Enhanced TCP SYN Attack Detection

Vrizlynn L. L. Thing
vlt@doc.ic.ac.uk

Morris Sloman
m.sloman@doc.ic.ac.uk

Naranker Dulay
n.dulay@doc.ic.ac.uk

Department of Computing
Imperial College London
180 Queen's Gate, London, SW72AZ

***Abstract* - In this paper, we analyze the stateless SYN-SYN&ACK and SYN-FIN/RST detection mechanisms for TCP SYN attacks. We indicate the inherent vulnerability of the SYN-FIN/RST detection mechanism caused by the computation of the RST packet counts. We indicate why SYN-SYN&ACK is a more efficient and reliable detection mechanism than SYN-FIN/RST. We come up with 'Bot Buddies' for TCP SYN attacks and explain how the use of them can compromise both mechanisms. We propose an enhanced detection mechanism incorporating the Bloom filter to handle these variations of TCP SYN attacks. We show that our enhanced mechanism overcomes the problems of the use of Bot Buddies and analyse its efficiency.**

Keywords - Distributed Denial of Service Attacks, TCP SYN Flood, DDoS Detection, Network Security.

1. INTRODUCTION

Since its first appearance in 1999, Distributed Denial of Service (DDoS) attacks [1] have continued to become more prevalent in the Internet, with attacks targeting banking and financial companies, online gambling firms, web retailers and governments. The 2007 Symantec Threat Report [2] indicates that over 5000 DoS attacks were observed worldwide on a daily basis. In 2006, backscatter analysis [3] was conducted where DDoS attack traffic was captured. It shows that over a period of 3 years from 2001 to 2004, 22 collected distinct traces revealed 68,700 attacks on over 34,700 distinct Internet hosts, with 95% of the attacks using TCP as their choice of protocol. A recent survey [4] of 55 tier 1, tier 2 and hybrid IP network operators in North America, Europe and Asia reported that DDoS attacks remain the most significant ISP security threat with TCP SYN attacks leading the pack.

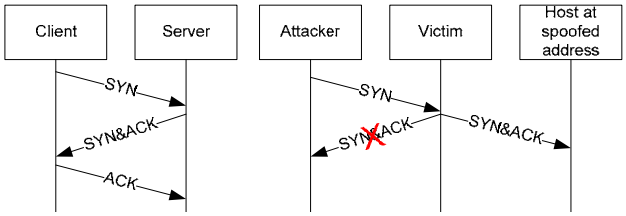


Figure (1a) TCP 3-way handshake (1b) TCP SYN Attack

In TCP, to establish a connection (before being able to carry out data transmission between the server and the client), the client sends a SYN to the server. The server allocates a buffer for the client and replies with a SYN&ACK[1] packet. At this stage, the connection remains in the half-open state while waiting for the ACK reply from the client to complete the connection setup, after which the 3-way handshake is achieved. TCP SYN DDoS attacks exploit the TCP 3-way handshake. The attackers spoof the source IP addresses of the massive amount of SYN packets they send to the victim servers. As a result, the SYN&ACK response packets do not reach the attackers' machines and the final ACK packets are not sent to the victim server to complete the 3-way handshake. Therefore, resources at the victim server are tied up for these half-open state connections created by the attackers preventing services to be granted to other legitimate requests.

Previous work on mitigating TCP SYN attacks include SYN cache [5], SYN cookie [6], SYNDefender [7] and Synkill [8]. These work aim to mitigate the flooding effect on the victim server. Previous work on detecting SYN attacks include Spectral analysis based detection [9], SYN arrival rate based detection [10], SYN-SYN&ACK based detection [11], and SYN-FIN/RST based detection [12]. Details are in the next section.

In this paper, we analyze the two existing stateless and efficient detection mechanisms, the SYN-SYN&ACK and SYN-FIN/RST schemes, in view of current and new variations of TCP SYN attacks. We find that these schemes are vulnerable to existing and new forms of attacks, and propose an enhanced SYN attack detection scheme to overcome these vulnerabilities.

2. RELATED WORK

In SYN cache [5], a hash table keeps track of the half-open state connections instead of relying on the backlog queue provided for each application. Therefore, a higher number of half-open state connections are possible but, during an attack, this is still insufficient. In addition, items in the hash table would have to be dropped constantly to allow for new requests and may result in even higher overhead and overwhelming the victim server during an attack.

