), rtu(Rtu2;RTU1).
```

Side-effect of patching :

```
9   state(S, connected(Host1, Host2)) :- not state(S, acci_block(Host1, Host2)), stateID(S).
10 state(S, acci_block(Host1, Host2)) :- state(S, patch(Host1, Host2)), stateID(S).
```

Constraint rules :

```
11 :- safe_set(F), unsafe_set(F), formula(F).
12 :- sec_set(F), insec_set(F), formula(F).
```

Figure 3: *Configuration of pipeline in the case study*

Safe State Formulae Φ_a:

```
1 match(S,f1):- state(S,sensor1(normal)).
2 match(S,f2):- state(S,sensor1(alarm)), state(S,command(pump,off)), state(S,command(valve,off)).
3 match(S,f3):- state(S,sensor1(alarm)), match(S, f4) .
```

Secure State Formulae Φ_e:

```
6 match(S, f4) :- 0{state(S,jammed(master,RTU)):rtu(RTU)}0, stateID(S).
```

Insecure State Formulae Ψ_e:

```
8  match(S, f5) :-  not state(S, patch(rtu1, rtu2)), stateID(S).
9  match(S, f6) :-  not state(S, patch(master, rtu1)), stateID(S).
10 match(S, f7) :-  not state(S, patch(master, rtu2)), stateID(S).
```

Figure 4: *State formulae in the case study*

unsafe formula set is empty). We interpret each safe formula as follows: (i) `f1`: `sensor1` collects `normal` reading. (ii) `f2`: in the case of `alarm`, the system is safe if both `pump` and `valve` are switched `off`. (iii) `f3`: in the case of `alarm`, the primary protection can be activated only if the system is secure, i.e. matching the *secure* formula `f4`. As given on line 6, `f4` guarantees secure connections between the `master` and all `RTU`s, i.e. there is exactly `0` jammed connection, supported by an *aggregate* statement. Regarding the security, the three connections (Master and RTU1, Master and RTU2, RTU1 and RTU2) are the primary concerns, and thus we expect they can be protected properly, otherwise the system is insecure. Such security requirements are captured by insecure formulae (`f5`,`f6` and `f7`) on line 8-10 in Fig.4. If any of the connections is not patched **not** $\texttt{state}(\texttt{S}, \texttt{patch}(\texttt{rtu1}, \texttt{rtu2}))$, the system is insecure.

Next we give three query states in ASP, as shown below, to analyse the dependency and conflicts between safety and security. The three states are identified by `s1`, `s2` and `s3`, and describe the scenarios in (b), (c) and (d) of Fig.1 respectively.

```
stateID(s1;s2;s3).
%% ------  state 1 ------ %%
state(s1, pump(on)).    state(s1, valve(on)).
state(s1, sensor1(alarm)).
%% ------  state 2 ------ %%
state(s2, pump(on)).    state(s2, valve(on)).
```

```
state(s2, sensor1(alarm)).
state(s2, jammed(master, rtu1)).
%% ------  state 3 ------ %%
state(s3, pump(on)).    state(s3, valve(on)).
state(s3, sensor1(alarm)).
state(s3, jammed(master, rtu1)).
state(s3, patch(rtu1, rtu2)).
```

All three states are in the case of `alarm`. In state `s1`, the safety can be restored by either primary protection or reflex actions, because none of the connections is jammed. We then break the security formula by attacking the connection to result in $\texttt{jammed}(\texttt{master}, \texttt{rtu1})$ in state `s2`, which leads to unavailability of primary protection, but reflex actions

could still restore the safety. The state s3 describes the connection between the two RTUs is patched to mitigate inter-RTU security vulnerabilities.

According to the state formulae in Fig.4, we then evaluate the three states by using the ASP programs presented in Section 5.1, to find out: (i) whether each state is absolutely safe (resp.secure) or unsafe (resp.insecure), (ii) the safety and security degree of each state, and (iii) any existence of conflicting effects caused by implemented measures. The final result produced by the ASP solver is given below:

```
Answer: 1
match(s1,f2) match(s1,f3) match(s1,f4)
match(s1,f5) match(s1,f6) match(s1,f7)
safety(s1,2) safe(s1)
security(s1,-3) insecure(s1)

match(s2,f2)
match(s2,f5) match(s2,f6) match(s2,f7)
safety(s2,1) safe(s2)
security(s2,-3) insecure(s2)

match(s3,f6) match(s3,f7)
safety(s3,0) security(s3,-2) insecure(s3)
conflict_effct(s2,s3)

