Defense Against SYN Flood DoS Attacks Using Network-based Mitigation Techniques

Patrik Goldschmidt
CESNET a.l.e., Prague, Czech Republic
goldschmidt@cesnet.cz

Jan Kučera
CESNET a.l.e., Prague, Czech Republic
jan.kucera@cesnet.cz

***Abstract*—TCP SYN Flood is one of the most widespread DoS attack types performed on computer networks nowadays. As a possible countermeasure, we implemented and deployed modified versions of three network-based mitigation techniques for TCP SYN authentication. All of them utilize the TCP three-way handshake mechanism to establish a security association with a client before forwarding its SYN data. These algorithms are especially effective against regular attacks with spoofed IP addresses. However, our modifications allow deflecting even more sophisticated SYN floods able to bypass most of the conventional approaches. This comes at the cost of the delayed first connection attempt, but all subsequent SYN segments experience no significant additional latency (<0.2ms). This paper provides a detailed description and analysis of the approaches, as well as implementation details with enhanced security tweaks. The discussed implementations are built on top of the hardware-accelerated FPGA-based DDoS protection solution developed by CESNET and are about to be deployed in its backbone network and Internet exchange point at NIX.CZ.**

***Index Terms*—TCP SYN Flood, DDoS mitigation, TCP SYN Authentication, RST Cookies, SYN Drop, TCP Handshaker**

I. Introduction

Transmission control protocol (TCP) is an integral part of the Internet protocol suite. As its importance is fundamental for the operation of the Internet, it is often misused to cause various cybersecurity threats. Data from the past several years show a strong trend towards TCP abuse to perform Distributed Denial of Service (DDoS) attacks. The report from Q2 2020 by Kaspersky Lab states that the most frequent way of a DDoS was TCP SYN flooding, utilized by 94.7% of all the attacks [1]. According to Cisco, the number of DDoS attacks will double to 14.5 million p.a. by 2022 [2].

The purpose of this paper is to describe details of this weakness, present existing countermeasures, and most importantly, convey our experience with the implementation and evaluation of three network-based SYN Flood mitigation strategies. Our implementations have been designed as a part of the DDoS protection for high-speed networks developed by CESNET [3].

In this work, we extend the device's mitigation capabilities by developing network-based heuristic approaches to mitigate SYN Flood attacks. The discussed methods are not as widespread as end-host TCP SYN Cookies but are way more effective in certain situations, such as when dealing with an enormous amount of spoofed IP addresses, and enable flexible utilization according to the current attack surface. Proposed algorithms are used only as a reactive defense against ongoing cyber-attacks, hence not affecting the traffic when no threats are detected. With the use of high-capacity data structures posing as IP whitelist and blacklist, our robust solution also supports SYN limiting, creating an especially strong mitigation mechanism even against more sophisticated attacks.

II. TCP Security Considerations

The creation of a reliable data channel required by the TCP is achieved by a three-way handshake. The process starts with an initiating host (client). The client sends a segment with the SYN flag set and generates a pseudo-random value of x used as the Sequence number (SEQ_x). The receiving host (server) then generates its SEQ_y, acknowledges the client's data by setting its ACK to the received segment $SEQ_x + 1$, and enables SYN and ACK flags. The client also acknowledges the received segment, and the channel setup is completed.

A. Known Vulnerabilities and Attacks

Attacks on the TCP are classified as either flood-based or injection-based. Flood attacks aim to exhaust the target's resources by flooding with bogus packets, making it inaccessible for regular clients, hence creating a denial of service. On the other hand, injection attacks are based on eavesdropping on the communication and injecting crafted segments into the TCP session. Injected data may contain malicious code, compromise the user's privacy [4], or reset the session [5].

The functionality of the most popular attack – TCP SYN Flood depends on the 3-way-handshake mechanism, during which the server waits for the arrival of the final ACK to mark the connection as established. The rationale behind a successful DoS assumes that the victim allocates a new state for every received SYN segment and that there is a limit of such states that can be stored. These are defined in RFC 793 [6] as Transmission Control Block (TCB) data structures. TCBs are used to store necessary state information for each TCP connection, and so they require a new memory allocated for each received SYN [7].