[1] SYN&ACK is used to represent that both the SYN and ACK flags are set.

SYN cookie [6] eliminates the need for the backlog queue to keep track of each SYN request. Instead, a SYN cookie, also used as the initial sequence number in the SYN&ACK sent back to the client, is computed based on a counter at the server, the Maximum segment size in the SYN queue entry and the TCP header of the request packet. The ACK returned from the client must contain a sequence number of the SYN cookie plus one to be valid. However, the drawbacks of this scheme are the overhead of computing the cookies during an attack. Retransmission required in the situation of lost ACK packet is also not possible as the state of the connection request is not stored. In addition, TCP options which are not encoded in the SYN cookie are lost, denying the use of certain TCP performance enhancement.

In SYNDefender [7], the firewall intercepts the SYN request from the client and sends the SYN&ACK packet on the behalf of the server. After the firewall receives the ACK packet, the request is then let through to the server. In this way, the server does not need to hold the half-open states and so does not deplete its resources. However, the weakness is the additional workload and processing within the firewall which might not cope during a high rate attack.

In Synkill [8], source IP addresses are classified in a database as good or bad based on observed network traffic and administratively supplied input. Bad source addresses are sent the RST packet to terminate their requests while good ones are allowed to carry on with the handshaking.

However, the above methods only deal with mitigating the flooding effect of the SYN attacks and most of them are stateful mechanisms, which could be subjected to DDoS attacks themselves.

Spectral analysis based SYN attack detection [9] monitors the arrival rates of the traffic flows as a signal. The power spectral of the signal for a normal TCP flow is found to exhibit strong periodicity around its round-trip time (RTT) in both directions, whereas that of the DDoS attack traffic flows are not regulated in such a way. However, the scheme deals with long TCP flows. For TCP flows with short durations, the effect of their statistical multiplexing may outweigh their intrinsic periodicity and be detected as attack flows. RTTs of flows also vary from trip to trip which makes it difficult to obtain a reliable model to represent the normal traffic for different traffic conditions. Another limitation of the scheme is that it cannot identify TCP flows with very small RTTs.

The SYN arrival rate based detection scheme [10] models the arrival rate of the normal SYN packets as a normal distribution. A very reliable model of the normal traffic pattern has to be maintained. It allows a high rate SYN attack to be detected quickly and accurately. However, compared to non-parametric approaches such as the SYN-FIN/RST detection [12], it was not able to detect low rate attack (e.g. < 14 SYNs/sec). Therefore, a low rate attack would still be able to bring down the victim server without being detected.

Non-parametric approaches such as the SYN-SYN&ACK [11] and SYN-FIN/RST [12] detection mechanisms allow attack detection even in the case of any variance of normal traffic making them insensitive to site and access patterns. The SYN-SYN&ACK detection mechanism is based on the inherent TCP SYN – SYN&ACK pairs' behavior. With a SYN request sent out by a client to a server, it must also receive a SYN&ACK reply. An attack host spoofing as a client would only send out massive amount of SYN requests and not receive any SYN&ACK replies from the victim server due to its spoofed address. The SYN-SYN&ACK agent monitors the difference of the number of outgoing SYN and incoming SYN&ACK packets. It then uses the non-parametric Cumulative Sum (CUSUM) approach to detect any abrupt rise in the difference. The SYN-FIN/RST approach proposed by the same authors allows detection both at the attackers' and victim's ends. For a normal TCP connection, a starting SYN request will be ended by a FIN/RST to close the connection. Therefore, correlation is performed between the number of SYN and FIN/RST packets instead. However, as RST can be classified as active (i.e. initiated to abort TCP connection) and passive (i.e. transmitted in response to packets destined to a closed port), and could not be easily differentiated at the monitoring agents, the scheme counts 75% of all RST packets to be active and the rest to be passive (i.e. background noise). Non-parametric CUSUM is also used to detect the abrupt rise in the difference between the SYN and the FIN/RST packets. We analyze both schemes in the next section at their current state and in view of proposed new variations of TCP SYN attacks.

