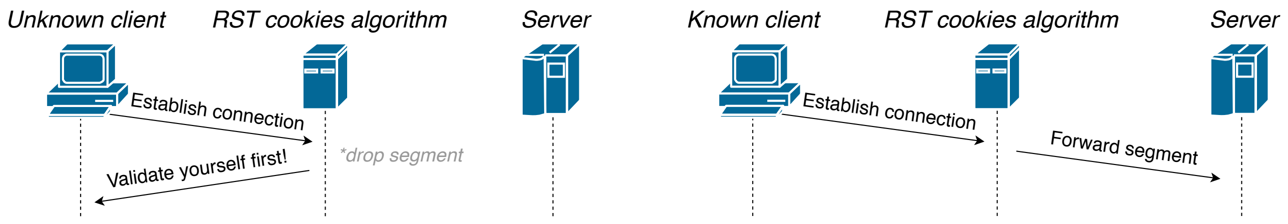
BRNO UNIVERSITY OF TECHNOLOGY FACULTY OF INFORMATION TECHNOLOGY

Excel@FIT 2019
http://excel.fit.vutbr.cz

TCP Reset Cookies – a heuristic method for TCP SYN Flood mitigation

Patrik Goldschmidt*



Abstract
TCP SYN Flood is one of the most widespread DoS attack types used on computer networks nowadays. As a possible countermeasure, this paper proposes a long-forgotten network-based mitigation method TCP Reset Cookies. The method utilizes the TCP three-way-handshake mechanism to establish a security association with a client before forwarding its SYN data. Since the nature of the algorithm requires client validation, all SYN segments from spoofed IP addresses are effectively discarded. From the perspective of a legitimate client, the first connection causes up to 1-second delay, but all consecutive SYN traffic is delayed only by circa 30 microseconds. The document provides a detailed description and analysis of this approach, as well as implementation details with enhanced security tweaks. The project was conducted as a part of security research by CESNET. The discussed implementation is already integrated into a DDoS protection solution deployed in CESNET's backbone network and Czech internet exchange point at NIX.CZ.

Keywords: TCP Reset Cookies — TCP SYN Flood — DDoS mitigation

Supplementary Material: Demonstration Video

*xgolds00@stud.fit.vutbr.cz, *Faculty of Information Technology, Brno University of Technology*

1. Introduction

Transmission control protocol (TCP) is an integral part of the Internet protocol suite. It is a component of underlying architecture which provides functionality for services like HTTP, FTP, SMTP and many more. As the TCP importance is fundamental for the operation of the Internet, it is often the target of various cyber-security threats, Distributed Denial of Service being a popular choice. Report from Q4 2018 by Kaspersky Lab states that the most frequent target of a denial of service attacks was TCP, targeted by 66.60% of all the attacks [1] (Figure 1). According to [2], the number of DDoS attacks will double to 14.5 million p.a. by 2022. These and many other facts should be convincing enough to understand that TCP is worth protecting.

Figure 1 also shows that most of the attacks on TCP are performed by SYN flooding. Details of this weakness and existing solutions to prevent it are discussed in Section 2.

The purpose of this paper is to present a mitigation strategy called TCP RST cookies. This method is not as widespread as TCP SYN cookies but is way more effective in certain situations, such as when dealing with spoofed IP addresses. Implementation of this method is also more suitable for IPS systems. With the use of data structures posing as IP whitelist and blacklist, our specialized implementation also supports SYN limiting, creating an especially strong mitigation mechanism even against more sophisticated attacks.

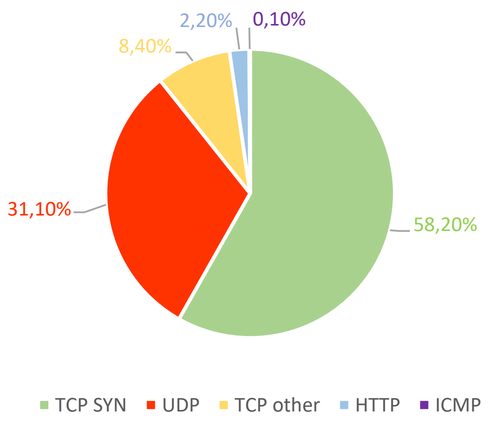


Figure 1. Distribution of DDoS attacks by type, Q4 2018 by Kaspersky Lab. [1]

Implementation of the presented method is already part of the anti-DDoS solution for high-speed networks developed by CESNET. The solution is actively used on CESNET's backbone network and was also recently applied to the Czech national Internet exchange point at NIX.CZ [3]. More information about this and our other projects can be found on www.liberouter.org.