Operating system kernels typically try to protect host memory from exhaustion by implementing a limit of contemporary

This research was supported by the Security Research Programme of the Czech Republic 2015 – 2022“ (BV III/1 – VS) granted by the Ministry of the Interior of the Czech Republic under No. VI20192022137 Adaptive protection against DDoS attacks.

978-3-903176-32-4 © 2021 IFIP

TCB structures called backlog. When its limit is reached, either incoming SYN segments are ignored, or uncompleted connections in the backlog are replaced. The primary goal of the SYN flooding is thus to exhaust the target's backlog with half-open connections. For this purpose, spoofed IP addresses that do not generate replies to SYN-ACKs, are often used.

Other attacks include various types of floods, e.g., SYN-ACK, RST, and FIN flood. More sophisticated techniques, such as Fake session, include ACK and FIN segments alongside many SYNs to simulate a real client's traffic. Another technique is Session attack, which utilizes a botnet to create many valid TCP connections at once and stretch them as long as possible, making the victim server inaccessible.

B. SYN Flood Mitigation Techniques

Modern operating systems provide relatively large backlogs, being less vulnerable to regular SYN flooding attacks. However, backlogs can not cover distributed variants of the attack, so specialized methods are still required. Linux kernels historically provided robust security by implementing two end-host countermeasures – *SYN cookies* and *SYN caching* [8].

SYN cache method utilizes hashing to store a lightweight fingerprint for every incoming TCP connection. This way, the operating system does not need to allocate the whole TCB, but only a fragment of the original memory is required. Therefore, it is able to queue more requests and so is harder to exhaust [8].

In contrast to SYN cache, the SYN cookies method does not need to store any state information at all. Essential data defining the connection alongside a timestamp and a secret are hashed into a 32-bit value representing the *SEQ* number of the SYN-ACK segment. Upon ACK response receipt, the server can reconstruct original SYN parameters and successfully establish a connection. However, the method denies SYN-ACK retransmission and also restricts TCP options usage [9].

A little-used but interesting approach is *TCP Random drop*. Its principle is to replace a random half-open connection when the backlog is full and another SYN is received. Replacement is done by sending a RST segment, discarding the TCB structure, and allocating a new one for the incoming connection. Dropped legitimate clients are expected to try to reestablish a connection. Its rationale is that by making the queue large enough, a server under attack can still offer a high probability of successful connection establishment, but legitimate sessions may still be occasionally denied [10].

Although often effective, the presented end-host mitigation techniques are not suitable in all scenarios. Some of them could be implemented as a part of the intermediary device software, but their usage would require a mapping between different *SEQ* and *ACK* numbers, making their operation rather inefficient. Therefore, other specialized approaches also exist.

Various TCP extensions and modifications with anti-DoS capabilities like TCP Cookie Transactions [11] and TCP Fast Open [12] are also available. However, none of them is globally used, mainly due to the lack of support from vendors.

Other network-based countermeasure techniques include *traffic filtering* [13] and its improved variant *reverse-path forwarding* [14]. Their fundamental idea is to deny all incoming traffic that does not match its source network prefix. This allows discarding all traffic from spoofed IP addresses, but its deployment would be necessary on the majority of Internet service providers, which cannot be generally relied on [15].

SYN Flood attacks were historically mitigated by firewalls, proxies, or IDS/IPS systems, which usually used SYN-ACK spoofing or ACK spoofing techniques [15] [16]. These practices are mostly present to this day but often reside in the cloud as a part of virtualized IDS/IPS systems instead of traditional per-network defense [17]. The methods mentioned above are, however, not always optimal. SYN-ACK spoofing does not solve the mentioned problem with degraded performance due to the required SYN/ACK values mapping. ACK spoofing can protect the server's backlog but distributed SYN floods may still cause network congestion or high processor utilization of the security intermediary device or the server itself. Therefore, another three methods providing both good performance and decent SYN-flood protection are introduced in Section III.

Both end-host and network-based techniques are frequently employed, and they generally do not interfere when used in combination [15]. Newer trends in DDoS mitigation also utilize artificial intelligence and machine learning principles, such as [18], which are generally able to protect against a wider range of attacks but suffer from a poorer performance when compared to their previously mentioned counterparts.