SATISFIABLE
```

The result shows that s1 satisfies a set of formulae and it matches the safety formula $\texttt{match}(\texttt{s1},\texttt{f3})$ by satisfying the secure formula $\texttt{match}(\texttt{s1},\texttt{f4})$, and thus the state is safe $\texttt{safe}(\texttt{s1})$ with degree $\texttt{safety}(\texttt{s1},2)$. As there is no information about patching provided in s1, none of the three insecure formulae is avoided, and thus the state is insecure $\texttt{insecure}(\texttt{s1})$ with security degree $\texttt{security}(\texttt{s1},-3)$. Similar evaluation is applied to state s2 where $\texttt{jammed}(\texttt{master},\texttt{rtu1})$ is present. In this case, the safety degree is reduced by one due to the failure to match f3. But the state is still safe $\texttt{safe}(\texttt{s2})$ thanks to the reflex action, which is not effected by the jammed connection. When a security measure $\texttt{patch}(\texttt{rtu1},\texttt{rtu2})$ is applied in s3, which as discussed earlier disables reflex actions, and thus s3 is not safe and the safety degree is reduced to 0. However, its security degree $\texttt{security}(\texttt{s3},-2)$ is actually increased from previous state $\texttt{security}(\texttt{s2},-3)$ due to the patch. Thus, a conflicting effect $\texttt{conflict\_effect}(\texttt{s2},\texttt{s3})$ is detected between s2 and s3 against the patching.

6. RELATED WORK

A survey article by Kriaa et al. (2015) provides a comprehensive summary about existing approaches of combing security and safety for ICS. The authors distinguish the two concepts with regard to the terminology framework by Piètre-Cambacédès and Chaudet (2010). Both concepts deal with risks to prevent systems from failure. However, they have distinct origins and consequences. The survey further classifies existing approaches into generic approaches, and model-based approaches.

Generic approaches mainly focus on the early stage of system design, to provide high-level guidelines for life-cycle or methodological process. These approaches (Aven 2007; Woskowski 2014) may result in an unified framework by merging safety and security requirements into a single (conflict-free) set. However, as argued by Eames and Moffett (1999), such unified frameworks might not be able to provide an accurate analysis at the interaction between security and safety due to the highly global abstraction and unification. In this case, recognition of conflicting requirements or effects is not possible. Therefore, Eames and Moffett (1999) tend to integration approaches. Risk assessments for security and safety operate separately to determine requirements. The interactions between safety and security are achieved by cross-reference from one domain to the other. Conflicts in this case are recognisable. Inspired by this, our approach also follows the integrating paradigm rather than unification, by encoding safety and security requirements in different sets of formulae. Given a query state, safety evaluation and security evaluation are conducted separately, but it is still possible to influence each other with the help of nested functions in state formulae.

Compared with generic approaches, model-based approaches analyse security and safety in detail by means of formal representations and tools (Piètre-Cambacédès and Bouissou 2010; Kriaa et al. 2014; Bloomfield et al. 2013; Sun et al. 2009). Bloomfield et al. (2013) investigate the security impact on safety assessment in the context of critical infrastructure, based on *Claims-Arguments-Evidence* (CAE). Piètre-Cambacédès and Bouissou (2010) employ *Boolean Logic Driven Markov Processes* (BDMP) to model interactions between security and safety. Such an approach offers (i) a tree-like representation to present the hierarchical relations between events and states, (ii) a quantitative analysis based on Markov processes by associating leaf events with estimated probabilities, and (iii) special “trigger” relations are designed to capture the interactions between events and states. This approach is also demonstrated with a case study in Kriaa et al. (2014). The work provides a way to model and analyse the interdependency between security and safety, but it is not clear how conflicting relations between them can be detected and evaluated. In contrast, the work Sun et al. (2009) focuses on the contradictory requirements between security and safety without considering the conditional dependency between them. Based on

this, the proposed approach in this paper provides a more comprehensive and generic mechanism to address the conditional dependency between security and safety, AND also the identification of conflicting states and evaluation of conflicting effects.