2. TCP Security Considerations

2.1 Theoretical background

To create and maintain a reliable way of communication, TCP implements several techniques for node synchronization. The communication process starts with a three-way-handshake, process during which both peers negotiate transmit options and establish a communication channel (Figure 2). Various TCP threats try to take advantage of this particular process.

The 3-way-handshake process is started by an initiating host (client). This client sends a segment with the SYN flag set and generates a pseudo-random value of x to be used as the Sequence number (SEQ) while setting the Acknowledge number (ACK) to 0. The receiving host (server) then generates its own SEQ y, acknowledges the received traffic by setting its ACK to received segment $SEQ + 1$ and sets SYN and ACK flags. The client also acknowledges a received segment, and the setup process is completed. From this moment, the client and the server may exchange data.

2.2 Known vulnerabilities and attacks

Attacks that abuse weaknesses of the TCP can be classified as either flood or injection attacks. Flood attacks typically target a single host or a network. Their aim is to exhaust the target's resources by flooding bogus packets, making it inaccessible for regular clients and creating a denial of service. On the other hand, injection attacks are based on eavesdropping the ongoing 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]. This document focuses on the flood attacks, which are typically associated with a DoS.

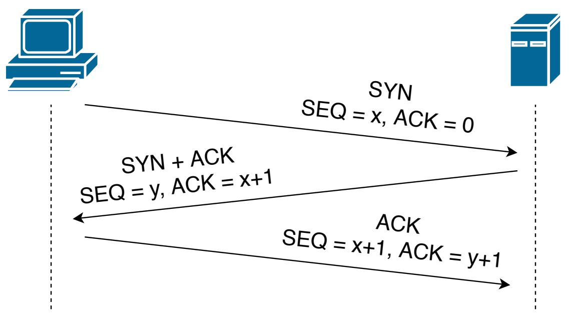


Figure 2. TCP three-way handshake process.

The functionality of the most popular attack – TCP SYN flood depends on the 3-way-handshake mechanism, during which server receiving the SYN message responds with an SYN-ACK segment and waits until the ACK arrival 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 described in RFC 793 as Transmission Control Block (TCB) data structures. TCB structures are used to store necessary state information for an individual connection. They may be implemented differently among the operating systems, but the key concept is that a new memory needs to be allocated upon every new TCP connection [6].

Operating system kernels typically try to protect host memory from getting exhausted by implementing a limit of contemporary TCB structures called backlog. When the backlog limit is reached, either incoming SYN segments are ignored, or uncompleted connections in the backlog are replaced. As illustrated in Figure 3, the primary goal of SYN SYN flooding is to exhaust the target's backlog with half-open connections. For this purpose, spoofed IP addresses that do not generate a reply to SYN-ACKs are often used.

Other attacks include various type of floods, such as SYN-ACK, ACK/PSH, ACK fragmentation, RST, and FIN flood. More sophisticated techniques, such as Fake session, combine several attack vectors. This type of attack includes ACK and FIN segments alongside many SYNs, simulating traffic of a real client. In this case, detection mechanisms often fail to recognize the ongoing attack. Another technique is Session attack, which utilizes a botnet to create a lot of valid TCP connections at once and stretch them as long as possible, making the victim server inaccessible [7].

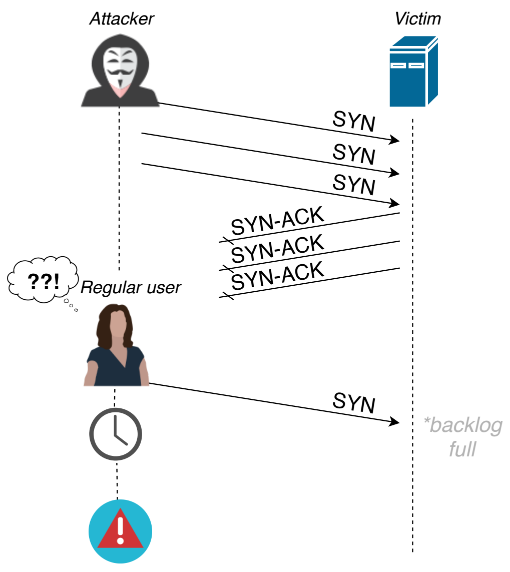


Figure 3. TCP SYN flood attack.