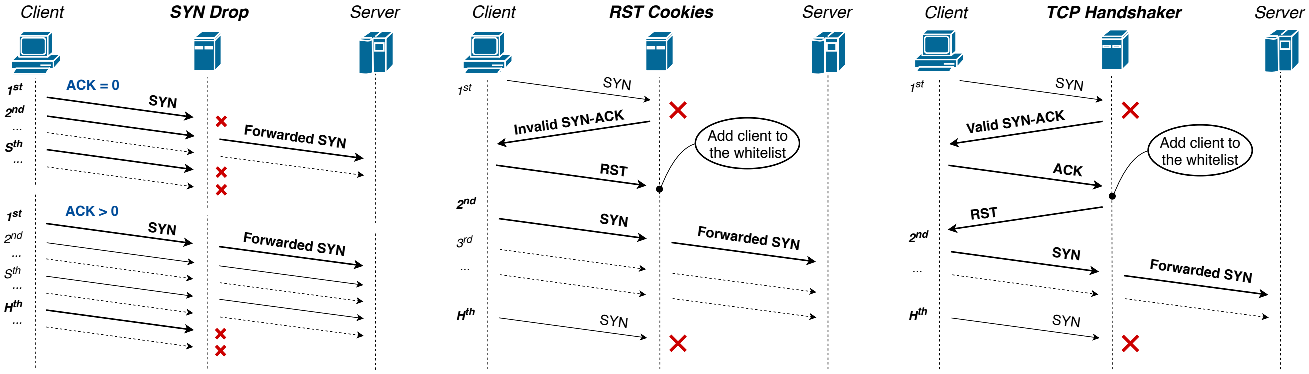
III. Network-based Mitigation Methods

The SYN Cookies end-host mitigation principle proved to provide adequate protection against SYN Flooding attacks, but its usage may be undesirable in certain situations. Since servers are typically busy handling clients' requests, it may be preferable to filter the traffic on the network level, thus not wasting their resources by processing potentially malicious traffic. For this purpose, three TCP SYN authentication algorithms are presented in the following subsections.

A. SYN Drop

SYN Drop mitigation method is based on a simple principle to limit the maximum number of sent SYNs from a single IP address. For this reason, the module needs to keep an internal state for all clients, monitor their connections, and count the number of SYNs and ACKs received from them. The number of allowed SYNs is given by *soft* (S) and *hard* (H) thresholds. When no ACK from the corresponding IP address is received in the particular time window, the soft threshold, allowing only a couple of packets, is active. If at least one ACK is received, SYNs are limited by the hard threshold, which may allow up to hundreds of connections in a few seconds. Additionally, the first SYN in the soft threshold's case is always dropped to prevent SYN port scanning (Fig. 1a).

The described mechanism effectively denies dummy heavy-hitters by policing the maximum number of SYNs a single host can send. Nevertheless, attackers utilizing massive IP address spoofing may produce enough traffic, managing to take down



(a) SYN Drop functionality.

(b) RST Cookies functionality.

(c) TCP Handshaker functionality.

Fig. 1: Network-based TCP SYN Flood mitigation techniques using TCP SYN authentication.

the server with only the soft threshold active. Alternatively, ACK messages injection or Fake session attacks would be able to fool the mechanism and make it ineffective. Therefore, we also present other SYN Flood mitigation methods in subsections III-B and III-C.

B. RST Cookies

The first mention of the RST Cookies approach can be traced back to 1996 [10]. Unfortunately, the method was never officially published, and the original proposal was only in the form of e-mail communication. It was also never popularized due to incompatibility with Windows 95 [19] and potential performance issues on slow networks. Mentioned e-mails were probably deleted, and so only a few resources exist to this day. For the purpose of our custom implementation, the method needed to be “reinvented”, estimating the behavior of the clients according to the specification and actually testing various operating systems for the expected compatibility.

