Survey of SYN flood attacks and the SYN cookies mitigation technique

Sarigiannis Dimitrios, Sympetheros Alexandros
École polytechnique fédérale de Lausanne EPFL
Lausanne, Switzerland
dimitrios.sarigiannis @epfl.ch
alexandros.sympetheros@epfl.ch

I. Introduction

A. Motivation for studying TCP

In the Transport Layer we have two main protocols: UDP and TCP. UDP is a connection-less protocol using datagrams instead of a stream of data. The benefit of UDP is that it doesn't require a connection to be made for it to exchange data, but simply sends and receives, without handshakes and acknowledgments. It is preferred for real time communication, where a little percentage of packet loss rate is preferable to the overhead of a TCP connection. TCP is a connection oriented stream over an IP network. Before sending or receiving any data, TCP requires that the server and client follow a three way handshake and initiate a connection between the two. Its advantage is that all sent packets will reach the destination and in the correct order, in contrast with UPD which may not deliver all messages and may deliver out of order.

B. Various Existing TCP attacks

In computing, a denial-of-service [4] (DoS) attack occurs when an attacker (client) is trying to make a machine or network resource unavailable to its intended users. Such interesting and well-known attacks that have been studied in the past are: Internet Control Message Protocol (ICMP) flood, SYN flood, HTTP POST DDOS attack, Teardrop attacks, Peer-to-peer attacks, Permanent denial-of-service attacks, Application-level floods, Slow Read attack, Distributed attack etc. During our survey, we are going to study the how TCP works, and after understanding the very basic notions of TCP we will explain in detail the SYN flood case, the possible solutions and final simulate such an attack and the corresponding mitigation.

C. Problem Presentation

To get familiar with the problem, a SYN flood occurs when a malicious client sends a flood of TCP SYN packets, often with a masqueraded sender address, causing the server to open connections, reply to the client and wait for the ACK response. Because the IP address given is incorrect, the ACK response is never received and as a result, the server keeps the connection half open, consuming resources until it creates too many half-open connections and stops being able to open new connections, thus unable to serve future legitimate clients. Some solutions to that problem are at the network level: Filtering, Firewalls and Proxies, Active Monitor and at the end-host level: Increasing Backlog, Reducing SYN-RECEIVED Timer, Recycling the Oldest Half-Open TCP, SYN Cache, SYN cookies and Hybrid Approaches. In our case, we will focus and experiment with the SYN cookies solution.

II. TCP States Description

Before describing all the possible states that a client or a server can be, the type of messages that are used in TCP in order to transfer from one state to another should be mentioned.

SYN: is a synchronize message which is used to initiate and establish a TCP connection but also to synchronize the sequence number between a client and a server.

FIN: is a finish message which is used in order to terminate a connection between a server and a client.

ACK: is an acknowledgment which is used to indicate that a message has been received from its destination.

The states of TCP are [1]:

CLOSED: represents the situation when there is no connection between a server and a client either because it has not been created yet or it has been destroyed.

LISTEN: represents the situation when a server is waiting to receive a SYN message from a client.

SYN-SENT: represents the situation when a client has already sent a SYN message to a server and is waiting for the corresponding SYN-ACK.

SYN-RECEIVED: represents the situation when both of the two sides have received a SYN message respectively to each other, have already sent their own SYN message and waiting for an ACK message to accomplish the connection setup.

ESTABLISHED: represents the situation when we can both of the two sides receive the final ACK message required for the connection to be established. When they reach that state, they can both start sending and receiving data between one another until they decide to destroy their connection.

CLOSE-WAIT: represents the situation when after being on an established state, a client has received a FIN message from a server.

LAST-ACK: represents the situation when a client has already received a close request, acknowledged it, has sent its own FIN message and is currently waiting for an ACK to that request.

FIN-WAIT1: represents the situation when a server waits for a connection termination request from a client or an acknowledgment of the connection termination request previously sent

FIN-WAIT2: represents the situation when a server waits for a connection termination request from the client

CLOSING: represents the situation when a server is waiting for an ACK for its own FIN message.

TIME-WAIT: represents the situation when the server has received a FIN message from the client (and vice versa), and sent its own FIN message and received an ACK for it. After a timeout, both of the two sides, transfer to the CLOSED state.

Figure 1 is a diagram representing all the possible states and the transitions required to go from one state to another.