2.3 DDoS mitigation techniques

Modern operating systems provide relatively large backlogs, so they are less vulnerable to regular SYN flooding attacks. However, backlogs can not cover distributed variants of this attack, so specialized methods are still needed. 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 of the IP address, port number and secret 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 required. A device implementing this method is, therefore, able to queue more requests, becoming harder to exhaust. In the BSD kernel from 2002, this optimization reduced the size of the per-connection data by 78 % while allowing up to 15359 entries [8].

In contrast to SYN cache, SYN cookies method does not need to store any state information at all, requiring no memory per-connection. Essential data defining the connection, alongside with a timestamp and a secret are hashed into a 32-bit value representing the *SEQ* number of the SYN-ACK segment. As depicted in Figure 2, the handshake is finished with an ACK message carrying the received SYN-ACK value $+ 1$. Upon ACK receipt, the server can reconstruct original SYN parameters and successfully establish a connection. The method is exceptionally effective against SYN floods, but its nature denies SYN-ACK retransmission and restricts usage of the high-performance options, such as TCP window size [9].

A little-used, but still interesting approach is *TCP Random drop*. The fundamental principle of this method is to replace a random pending half-open connection when the TCB queue is full, and another SYN is received. Connection replacement is done by sending an RST segment, discarding corresponding TCB structure and allocating a new one for the incoming connection. Legitimate clients dropped with RST are expected to try to reestablish a connection again. The rationale for this approach is that by making 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 effective, presented end-host mitigation techniques are not suitable in all cases. Some of them could be implemented as a part of intermediary device software, but their usage would require a mapping between different *SEQ* and *ACK* numbers, making their operation highly ineffective. Therefore, other specialized methods to mitigate DDoS attacks are used.

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

Other potential ways for protection against TCP DoS attacks include network-based countermeasure techniques. The simplest of them is *traffic filtering* [13]. Its fundamental idea is to deny all incoming traffic from IP addresses that do not match their source network prefix (packets intentionally crafted with false IP). This process would allow discarding all traffic from spoofed IP addresses, but it indeed requires to be implemented by all ISPs, which cannot be guaranteed.

SYN flood attacks in modern networks are mitigated by firewalls, proxies or IDS/IPS systems, which usually use initiator SYN-ACK spoofing or listener ACK spoofing techniques as described in [14]. Nevertheless, SYN-ACK spoofing does not solve the previously mentioned problem with degraded performance due to required SYN/ACK values mapping. ACK spoofing may protect the server's backlog but distributed SYN flood may still cause network congestion or high processor utilization of the security intermediary device or server itself, making it unreachable. One barely-known, but very secure method is *TCP RST cookies*, which will be discussed in Section 3.

In real-world situations, both end-host and network-based solutions are frequently employed, and they generally do not interfere when used in combination [15]. Current trends in DDoS mitigation also utilize cloud technologies (e.g. Cloudflare[1]) instead of traditional IDS/IPS systems, but the mitigation principles stay mostly the same as those described in this document.

[1] https://www.cloudflare.com/

3. TCP RST Cookies - Introduction

The first mention of TCP Reset Cookies approach can be traced back to 1996 according to citation in [10]. Unfortunately, the method was never officially published, and the original proposal was only in the form of e-mail communication. The approach was never popularized because it was not compatible with Windows 95 clients [16] and computer networks were much slower, so execution of the method had created unacceptable delays. Mentioned e-mail communication was probably deleted, and so only a few resources about this approach 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.

TCP Reset cookies is a heuristic DDoS mitigation technique, which utilizes the three-way-handshake mechanism and relies on the client’s behavior as defined in the RFC 793. The main idea is to establish a security association with clients before allowing their connection requests. This is achieved by intentionally crafting an invalid SYN-ACK response to the first SYN received from a client. RFC 793, section 3.4 [17] defines the behavior as follows:

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

To distinguish that the RST segment is associated with the receipt of invalid SYN-ACK, RFC 793, section 3.4 [17] also defines requirements on the sent RST:

> *If the incoming segment has an ACK field, the reset takes its sequence number from the ACK field of the segment, otherwise the reset has sequence number zero and the ACK field is set to the sum of the sequence number and segment length of the incoming segment.*

According to these preconditions, we can distinguish legitimate client from the attacker, supposing that regular client will send an RST reply, whereas the attacker will not. When the 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 (Figure 4).