RST Cookies is a heuristic mitigation technique utilizing the 3-way handshake mechanism while relying on the client’s behavior as defined in RFC 793. Its fundamental idea is to establish a security association with clients before allowing their connection requests. This is achieved by crafting an intentionally invalid SYN-ACK response to the first SYN received from a client. RFC 793 [6] defines the behavior as follows:

> *If the connection is in any non-synchronized state, and the incoming segment acknowledges something not yet sent (carries an unacceptable ACK), a reset is sent.*

To distinguish whether the RST segment is associated with the invalid SYN-ACK reception, RFC 793, section 3.4 [6] also defines requirements on the sent RST:

> *If the incoming segment has an ACK field, the reset takes its sequence number from that ACK field.*

According to these preconditions, we can distinguish a legitimate client from an attacker without storing any state information locally. Instead, the client’s authentication is performed solely on the value stored in the *ACK* field of the SYN-ACK reply. The first SYN from a new (not whitelisted) client is dropped, and an invalid SYN-ACK reply is used to verify its validity (Fig. 1b). The regular client will send a valid RST reply, whereas the attacker without the real TCP stack will not. When the valid RST is received, a security association is established by whitelisting the client’s IP address. SYN traffic originating from whitelisted IP addresses is forwarded to its desired destination without further tampering (Fig. 1b).

The algorithm hence blocks all received SYN segments from unknown hosts until a security association with them is established. On account of this behavior, the protected server does not initially know about the intentions to establish a session and thus no state information are allocated until the client is considered legitimate. This mechanism effectively denies all attacks from spoofed IP addresses, which can not generate a valid reply. Random RST segments can not fool the security mechanism because the specific *SEQ* value is expected. For the attacker without a valid TCP stack, it is rather problematic to inject a RST segment with the desired *SEQ*. Therefore, the only viable way is to use legitimate (non-spoofed) clients. As further discussed in Section IV, we enhanced the original algorithm by implementing a hard threshold to limit the maximum number of SYNs from already-validated clients (Fig. 1b), thus merging it with the functionality of SYN Drop.

C. TCP Handshaker

SYN Authentication with *TCP Handshaker* is practically a version of the end-host SYN Cookies method ported to the network-based environment. As outlined in Section II-B, the trustworthiness of the initiating host is verified by setting a specific *SEQ* in the SYN-ACK response and then verifying the value from the ACK segment the client returns.

The TCP Handshaker method (Fig. 1c) works according to this principle, where a specific *SEQ* value is set in the SYN-ACK response and expects the client to confirm its validity by responding with a required value of $SEQ + 1$ in its ACK segment. If the values are matched, its IP address is added to the whitelist, and its SYN data are not intervened anymore. However, after the client sends an ACK finalizing the handshake, it thinks that the session is established because the 3-way handshake was successfully finished from its perspective. The problem at this point is that the server knows nothing about the session since the client’s SYN was intercepted by the algorithm, and thus never reached the server. To synchronize nodes in this state, the TCP Handshaker needs to send a RST

```
1:  $entry \leftarrow$ IP data from association table
2:  if ($entry == \textbf{NIL}$):
3:      send (invalid) SYN-ACK; drop SYN and exit;
4:  else if ($t_s - entry.t_a > t_m$):
5:      delete IP from association table;
6:      send (invalid) SYN-ACK; drop SYN and exit;
7:  if (SYN limiting enabled):
8:      if ($t_s - entry.window\_start \geq 1\text{s}$):
9:          $entry.syn\_cnt \leftarrow 0$;
10:         $entry.window\_start \leftarrow t_s$;
11:     else if ($entry.syn\_cnt \geq$ SYN limit):
12:         if (Blacklist enabled):
13:             add IP to blacklist;
14:         delete IP from association table;
15:         drop SYN and exit;
16:     $entry.syn\_cnt \leftarrow entry.syn\_cnt + 1$;
17: allow SYN and exit;
```

Fig. 2: SYN Processing of RST Cookies / TCP Handshaker.

for the client's session after its ACK is processed. When the RST is received, the client closes its session and may start another one automatically based on its implementation (Fig. 1c). This behavior is further discussed in Section V.