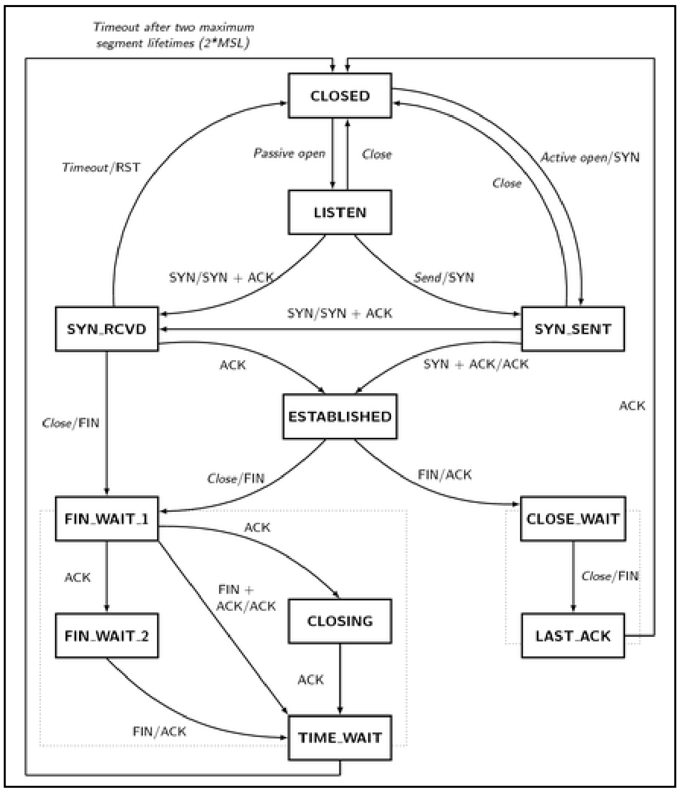

Figure 1. TCP States by Ivan Griffin, 2012. textample.net (CC BY-NC-SA 2.5)

III. SYN FLOOD ATTACK

Unlike other DOS attacks, SYN flood [5] attacks do not rely on overloading the network with resources nor do they attempt to overload the server's processors or memory. Instead SYN flood attacks rely on forcing the server to allocate resources for connections that will never be used and initially block legitimate clients from access to the server.

In the internet, when a client wants to connect to a server, in most cases the server has no previous knowledge of the client and thus, at least in the beginning, cannot distinguish a legitimate client from an attacker. Also, typically operating systems implement the listen part of the TCP connection in a similar manner, specifically by not requiring pre-known knowledge of the client trying to connect, also known as an "unbound" listen. It is assumed that the operating systems implement the connection by allocating space for every TCP SYN segment received and that there exists a limit to the space available. By taking advantage of the lack of previous knowledge of each client and the similar implementation of each operating system, a SYN flood attack is possible.

From the definition of how TCP works, we know that for a connection to be made, it is required to have a TCP three-way handshake. This handshake contains three messages, one message from the client that wishes to establish a connection (SYN), one message from the server responding positively to the request (SYN-ACK) and finally one last message from the client acknowledging that the connection has been made (ACK). The way SYN flood works is by following the above flow of messages, but omitting to send the final acknowledgment and thus keeping the server waiting.

Regarding the implementation, when the server receives a SYN segment at a local port with connection set to LISTEN state, it changes the state to SYN-RECEIVED and initializes its Transmission Control Blocks with the necessary values as well as binds the necessary resources for this connection. After allocating the resources needed for the connection, the server sends an SYN-ACK back to the client and waits for a ACK message in order for it to complete the three way handshake and begin the transmission of data. After sending the request, the attacker will never send an ACK but instead continuously send new SYN requests, forcing the server to open new connections but never truly establish them. It should be mentioned that the SYN message sent from the attacker contains an reply IP address different than that of the attacker, so when the server sends the SYN-ACK, it is sent to a random destination and thus no ACK is ever received. The choice of the IP address is either random, nonexistent or carefully chosen to make the attack much harder to prevent or block.

At first, our intuition will tell us that the server should close such half-open connections after a period of time and block such an attack, but this is not that simple because it is possible legitimate clients have sent ACK and the segments are held up in the network. The server does close half open connections after a certain timeout, but usually this is much larger than the time required for the attacker to send more SYN requests. Because the server cannot distinguish between legitimate clients and malicious, it will continuously open new connections for each request, until it exceeds its available resources. At this time, it will be unable to serve new requests, effectively blocking legitimate clients from connecting to the server.

After understanding what SYN flood is for, it should be pointed out that this type of attack focuses on blocking all new incoming connections to the server, but will not affect previous connections that have already been established nor outgoing connections from the server. Also, it is interesting that this attack focuses on a specific listening port of the server and is not a global attack on the server or the network.

It should be noted that in order for the above barrage of SYN requests to work well, as mentioned previously it is required to carefully change the IP address of the sender(spoofing the IP), in order for the server or the network to be unable to detect the origin of such an attack. Also, it is mandatory to choose appropriate parameters of the SYN request, such the SYN request size and window size. Finally, another important factor is the frequency of SYN messages sent from the attacker, since it should be smaller than the server's timeout for half open connections but also take into consideration possible mitigation strategies of the server.

IV. Mitigation Tactics and Evaluation

There exist two general categories of possible mitigations to deal with this type of DOS attack, one is end-host countermeasures and the other is network based countermeasures [2].