Prior to design and implementation phases, we needed to ensure that modern operating systems are

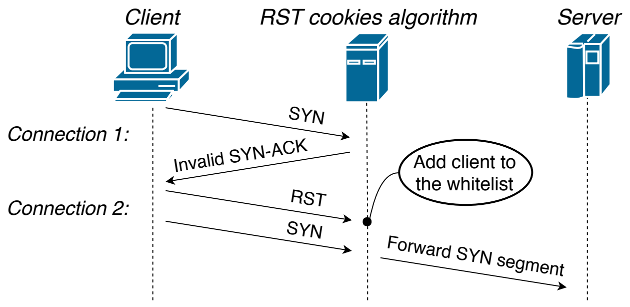


Figure 4. RST cookies functionality.

respecting the TCP standard and send RST segments as a response to invalid SYN-ACKs as expected. Our tests with a prototype have proved, that tested systems, namely Windows XP – Windows 10, Linux kernels 3 and 4, FreeBSD 11, Apple iOS 12, and macOS 10.14 are all compatible with the RST cookies technique.

4. Design and Implementation remarks

4.1 SYN processing

For the correct method functionality, processing of all ingress SYN and RST segments originating from outside of the protected network must be ensured. When the SYN is received, the RST cookie algorithm must determine whether it is from a new client or a client that is already associated. For this purpose, our implementation uses a hash table, IP address being the key. Each entry contains a timestamp specifying when the association has been created (t_a), allowing entries to age. So, upon an SYN segment arrival (t_s), the algorithm has to check if the IP address of the source is contained in the whitelist and its entry timestamp does not exceed the maximum specified age time (t_m), thus validating the following condition:

$$t_s - t_a < t_m$$

If the preceding condition is met, the SYN segment is forwarded into its desired destination. Otherwise, an invalid SYN-ACK is assembled and sent as a response to the processed SYN as mentioned Section 3.

If regular data structures were used, manual removal of the expired entries would be required. However, our hash table provides limited row capacity, and so inserting an element to the full row causes the one with the oldest timestamp to be replaced. This process ensures proper aging mechanism and memory management even without additionally needed interventions.

Our implementation enhances the regular algorithm functionality by adding a counter and a timestamp to the hash table alongside the existing association timestamp. The counter is used for counting SYN segments from the associated clients, and the timestamp denotes the start of a 1-second time window. By using these two extra variables, the algorithm can limit the number of SYN segments sent by already-associated clients. This approach might stop even more sophisticated attacks, which successfully pass through the security association phase. When combined with a blacklist, the ability to detect these smart attackers and deny their traffic completely is available. SYN processing is depicted in Algorithm 1.

```
$entry \leftarrow$ Source IP data from association table;
**if** $entry == NIL$ **then**
    send invalid SYN-ACK;
    Drop packet and **exit**;
**else if** $t_s - t_a < t_m$ **then**
    Delete src IP from association table;
    Send invalid SYN-ACK;
    Drop packet and **exit**;
**end**
**if** *SYN limiting enabled* **then**
    **if** $t_s - t_{entry.window\_start} < 1s$ **then**
        **if** $entry.syn\_cnt \geq SYN\ limit$ **then**
            **if** *Blacklist enabled* **then**
                Add IP to blacklist;
            **end**
            Delete src IP from association table;
            Drop packet and **exit**;
        **end**
    **else**
        $t_{entry.window\_start} = t_s$;
        $entry.syn\_cnt \leftarrow 0$;
    **end**
    $entry.syn\_cnt \leftarrow entry.syn\_cnt + 1$
**end**
Allow packet and **exit**;
```

Algorithm 1: RST cookies – SYN processing.

4.2 RST processing

When an RST segment is being processed, RST cookies must decide, whether the segment is a part of its mechanism or belongs to the regular TCP traffic. This is done by looking at the *SEQ* of the obtained data. If the received *SEQ* is equal to *ACK* sent in the previously generated invalid SYN-ACK, RST is a response to it. In this case, the processed RST segment is dropped and client IP address added to the whitelist. Otherwise, the algorithm forwards the segment to its destination.