IV. Design and Implementation Remarks

Since the SYN Drop technique is rather simple, this section will mostly focus on the problematics related to RST Cookies and TCP Handshaker. The following subsections discuss SYN processing and the client authentication process in more detail.

A. SYN Processing

All of the presented methods require to process all ingress SYN segments. In addition, RST Cookies and TCP Handshaker have to process all RSTs or ACKs, respectively. When a SYN is received, the algorithm must determine whether it originates from a new client or a client that is already verified. For this purpose, we use a hash table with the source IP address as its key. The contents of its entries depend on the mitigation method, but various timestamps and counters have to be used. For example, each entry for RST Cookies or TCP Handshaker contains a timestamp specifying when the association has been created (t_a), allowing entries to age. Therefore, upon a SYN segment arrival (t_s), these algorithms have to check whether the IP address is contained in the whitelist and its entry timestamp does not exceed the maximum specified age time (t_m), thus validating the condition: $t_s - t_a > t_m$. If the condition is met, the SYN is dropped, and a valid or invalid SYN-ACK is assembled and sent as a response. Otherwise, the processed SYN is forwarded to its desired destination.

A regular version of SYN processing is depicted in Fig. 2 (lines 1-6). We enhanced the algorithm functionality by adding a counter (*syn_cnt*) and timestamp (*window_start*) to the hash table alongside the existing association timestamp. These are used to implement the hard SYN threshold functionality for already associated clients (lines 7-16). The modified algorithm with the hard limit will successfully block sending large amounts of SYNs by any sophisticated attackers, who would manage to guess the whitelisted IPs or somehow pass through the security association phase. When combined with a blacklist, an ability to detect these smart attackers and deny their traffic entirely is available as well (lines 12-13).

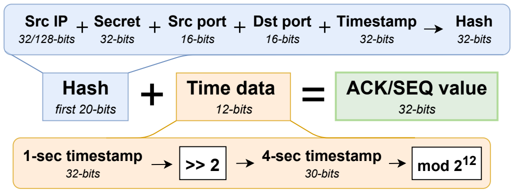


Fig. 3: SYN-ACK generation – the hashing method.

B. Validation Packets Processing

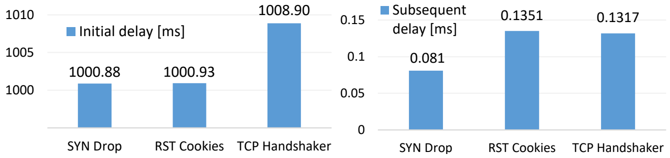
RST segments for RST Cookies and ACKs for TCP Handshaker are used for client validation and thus need to be treated differently in these particular methods. These algorithms have to decide whether the RST/ACK is a part of its authentication mechanism or belongs to the regular TCP traffic. This is done by looking at the *SEQ* (for RSTs) or *ACK* (for ACKs) of the obtained data. If this value is equal to the expected value defined by the algorithm, it is probably a response to its previously sent SYN-ACK. In this case, the processed segment is dropped, and the client's IP address whitelisted. Otherwise, the algorithm has to forward the segment to its destination.

C. ACK/SEQ Values Generation and Validation

The key concept of both RST Cookies and TCP Handshaker is to generate either valid or invalid SYN-ACKs and use their *ACK* or *SEQ* values for future client authentication. In the RST Cookies' case, an invalid SYN-ACK is crafted by setting its *ACK* value differently from the $SEQ + 1$ of the SYN it is referring to. TCP Handshaker requires a valid SYN-ACK, but we are still free to set its *SEQ* as desired. Both of the algorithms hence need to generate a value that cannot be easily guessed by an attacker and which they can reconstruct when required. The simplest solution to this is a constant value placed in each SYN-ACK response and then checked for the match in the RST/ACK. This approach would be functional, but a smart attacker could monitor the traffic and inject the required type of packet with the given constant to trick the security mechanisms. To tackle this issue, we propose a system of dynamic TCP number generation and validation.

Our design of the dynamic TCP number generator and validator uses two policies with two security levels. The first policy generates random numbers periodically and assigns the values to SYN-ACK segments according to the generation time. When a client's response is being processed, the algorithm iterates over the structure of these lastly generated values and searches for a match between the generated and the value read from the segment. The number of iterated elements depends on the generation period and the validity of generated values. When configured sensibly, this method is faster and allows considerably better throughput than its counterpart.