3. SYN-FIN/RST AND SYN-SYN&ACK ANALYSIS

In SYN-FIN/RST, SIN, FIN and RST packets in both directions are monitored and counted. To address the issue of passive RST packets, only 75% are counted as valid RST. These valid RST are added to the FIN packets and the difference between the number of SYN and FIN/RST packets for each monitoring interval is computed. Any abrupt positive fluctuation in this difference would signal the occurrence of an attack.

We find that classifying 75% of RST packets as valid ones weaken the detection mechanism. RST packets can be generated by a host for the following events:

i. arrival of data packets for which no connection has been established

ii. arrival of TCP segments with inappropriate sequence numbers

iii. arrival of SYN&ACK packets for which no SYN has been initiated

iv. arrival of TCP packets for closed ports

In the event of a SYN attack, attackers could target random ports as in [13]. In this case, the victim server would generate RST packets due to (iv). In addition, source address spoofing is used by the attackers. Therefore, the SYN&ACK packets generated by the victim server would be delivered to hosts located at the spoofed addresses. They would generate RST packets to the victim server due either to the events in (iii) and (iv) as well. Whichever the case, RST packets generated by the above events should not be classified as valid packets.

In [13], DDoS attack tools in Bots could send a mixture of SYN and ACK packets to the victim. Instead, a new variation of attack could be easily created by sending out a mixture of SYN, FIN and RST packets to the victim server. This would result in balancing the SYN and FIN/RST packets, and thus weakens or even defeats the SYN-FIN/RST detection mechanism.

As mentioned in the previous section, the SYN-SYN&ACK approach is applicable at the source of the attack instead of the victim's end due to the detection which is based on the behaviour of SYN and SYN&ACK pairs. Having the detection mechanism closer to the attack sources allows a speedier detection before a damaging impact could be at the the victim server. As the attackers send out SYN packets and do not receive any SYN&ACK from the victim due to source IP address spoofing, this approach is very efficient in attack detection. In comparison to the SYN-FIN/RST detection scheme, the SYN-SYN&ACK scheme also allows a higher degree of correlation and accuracy detection due to the shorter round trip time between the SYN-SYN&ACK pairs instead of the time difference between the SYN-FIN/RST pairs, which last for the whole duration of the TCP session. Therefore, a shorter monitoring interval and detection time could be achieved in the case of the SYN-SYN&ACK detection approach.

As mentioned above, it would be possible to create a new variation of the attack by sending out a mixture of SYN and SYN&ACK packets as well. However, as only the outgoing SYN and incoming SYN&ACK packets are counted, even if the attackers were to make such modifications to the attack code, it would not have any impact on the detection mechanism. Instead, the SYN&ACK packets would only weaken the attack (by reducing the SYN attack traffic sent to the victim due to resources used for sending out SYN&ACK packets).

Instead, we come up with another new variation of co-ordinated attack to defeat the SYN-SYN&ACK detection mechanism. We named this attack the 'Bot Buddy' Attack (shown in Figure 2), as it requires the co-operation of bots within the botnet carrying out the SYN attack. For each SYN packet sent out to the victim server, a SYN&ACK packet with the source address spoofed to the victim server is sent to another bot within the botnet. In this case, each outgoing SYN packet has an incoming SYN&ACK "reply". This attack will therefore circumvent the SYN-SYN&ACK detection. Although the diagram shows a 2-buddy botnet system which is the safest case for the attacker, it is also feasible to have 1 bot responsible for sending to multiple bots (i.e. 1 to Many Bot Buddies Attack) as shown in Figure 3. The reason is that only the outgoing SYN and incoming SYN&ACK counts are monitored. In addition, as the Bot master has a list of all the bots in the botnet, this attack could be easily implemented.