4.3 ACK values generation and validation

As mentioned in previous subsections, the algorithm needs to generate an invalid SYN-ACK segment and then match a corresponding RST to it. Generally, invalid SYN-ACK is crafted by setting its *ACK* value differently from $SEQ + 1$ of the SYN it is responding to. Responding with random numbers is possible, but would not allow the process of client verification. The simplest implementation allowing us to validate incoming RST data uses a constant, that is placed in each

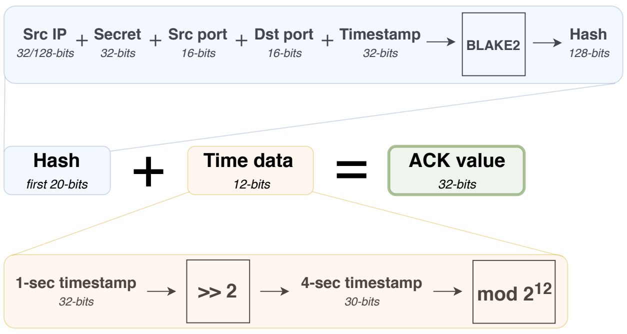


Figure 5. ACK generation – the hashing method.

SYN-ACK response and then checked for the match in the RST. This approach would be functional, but a smart attacker could monitor the traffic and inject the RST packet with a given constant to trick the security mechanisms. To tackle this issue, we propose a system of dynamic ACK generation and validation.

Our implementation of the dynamic ACK generator and validator uses two policies. The first policy generates random numbers periodically and assigns an *ACK* values to SYN-ACK segments according to their generation time. When an RST segment is being processed, the algorithm iterates over the structure of these lastly generated values and searches for a match between the generated *ACK*s and the *SEQ* read from the RST segment. The number of iterated elements depends on the *ACK* generation period and the validity of the generated values. When configured sensibly, this method is faster and allows better message throughput than its counterpart.

The second approach is somewhat inspired by the SYN cookies principle. As illustrated in Figure 5, a unique hash is computed for every connection. Segment source IP, a 32-bit secret, TCP source, and destination ports and a 32-bit timestamp are hashed into a 128-bit string. The first 32-bits are taken, and 12 least significant are replaced with a modulo of the shifted timestamp with 4-second precision. This technique provides a reasonable trade-off between security and performance because the attacker would have to guess 2^{20} possibilities from the hash alongside four different timestamps. Four-second precision was chosen because it would take $2^{2^{12}}$s $\sim$ 194 days to perform a replay attack due to the timestamp repetition. Though not yet implemented, this duration could be prolonged by a pool of secrets, which would prolong the possibility of a replay to the previously calculated value multiplied by their number. To verify a received RST, the algorithm reconstructs the timestamp by deriving its value before modulo operation application, shifting it back to 1-second precision, and computing the hash function for every possible second in the given time window, because perfect precision was lost due to the previous shift. If the reconstructed timestamp is within the timeout range and first 22-bits of the computed hash match the first 22-bits of the *SEQ* in analyzed RST, the client is considered legitimate. This method provides undoubtedly stronger security since unique *ACK* is generated per connection instead of the one value for all connections in the given time window. On the other hand, the latency is slightly increased, and packet throughput is also fairly reduced (Table 1).

To ensure a high level of security, our implementation uses cryptographically secure random numbers, making the estimation of generated *ACK* values rather problematic. The hashed variant uses a one-way cryptographic hash algorithm BLAKE2b. These functions are provided by *libsodium*[2] cryptographic library.

5. Results and closing remarks

5.1 RST cookies in practice

When used on a real network, algorithm behaves transparently for all protected devices. Method blocks all received SYN segments until a security association is established. After this process, the TCP communication is not intervened anymore. On account of this behavior, the protected server does not know about the client's intention to establish a session and thus does not need to allocate any state information.

From the perspective of a client, the first attempt to establish a session always fails. As stated in Section 3, this is not a problem in modern operating systems which send an RST segment and try to reestablish the session. However, the duration of the reestablishment process is host-dependent. Our tests have measured this time to be roughly 250ms on Apple systems, whereas Linux and Windows systems tried to reestablish the session after approximately 1 second. According to these findings, we can conclude that the first connection through RST cookies is delayed by up to 1 second, but all consecutive connections experience no significant delay (Figure 6).