The second approach is somewhat inspired by SYN Cookies. As illustrated in Fig. 3, a unique hash is computed for every connection. The source IP, a 32-bit secret, source and destination ports, as well as a 32-bit timestamp, are hashed into a 32-bit string. Its 12 least significant bits are replaced with a modulo of the timestamp shifted to 4-second precision. This technique provides a reasonable trade-off between security



(a) Initial session delay. (b) Subsequent session delay.

Fig. 4: SYN session delays (including 3-way handshake).

and performance because the attacker would have to guess 2^{20} possibilities from the hash alongside four different timestamps. Four-second precision protects against a replay attack for $2^{2^{12}}$s $\sim$ 194 days (required for the timestamp repetition). Received value verification is then done by reconstructing the timestamp by deriving its value before the modulo operation was applied. This process involves shifting back to the 1-second precision and computing the hash function for every possible second in the given time window. If the reconstructed timestamp is within the timeout range and the first 20-bits of the computed hash match the first 20-bits of the *SEQ* (*ACK*) in the analyzed RST (ACK), the client is considered legitimate. This method undoubtedly provides stronger security since unique values are generated per connection instead of the single value for all connections in the given time window.

Both policies can further be combined with two security modes – crypto- and non-cryptographic. The cryptographic mode uses cryptographically-secure hashing and number generation, aiming to deny generated values guessing and estimation completely. On the other hand, the non-cryptographic variant utilizes regular hashing and pseudo-random numbers, providing high performance at the cost of lowered security.

V. Results and Closing Remarks

Firstly, the compatibility of the presented methods with modern operating systems has been confirmed. This was done with a browser and various console applications, which were supposed to establish a connection to a server protected by our algorithm implementations. As expected, all three methods always caused the first session establishment to fail, and a client had to react accordingly. Our results show that systems without any IPS activated, namely Windows XP – Windows 10, Linux kernels 3, FreeBSD 11, Apple iOS 12, macOS 10.14, and higher are all respecting the TCP standard, and thus being fully compatible. All the methods behave transparently to the protected devices, and all OSes tried to automatically re-establish the session, making the methods transparent for user applications as well. However, we discovered that newer Fedora-based distributions utilize *nftables* stateful connection tracking to drop invalid packets. Such configuration effectively prevents the OS from receiving invalid SYN-ACK responses, making it RST Cookies-incompatible.

The client's behavior is fully dependent on the used method because it defines *how* the session fails. The following paragraphs will examine these behaviors, whereas Fig. 4 summarizes the initial and subsequent session establishment delays for an average of 10k *netcat* data transfers on CentOS 7.8.

SYN Drop causes the session to fail by dropping the first received SYN. The time of its retransmission depends on the client's TCP stack, influenced either by an operating system or an application. The most frequent value we came across was 1000ms (Fig. 4a), but other values may also be present [20].

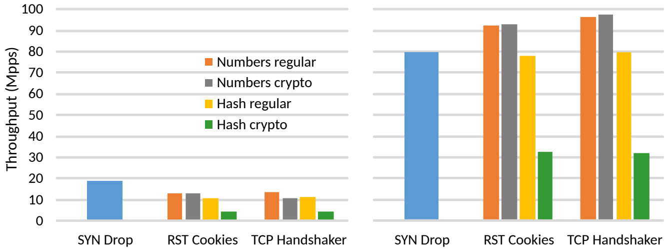
RST Cookies brings a session into an erroneous state by an invalid ACK. In this case, the client is supposed to reply with a RST segment and try to reestablish the session on the same port. This process is also application and host-dependent. The typical period we encountered was also 1000ms (Fig. 4a) on Windows OSes and most *nix (including Android and iOS) applications and popular browsers (Chrome, Edge).

TCP Handshaker closes the first client's session with a RST. In this case, the session needs to be reestablished on a different source port, which OSes and simple programs typically do not perform. However, more robust applications tend to open a new port automatically after a certain period. Since this process requires OS kernel intervention, the initial session delay is slightly higher as in the previous methods (Fig. 4a).