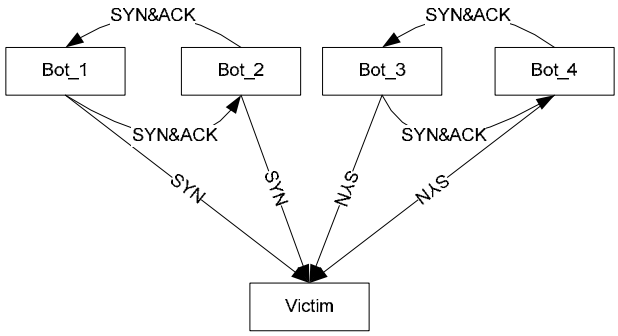


Figure 2: Bot Buddy Attack

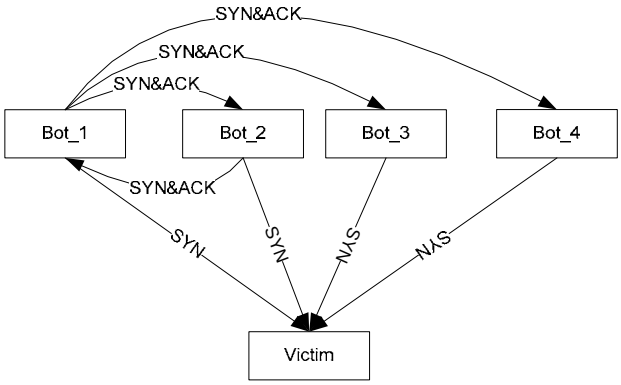


Figure 3: 1 to Many Bot Buddies Attack

As the SYN-FIN/RST detection mechanism has an inherent flaw and the SYN-SYN&ACK detection mechanism proved to be a more efficient and effective for SYN attack detection, we propose an enhancement to the SYN-SYN&ACK detection mechanism to resolve the above vulnerabilities.

4. ENHANCED TCP SYN ATTACK DETECTION

As in the original SYN-SYN&ACK detection mechanism, the packet sniffing agents are located at the leaf routers that connect end hosts to the Internet. We consider the SYN and SYN&ACK packets sent by the bots. We assume that the source IP addresses of the SYN packets are randomly spoofed (systematic spoofing will be considered in future work), whether all 4 bytes or just the host suffix of the IP address. Therefore, the following situation shown in Figure 4 arises.

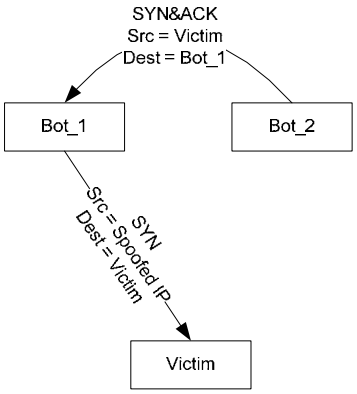


Figure 4: Attack Packet's Header

Since the source IP address used in the SYN packet does not match the destination IP address in the SYN&ACK packet received at Bot_1, we could perform SYN-SYN&ACK pair matching to eliminate the effect caused by the Bot Buddy Attack on the SYN-SYN&ACK detection mechanism. Instead of storing the flow addresses to perform matching, we propose using Bloom filters [14] to achieve space and time efficiency. We describe the algorithm of our mechanism as follows.

In our detection mechanism, we define our Bloom filter, *F[0…m-1]*, as an *m*-bit array which is initialized to 0. We define each element to be stored in the filter as e_{out}, which corresponds to the SYN packet being sent out.

$$e_{out} = src \oplus dest \tag{1}$$

k hash functions, $h_1()…h_k()$, are used to compute *k* key values for e_{out} and e_{out} is stored into the filter by,

$$F[h_i(e_{out}) \bmod m] = 1, \quad for\, i = 0\, to\, k \tag{2}$$

For the incoming SYN&ACK packet, the same equation (1) to compute e_{out} is used to compute the corresponding element, e_{in}. In a normal TCP 3-way handshake, there will be an outgoing SYN with element value of e_{out} which is equal to an incoming SYN&ACK with element value of e_{in}. To be counted as a valid SYN&ACK packet for inclusion into the CUSUM detection algorithm, all the bits at positions, $\{h_i(e_{in}) \bmod m\}$ in the filter array must be set to 1. In the next section, we perform evaluations and analysis of our detection mechanism.