Though measured delay caused by the algorithm seems horrific, it is important to realize that the method should only be active during the ongoing attack. For example, our DDoS protection solution can detect abnormal traffic and turn on the mitigation mechanisms when necessary. Therefore, no delays are caused during regular operation, and when the method is active, a 1-second delay is definitely an acceptable trade-off for service availability during the attack.

When considering an intermediary device running

[2] https://libsodium.gitbook.io

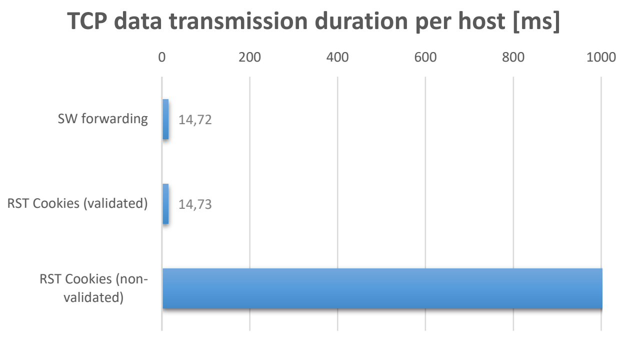


Figure 6. Transaction time performance comparison – Scientific Linux 7.4.

RST cookies, the most significant factors are memory requirements and packet throughput limitations. Usage of hash tables as whitelists plays a significant role in both of them. Every entry in whitelist requires 8B of data for timestamp and an additional 12B if SYN limiting is enabled. Considering 2^{20} whitelist rows, 4 clients per row, we obtain 4.2M client entries taking up to 80MB of memory. This could be considered as a reasonable outcome. In the context of packet throughput, the most significant metrics is the number of processed hash functions. Our implementation contains at least one hashing per TCP segment. When hashed ACK mechanism is used, two hashes per SYN and up to five hashes per RST are needed.

R:S ratio	**Method throughput (Mfps)**		
	SW forwarded	RCks (window)	RCks (hash)
0	17.97	7.30	2.02
0.1	17.33	7.36	2.12
0.2	17.07	7.37	2.24
0.3	16.64	7.40	2.42
0.4	16.52	7.61	2.61
0.5	16.56	7.61	2.77
0.6	16.49	7.62	2.91
0.7	16.54	7.76	3.03
0.8	16.35	7.85	3.22
0.9	16.40	7.87	3.34
1.0	16.48	7.90	3.47

Table 1. Million of frames per second per thread throughput comparison. Based on RST:SYN segments traffic ratio. 1Mfps $\sim$ 680 Mbps.

Table 1 illustrates RST cookies frame processing ability during a simulated attack. In this case, *SEQ* values in RSTs were randomized. When the timestamp of the received *SEQ* does not fit into the specified time window, no hashing occurs, and the segment is straightly forwarded. That is why the actual module throughput was increasing as the ratio between RST and SYN segments was growing. In the real network, legitimate clients would send RSTs fitting into the time window, so at least one extra hashing would need to occur, and actual throughput would decrease.

5.2 Limitations and drawbacks

In addition to memory and throughput, other considerations resulting from the method itself should be discussed. One of them is a situation when the supposedly invalid SYN-ACK segment is accidentally valid. This incident happens when the generated *ACK* value is exactly equal to the $SEQ+1$. In this case, three-way-handshake will continue as usual – the client will try to finish the session with an ACK segment. This data will be forwarded to the server, which will respond with an RST because it received a segment to non-existing session [17]. When the client obtains an RST, it tries to reestablish the session with a new SYN. The probability of this phenomenon is $1/2^{32}$ while having no significant impact on the regular TCP operation.

Another limitation is derived from our specific hash table implementation, which can store only four entries per row. On account of this, a situation when a legitimate client is removed from the table before its age time expires may occur. This happens when the chosen size of the hash table is not respecting the properties of a protected network. Our algorithm offers a statistics logging that may help to detect this situation and adjust the hash table size appropriately.

6. Conclusions

This paper has focused on the analysis of TCP SYN flood attack and proposed a method TCP Reset cookies as its possible countermeasure. The analysis and the discussion resulted in concrete implementation and its application in the production environment.