While simple programs like netcat rely on an OS's TCP stack and a single port, more sophisticated applications usually initiate multiple TCP connections at once for a single user request. Two to four sessions are opened initially, followed by another after 200-300ms (e.g., Chrome). Although the retransmission period of 1s is rather high, additional attempts after 200ms are typically sufficient to set up a TCP channel since the client's IP address is already whitelisted. Such applications are thus able to successfully establish the connection without waiting for a retransmission timer. Fig. 4b shows that subsequent whitelisted connections experience no significant delay (<0.2ms). We evaluated all the values for the worst-case scenario using cryptographic hashing. Therefore, simpler security mechanisms like non-cryptographic random number generation tend to reduce these delays slightly. Though delays up to 1s may seem high, it is important to realize that these methods are activated only when an ongoing attack is detected. Therefore, no delays occur during a regular operation, and a slight initial delay is highly preferable to service unavailability while an attack is in progress.

Memory requirements and packet throughput shall be considered as well. All the presented methods require a whitelist data structure to monitor SYN-sending IP addresses and store state information for decision-making. RST Cookies and TCP Handshaker require 20B per whitelisted client if SYN limiting is enabled. For example, peaks on the CESNET's network from March to August 2020 reached up to 540k TCP flows per second. For this purpose, only 640 MiB of memory would be needed for 33.55M client entries, hence containing all of them for a period of one minute. Although the SYN Drop method requires only 13B per client, it needs to store state information for all (also spoofed) clients.

Packet throughput is mostly influenced by the number of processed hash functions. Our implementations contain at least one hashing per TCP segment to access the whitelist. When the hashing security mechanism is used, two hashes per SYN and up to five per RST (RST Cookies) or ACK (TCP Handshaker) segments are needed to validate them. Fig. 5a shows the



(a) Single core / thread.

(b) 8 cores / threads.

Fig. 5: Mitigation methods performance comparison.

throughput comparison of these mitigation methods during the simulated SYN Flood attack. The attack was composed of SYN packets originating from randomized IP addresses and ports. As expected, the SYN Drop method performed the best, achieving a throughput of 29.69Mpps. Usage of more sophisticated mitigation furthermore reduced the throughput to less than 15Mpps according to the used security mode.

Modern network interface cards support a mechanism called RSS (Receive Side Scaling), enabling to distribute packets into multiple receive queues, which can be processed by separate CPUs. Therefore, we commonly run numerous instances of these algorithms on multiple CPU cores. This way, a throughput of more than 97Mpps (∼71Gbps) on 8 independent queues can be achieved (Fig. 5b). In this case, the packet rate of more robust methods is higher than of the SYN Drop. This is due to the nature of mixed IPv4/IPv6 segments with different lengths, which have to be forwarded as they are. For this reason, the TX interface buffers are used inefficiently, and so the overall performance decreases. On the other hand, our custom-generated SYN-ACKs are padded, so the performance for these high-speed packet rates may be significantly higher.

Alongside legitimate and attack data, RST Cookies and TCP Handshaker have to cope with responses to the traffic they generate as well. Its volume could become rather significant as the clients not contained on whitelists try to establish new connections. For this reason, we also provide throughput comparisons with a changing vector of clients' RST (ACK) messages in contrast to the received number of SYNs. Note that SYN analysis requires one memory access for whitelist search and an alternative SYN-ACK generation if the client is not verified. In contrast, RST/ACK analysis requires to validate their *ACK* value, either by memory access for the random numbers variant or up to four extra hash computations. As expected, only hashing security policies are affected since memory access is negligible when compared to hashing.

For example, consider a RST:SYN ratio of 0.1 for non-cryptographic RST Cookies hashing and suppose that all the RSTs are destined to the mitigation mechanism, the throughput for 1 thread is 10.4Mpps. If the ratio increases to 1.0 (all SYN senders need to verify themselves with a certain RST), the throughput falls to 9.6Mpps. Even more significant decline can be observed for the cryptographic hash policy, which falls from 4.0Mpps to 2.3Mpps per thread. Since TCP Handshaker uses the same mechanism for ACK generation and validation, its results are quite the same as the cases described here.