5. EVALUATION AND ANALYSIS

Using the Bloom filter to validate the SYN&ACK replies allows space and time efficiency. The time required to store and search for an element in the filter is a fixed constant, O(*k*), which is independent of the size of the filter and the number of stored elements. The space allocated to the Bloom filter is *m* bits. This allocation depends on the availability of storage space on the leaf router. The Bloom filter has a zero false negative (i.e. if an element has been stored, it will be found in the filter) but a non-zero false positive (i.e. if an element has not been stored, it might still return the status of "found"). Assuming that the hash functions spread the elements evenly across the filter array, let p_0 be the probability that a bit in the array is not set (i.e. 0) by any of the hash functions after storing *n* elements. Let p_1 be the probability that the bit is set (i.e. 1).

$$p_0 = \left(1 - \frac{1}{m}\right)^{kn} \tag{3}$$

$$p_1 = 1 - p_0 \tag{4}$$

For a false positive error to occur, during a search of an element, each of the *k* array positions computed by the hash functions for the element must be set to 1. Therefore, the probability of a false positive error, p_e, is given by:

$$p_e = (p_1)^k \tag{5}$$

We see that, as *m*, the array size, increases, p_e will decrease. However, as *n*, the number of elements to be stored, increases, p_e will increase too.

Other than the false positive error from the Bloom filter, we may consider the possibility of error coming from the address spoofing algorithm. That is, if the spoofed source IP address of the outgoing SYN packet happens to be the same as the destination of the incoming SYN&ACK reply generated by the Bot Buddy. This address would also be the real address of the Bot sending out the SYN attack packet. Although the SYN packet would not constitute an attack packet in this case as a SYN&ACK reply would be received from the victim thus completing the 3-way handshake and establishing the connection, we have to take into consideration the additional SYN&ACK reply that would come from the bot buddy. Assuming that all 32 bits of address are spoofed, the probability of collision (with the real source address) is given as:

$$p_c = \frac{1}{2^{32}} \tag{6}$$

However, if partial address spoofing whereby network prefix of the real address is preserved when performing spoofed address generation, and *q* is the number of bits of the preserved network prefix,

$$p_c = \frac{1}{2^{32-q}} \tag{7}$$

We now analyze our enhanced detection mechanism taking into consideration the false positive error of the Bloom filter and the collision of the spoofed address. As in the original SYN-SYN&ACK detection mechanism, let $\{\Delta_t,\ t=0,1,\ldots\}$ be the number of outgoing SYN packets

minus that of the incoming SYN&ACK packets collected from each sampling period.

$$\Delta_t = SYN_t - SYN\,\&\,ACK_t \quad (8)$$

To alleviate its dependence on the time, access pattern and size of the network, $\{\Delta_t\}$ is normalized by the average number $\overline{K}$ of incoming SYN&ACK packets during the sampling period. The recursive estimation of $\overline{K}$ is given by:

$$\overline{K}_t = \alpha \overline{K}_{t-1} + (1-\alpha) SYN\,\&\,ACK_t \quad (9)$$

where t is the discrete time index and α is a constant between 0 and 1 to represent the memory in the estimation. We define $X_t = \Delta_t / \overline{K}_t$ whereby the mean of X_t, denoted by c, is much less than 1. In general, $E(X_t) = c < 1$. A parameter $a > c$ is chosen and $\widetilde{X}_t = X_t - a$ is defined so that a negative mean is achievable during normal operation. When an attack occurs, $\widetilde{X}_t$ quickly become a large positive number. The abrupt rise detection is based on the observation of $h >> c$, whereby the increase in the mean of $\widetilde{X}_t$ can be lower bounded by h.

$$y_0 = 0 \quad (10)$$

$$y_t = (y_{t-1} + \widetilde{X}_t)^+ \quad (11)$$

y_t is defined as the maximum continuous increment until time n. A large y_t is a strong indication of an attack. Equation (11) indicates that y_t is set to $(y_{t-1} + \widetilde{X}_t)$ if this value is ≥ 0 else it is set to 0. N is defined as the attack threshold, i.e. $y_t \geq N$ indicates an attack is detected.