7. CONCLUSION AND FUTURE WORK

In this paper, we report the initial achievement on integrating evaluation of security and safety for industrial control systems. We introduce an approach to model and evaluate the various interdependencies between security and safety requirements. By this approach, requirements and expectations from the view of security and safety can be provided separately, which are then encapsulated as state formulae. In particular security conditions can be embedded as part of a safety formula, or vice-versa, which is able to accurately capture the conditional dependency between them. Moreover, we offer two evaluation mechanisms, where the former aims to answer the question if a given state is safe or not, secure or not, whereas the latter further measures how secure/safe the state is. With the help of the two evaluation mechanisms, we enable the detection of conflicting states, in which safety and security cannot be fully satisfied, and also the measurement of the conflicting effects caused by changes of requirements or countermeasures.

There are several promising directions set out for the future work (ii) the natural next step would be conflict resolution between safety and security. We plan to adopt the Belnap four-value logic (Hankin et al. 2009) and D-algebra (Ni et al. 2009) to tackle it. Both of them have been successfully applied to compose access control policies while resolving conflicts that may arise. (ii) we are also interested in possible ways for automatic generation and learning of state formulae from concrete use cases. (iii) currently each state formula is treated equally. In the future, we would assign different weights to distinguish them according to the consequences of violating them.

ACKNOWLEDGEMENT

This work is supported by the EPSRC project RITICS: Trustworthy Industrial Control Systems.

REFERENCES

Aven, T. (2007) A unified framework for risk and vulnerability analysis covering both safety and security. Reliability Eng. Syst. Safety, 92 (6), 745–754.

Baral, C. (2003) *Knowledge representation, reasoning and declarative problem solving*. Cambridge, U.K.: Cambridge University Press.

Bloomfield, R., Netkachova, K., and Stroud, R., (2013) Security-informed safety: If its not secure, its not safe. In: *Software Engineering for Resilient Systems*. Berlin, Germany: Springer, 17–32.

Eames, D. P. and Moffett, J. (1999) The integration of safety and security requirements. In: *Computer Safety, Reliability and Security*. Berlin, Germany: Springer, 468–480.

Gebser, M., Kaminski, R., Kaufmann, B., Ostrowski, M., Schaub, T., and Schneider, M. (2011) Potassco: The Potsdam answer set solving collection. *AI Commun.*, 24 (2), 107–124.

Hankin, C., Nielson, F., and Nielson, H. R. (2009) Advice from belnap policies. In: *IEEE 22nd Computer Security Foundations Symposium*, 234–247.

Kriaa, S., Bouissou, M., Colin, F., Halgand, Y., and Pietre-Cambacedes, L., (2014) Safety and security interactions modeling using the BDMP formalism: Case study of a pipeline. In: *Computer Safety, Reliability, and Security*. Berlin, Germany: Springer, 326–341.

Kriaa, S., Pietre-Cambacedes, L., Bouissou, M., and Halgand, Y. (2015) A survey of approaches combining safety and security for industrial control systems. *Reliability Eng. Syst. Safety*, 139, 156–178.

Langer, R. (2012) Robust control system networks how to achieve reliable control after stuxnet.

Ni, Q., Bertino, E., and Lobo, J. (2009) D-algebra for composing access control policy decisions. In:*Proceedings of the 4th International Symposium on Information, Computer, and Communications Security*, ACM, 298–309.

Novak, T., Treytl, A., and Palensky, P. (2007) Common approach to functional safety and system security in building automation and control systems. In: *IEEE Conference Emerging Technologies and Factory Automation*, 1141–1148.

Pièrre-Cambacédès, L. and Bouissou M. (2010) Modeling safety and security interdependencies with BDMP (boolean logic driven markov processes). In: *IEEE International Conference Systems Man and Cybernetics (SMC)*, 2852–2861.

Pièrre-Cambacédès, L. and C. Chaudet (2010). The SEMA referential framework: Avoiding ambiguities in the terms security and safety. *Int. J. Critical Infrastruct. Protection*, 3 (2), 55–66.

Sun, M., Mohan, S., Sha, L., and Gunter, C. (2009) Addressing safety and security contradictions in cyber-physical systems. In: *Proceedings of the 1st Workshop on Future Directions in Cyber-Physical Systems Security (CPSSW09)*.

Woskowski, C. (2014). Woskowski, C. (2014) A pragmatic approach towards safe and secure medical device integration. In: *Computer Safety, Reliability, and Security*. Berlin, Germany: Springer, 342–353.