VI. Conclusions

This paper has focused on the analysis of the TCP SYN Flood attack and discussed three network-based mitigation methods as its possible countermeasure. Presented experimental results are based on a custom implementation and evaluation with commonly used operating systems and applications.

The simplest method, SYN Drop, offers sufficient protection against pure SYN Floods from regular or spoofed IP addresses. Advanced algorithms – RST Cookies and TCP Handshaker, allow detection and blocking of more sophisticated attacks, able to bypass conventional techniques by simulating the traffic of a real client. The mitigation can function on up to 97Mpps, but prolongs the establishment of the first session.

Further optimizations for better performance and memory requirements can still be conducted. Used memory can be minimized by probabilistic data structures such as Bloom filters. Appropriate hash functions could also improve the overall throughput significantly. Even bigger performance demands may lead to offloading a part of SYN Flood mitigation algorithms into the FPGA device programmable dataplane.

Our future work will focus on observation and analysis of attack vectors of real-world situations, given our experience with DDoS protection solution deployment in operational environments at CESNET's backbone and NIX.CZ.

References

[1] Kupreev *et al.*, "DDoS Attacks in Q2 2020," Kaspersky Lab, Tech. Rep., Aug 2020, https://securelist.com/ddos-attacks-in-q2-2020/98077.

[2] Cisco Systems, "Cisco Visual Networking Index: Forecast and Trends, 2017–2022 White Paper," Tech. Rep., Jan 2017.

[3] CESNET a.l.e., "DDoS Protector," Sep 2020, https://www.liberouter.org/technologies/ddos-protector/.

[4] B. Harris and R. Hunt, "TCP/IP security threats and attack methods," *Computer Communications*, vol. 22, no. 10, 1999.

[5] P. A. Watson, "Slipping in the Window: TCP Reset Attacks," Jan 2004.

[6] J. Postel, "Transmission Control Protocol," RFC 793, IETF, Sept 1981.

[7] W. Eddy, "TCP SYN Flooding Attacks and Common Mitigations," RFC 4987, IETF, Aug 2007.

[8] J. Lemon, "Resisting SYN Flood DoS Attacks with a SYN Cache," in *Conference on the BSD operating system (BSDCon)*, 2002.

[9] D. J. Bernstein and E. Schenk, "SYN Cookies proposal," Sept 1996, http://cr.yp.to/syncookies/archive.

[10] L. Ricciulli, "TCP SYN Flooding Defense," in *Communication Networks and Distributed Systems Modeling and Simulation (CNDS)*, 1999.

[11] W. Simpson, "TCP Cookie Transactions," RFC 6013, IETF, Jan 2011.

[12] Y. Cheng, J. Chu *et al.*, "TCP Fast Open," RFC 7413, IETF, Dec 2014.

[13] P. Fergusson and D. Senie, "Network Ingress Filtering: Defeating Denial of Service Attacks," RFC 2827, IETF, May 2000.

[14] Baker, F. and Savola, P., "Ingress Filtering for Multihomed Networks," RFC 3704, IETF, Mar 2004.

[15] W. M. Eddy, "Defenses Against TCP SYN Flooding Attacks," *The Internet Protocol Journal*, vol. 9, no. 4, Dec 2006.

[16] C. L. Schuba, I. V. Krsul *et al.*, "Analysis of a Denial of Service Attack on TCP," in *Symposium on Security and Privacy (SP)*, 1997.

[17] O. Osanaiye *et al.*, "Distributed denial of service (DDoS) resilience in cloud," *Journal of Network and Computer Applications*, vol. 67, 2016.

[18] C. Li *et al.*, "Detection and defense of DDoS attack-based on deep learning," *International Journal of Communication Systems*, 2018.

[19] Linux Kernel Mailing List Archive, "T/TCP: SYN and RST Cookies," Apr 1998, https://lists.gt.net/linux/kernel/12829.

[20] V. Paxson, M. Allman, J. Chu, and M. Sargent, "Computing TCP's Retransmission Timer," RFC 6298, IETF, June 2011.