Taking into consideration the false positive errors in the Bloom filter search and the collision error of the spoofed address, we get

$$\Delta_t = SYN_{t,norm} - SYN\,\&\,ACK_{t,norm} + SYN_{t,att} - Err_{t,att} \quad (12)$$

during a Bot Buddy SYN attack. $SYN_{t,norm}$ are the number of SYN packets and $SYN\&ACK_{t,norm}$ are the number of SYN&ACK packets from the legitimate traffic respectively, in interval t. $SYN_{t,att}$ are the number of SYN packets and $Err_{t,att}$ are the number of SYN&ACK packets from the Bot Buddy attack respectively, at interval t. Note that in the original SYN-SYN&ACK detection mechanism, $Err_{t,att}$ will be large (i.e. $\approx SYN_{t,att}$) and the attack will not be detectable. In our detection approach, $Err_{t,att}$ is given by:

$$Err_{t,att} = \sum_{i=1}^{SYN_{t,att}} (2p_c + (1-p_c)(p_e)) \quad (13)$$

As in the case of collision, $2p_c$ represents one SYN&ACK returned from the victim server and the other from the Bot Buddy. Only when there is no collision is there a possibility of false positive error in the Bloom filter (i.e. there is no matching SYN element stored but searching returns true). This is represented by $(1-p_c)(p_e)$ in the equation. n in equation (3), which is used to derive p_e, refers to the number of stored elements already present in the bloom filter. At the beginning of the first sampling period, n starts with 0. Subsequently, n is incremented as SYN packets, both legitimate and attack, arrive. We will consider element removal in the future work (plan in Section 6).

Using the experiment parameters in the paper describing the original SYN-SYN&ACK detection mechanism, the sampling period is set to 20 seconds. We choose the attack rate to be 60 SYN packets/second.

In [11], the normal traffic traces dated September 2000, of the outgoing SYN and incoming SYN&ACK were obtained from the University of North Carolina (monitored on the high-speed OC-12, 622Mbps link connecting its Chapel Hill campus network to the rest of the world). The traces show that the normal outgoing SYN packets fluctuated from around 1200 to 1900, while the incoming SYN&ACK packets fluctuated from around 1050 to 1700, in 10-second sampling intervals. The traces show consistent synchronization between the SYN and SYN&ACK packets. Therefore, Δ_t is consistently around 200 during normal operations in 10 seconds, which is 400 in 20 seconds, the detection mechanism's sampling period.

As for our mechanism, we set k=1 (i.e. one hash function), m=33,554,432 (i.e. Bloom filter size of 4MB) and assume that the bot spoofs 32 bits of the source address. In [15], the RTTs for FTP downloads at 6 different Internet sites have been found to average to 65.27ms. Therefore, we assume that each SYN&ACK packet would take 65.27ms to reach the leaf router of the attack bot. The outgoing SYN packets would not incur much delay as the leaf router is located very close to the attack source. The attack arrival rates are set to 60 packets/second (i.e. around 3 packets every 50ms assuming 15.27ms incurred for the generation of the attack packets and minor trip time to the leaf router), and the normal SYN traffic arrival rate is averaged at 155 packets/second (i.e. (1200+1900)/(2*10)) or around 10 packets every 65.27ms; no delay is assumed here as the data is obtained from traces and is the actual arrival rate at the leaf router). Therefore, we assume that n at each arrival of a Bot Buddy SYN&ACK to be 20x1550=31000 (i.e. legitimate SYN packets in 1 sampling period) plus the time slots of 65.27ms that have passed

multiplied by 13, as we assume starting the attack one sampling period later than the legitimate traffic.

The first computation of $Err_{t,att}$ will begin only after the arrival of the first SYN&ACK packet (i.e. comes after SYN packets and n will be > 0). Figure 5 shows the $Err_{t,att}$ for the samples. We observed that after 10 minutes of attack, the number of incorrectly validated SYN&ACK packets is just about 1. Therefore, we can be assured that our enhanced detection mechanism will detect SYN attacks effectively even in the face of the Bot Buddy attack.