The RST cookies method provides an efficient way to block SYN flooding attacks from spoofed IP addresses. The basic variant of the method also provides complete protection against pure SYN attacks from non-spoofed addresses. Detection and blocking of more sophisticated variants, able to bypass conventional techniques like SYN cookies, is also available through conjunction with the SYN counter. On the other hand, RST cookies usage significantly prolongs the establishment of the first session and decreases segment throughput to less than half during windowed mode or by 7 to 8-times when hashing mode is used.

To furthermore optimize memory requirements, probabilistic data structures, such as Bloom filters, could be used. However, data throughput would be negatively impacted even more. Experimenting with the small number of hash functions might provide reasonable throughput while keeping the memory requirements exceptionally low.

Since our implementation is a part of the more robust security solution, logic to switch between different SYN flood mitigation strategies is required. To address this issue, we are currently developing an algorithm that monitors network traffic and chooses the most suitable mitigation method according to discovered patterns. This mechanism is based on fitness, analysis, and decision-making cores. The fitness core evaluates SYN flood mitigation algorithms, the analysis core searches for traffic patterns and attack vectors, whereas the decision-making core matches computed ratings with the revealed patterns. This approach is believed to provide an automatic method switching mechanism, which would be able to respond to the dynamic environment of modern SYN flood attacks.

Acknowledgements

I would like to thank my supervisor *Ing. Jan Kučera* for his friendly nature, support, and endless knowledge, which had a major impact on the project quality and its results. Many thanks also go to *Ing. Martin Žádník Ph.D.*, who introduced me to the CESNET's security research projects back in 2017.

References

[1] Kupreev, Badovskaya, and Gutnikov. Ddos attacks in q4 2018. Technical report, Kaspersky Lab, February 2019. Available at: https://securelist.com/ddos-attacks-in-q4-2018/89565.

[2] Inc. Cisco Systems. Cisco visual networking index: Forecast and trends, 2017–2022 white paper. Technical report, 70 West Tasman Dr., San Jose, CA 95134 USA, January 2017. Updated on February 27, 2019.

[3] CESNET. Národní propojovací uzel nix.cz bude testovat ddos ochranu vyvinutou ve sdružení cesnet. Press release (czech), March 2019. Available at: https://www.cesnet.cz/sdruzeni/zpravy/tiskove-zpravy/narodni-propojovaci-uzel-nix-cz-bude-testovat-ddos-ochranu-vyvinutou-ve-sdruzeni-cesnet/.

[4] Hunt R. Harris B. Tcp/ip security threats and attack methods, June 1999.

[5] Paul A. Watson. Slipping in the window: Tcp reset attacks, October 2003.

[6] W. Eddy. TCP SYN Flooding Attacks and Common Mitigations. RFC 4987 (Informational), August 2007.

[7] Riorey. Taxonomy of ddos attacks. (online). Retrieved on 11.03.2019. Available at: http://www.riorey.com/types-of-ddos-attacks.

[8] Jonathan Lemon. Resisting syn flood dos attacks with a syn cache. In *BSDCon 2002*, 2002.

[9] Daniel J. Bernstein and Eric Schenk. Syn cookies proposal, September 1996. (online). Retrieved on 12.03.2019. Available at: http://cr.yp.to/syncookies/archive.

[10] Ricciulli L. and Lincoln P. Kakkar P. Tcp syn flooding defense, 1999.

[11] W. Simpson. Tcp cookie transactions, January 2011.

[12] Cheng Y., Chu J., Radhakrishnan S., and Jain A. Tcp fast open, December 2014.

[13] Fergusson P. and Senie D. Network ingress filtering: Defeating denial of service attacks which employ ip source address spoofing, May 2000.

[14] Christoph L. Schuba, Ivan V. Krsul, Markus G. Kuhn, Eugene H. spafford, Aurobindo Sundaram, and Diego Zamboni. Analysis of a denial of service attack on tcp. In *Proceedings of the 1997 IEEE Symposium on Security and Privacy*, SP '97, pages 208–, Washington, DC, USA, 1997. IEEE Computer Society.

[15] Wesley M. Eddy. Defenses against tcp syn flooding attacks. In *The Internet Protocol Journal*, volume 9. Cisco Systems Inc., December 2006.

[16] Linux Kernel Mailing List Archive. T/tcp: Syn and rst cookies, April 1998. (online). Retrieved on 12.03.2019. Available at: https://lists.gt.net/linux/kernel/12829.

[17] J. Postel. Transmission control protocol. RFC 793 (Standard), September 1981.