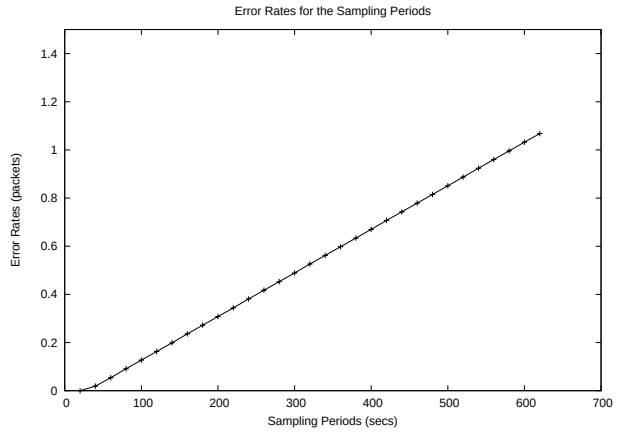


Figure 5: Error Rates Computation

6. FUTURE WORK

Both development of the original SYN/SYN&ACK detection mechanism and our enhanced detection mechanism is near completion. We plan to run experiments to compare the performance of both approaches in the situations of the original TCP SYN attack and new variations of the attack. As hash functions are used to compute the key values for storage of the elements in the Bloom filter, we would like to find out the overhead incurred in an experimental scenario. The selection of k (the number of hash functions) and m (the size of the Bloom filter), and their effect on the detection speed and false positive rate will also be studied. We have not considered the removal of elements from the Bloom filter, which is an essential feature of the scheme, in this paper. We will include the study in our future work as well, considering Counting Bloom Filter.

7. CONCLUSION

We have analyzed the stateless SYN-SYN&ACK and SYN-FIN/RST detection mechanisms. We discover the inherent vulnerability of the SYN-FIN/RST detection mechanism caused by the RST packet counts. Both mechanisms also suffered in terms of reliability in view of our new variations of TCP attacks. Although, the SYN-SYN&ACK detection mechanism is found to be more efficient and reliable compared to the SYN-FIN/RST, it fails to detect our Bot Buddy attack. We proposed an enhanced detection mechanism incorporating the Bloom filter to handle the attack. We analyzed and evaluated our enhanced mechanism and found it to work as effectively as the original SYN&ACK, as if the Bot Buddy attack is not present.

8. ACKNOWLEDGEMENTS

This research is continuing through participation in the International Technology Alliance sponsored by the U.S. Army Research Laboratory and the U.K. Ministry of Defence.

9. REFERENCES

1. Diane E. Levine and Gary C. Kessler, "Chaper 11 - Denial of Service Attacks, Computer Security Handbook, 4th Edition", Editors - Seymour Bosworth, Michel E. Kabay, 2002
2. Symantec Internet Security Threat Report, Volume XI, Mar. 2007
3. David Moore, et al., "Inferring Internet Denial-of-Service Activity ", ACM Transactions on Computer System (TOCS), May 2006, **24**(2), pp. 115-139.
4. Arbor Networks, "Worldwide ISP Security Report", Sept. 2006
5. Jonathan Lemon, "Resisting SYN flood DoS attacks with a SYN cache", USENIX BSDCon, 2002
6. Daniel J. Bernstein and Eric Shenk, http://cr.yp.to/syncookies.html, 1996
7. Check Point Software Technologies Ltd., SynDefender, http://www.checkpoint.com
8. Christoph L. Schuba, et al., "Analysis of a Denial of Service Attack on TCP", IEEE SYmposium on Security and Privacy, May 1997
9. Chen-Mou Cheng, et al., "Use of Spectral Analysis in Defense Against DoS Attacks", IEEE GLOBECOM, 2002
10. Yuichi Ohsita, et al., "Detecting Distributed Denial-of-Service Attacks by Analyzing TCP SYN Packets Statistically", IEICE Trans. Comm, Oct. 2006, **E89-B**(10), pp. 2868-2877
11. Haining Wang, et al., "SYN-dog: Sniffing SYN Flooding Sources", International Conference on Distributed Computing Systems, Jul. 2002
12. Haining Wang, et al., "Detecting SYN flooding attacks", IEEE INFOCOM, Jun. 2002
13. Vrizlynn L. L. Thing, et al., "A Survey of Bots Used for Distributed Denial of Service Attacks", IFIP International Information Security Conference (SEC), May 2007
14. Burton H. Bloom, "Space/time trade-offs in hash coding with allowable errors", Communications of the ACM, Jul. 1970, **13**(7), pp. 422-426.
15. Bryan Veal, et al., "New Methods for Passive Estimation of TCP Round-Trip Times", Passive and Active Measurements (PAM), 2005