Following we will explain the two categories and some examples in each case, and in the next section will focus on the mitigation of interest, that of SYN Cookies.

The techniques used by the server are:

- Increasing TCP Backlog: This is the simplest solution, since it only requires increasing the server's backlog of connection sockets, increasing the number of connections it can handle. This solution shouldn't be considered sufficient without using other measures as well since the attacker can scale his attack to numbers larger than the backlog can support.
- Reducing the SYN-RECEIVED Timer: Like above, this is also a simple solution that requires the server to decrease the time it waits for an ACK after sending a SYN-ACK. This technique has the disadvantage of discarding many connections from legitimate clients that for some reason took longer than expected to send the ACK message. Also this solution is not practical given that it is easy for the attacker to increase the frequency of the SYN packages it sends and the server cannot decrease the time under a value otherwise it blocks all legitimate requests.
- SYN Caches: This technique is more sophisticated than the previous but is much better at mitigating the problem. It is required to use a hash table internally with a maximum size. The way this technique works is that for every SYN request, instead of opening a connection and storing all the information in the TCB, only a small portion is put in the TCP and the rest is stored in the hash table. When the server receives the expected ACK, it then retrieves the data from the hash table and updates the TCP with all necessary information. If at some point the hash table fills up, the oldest buckets are discarded.
- SYN Cookies: This technique resembles the previous (SYN Cache) but is even more effective. Specifically, when the server receives a SYN segment, instead of creating a connection and TCB, it simply stores the SYN information it requires and sends a SYN-ACK back to the client. In the case that the client responds with ACK, the server then retrieves the SYN information stored and creates the connection. This technique works based on the premises that the ACK sent from the client contains the same sequence number as the SYN, otherwise there would be no way to match which client sent which SYN and ACK. This implementation is very effective for dealing with SYN floods, but there exist a few drawbacks explained later.
- Hybrid Approaches: Finally, there exist hybrid approaches that combine the above solutions based on the circumstances and the operating system choices. Depending on the circumstances, SYN Cookies or Cache may be disabled and enabled when an attack occurs for example.

On the other hand, the techniques used by the network are:

- Filtering: The classical approach that can be done at network level for this kind of problem is to filter packets. This can be done for example by blocking all packets that contain an IP source that doesn't match the source it came from. This technique is effective for some cases, such as when the attacker spoofs his IP, but is ineffective when the attacker uses a distributed set of machines to attack. Also, this technique relies on the intermediate nodes of the network to filter packets and cannot be changed from the server.
- Firewalls and Proxies: Using this approach, it is possible inside the network, for firewalls and proxies to help the server under attack. This can be done either by sending correct SYN ACK's to the attacker using a spoofed IP, and thus hoping for a correct ACK to return to the server. Otherwise it is possible to send spoofing ACKs to the server, in order for it to create the connection and then terminate it.
- Active Monitor: This solutions is similar to the above, as it observes and injects traffic into the network, but differs in that it is not a dedicated firewall but a simpler, easier to configure and usually a cheaper device.

Summing up, most of the network based solutions are highly effective and can be added with little to none changes on the end-hosts. For these reasons, such approaches are generally used and the implemented TCP usually expects some network protection to be available. Nevertheless, the server must provide some protection of it own, and in this case, it usually chooses a combination of the above, depending on the circumstances. Increasing the backlog and reducing the SYN-RECEIVED timer are not very helpful and used in moderation. SYN cache and cookies on the other hand have proven to be very effective. The two techniques differ in that the cache uses a high memory footprint but the cookies may not be 100% compatible with some TCP options and possibly break future TCP extensions. In most cases, SYN cache is enabled by default and cookies are used when needed or when expected to be needed.

V. The SYN Cookies Mitigation

A. *Analysis of SYN Cookies*

As we have mentioned before, SYN COOKIES is a technique that is used to resist SYN Flood attacks. The server uses this technique when the attacker tries to fill up the server's SYN queue with half-open connections, by recording all the SYN messages sent from the attacker or any client in general but not creating a connection. For every such SYN request, the server replies with a SYN-ACK. If at some point in the future, the server receives an ACK from a client, malicious or not, it will now create the connection, using the sequence number in the ACK to retrieve the recorded information for the relative SYN request and reconstruct the SYN request. This system works based on the basis that SYN requests and ACK answers use the same sequence number, otherwise there would be no way to know which ACK belongs to which SYN. In essence it is like the server extends its queue but in practice it does not. Regarding the implementation of SYN COOKIES, this mitigation technique deals with this kind of attack by using cryptographic techniques.

Let us now go more deeply to the cryptographic technique and explain how the server can actually reconstruct the previous SYN message of the client by using these cryptographic techniques. The TCP server constructs the SYN COOKIES by using the initial TCP sequence numbers that it was going to use if it was acting normally to accomplish the TCP handshake with the client. So by constructing the SYN cookies, the server now is able to have its own initial sequence number which will be a little bit different from the client's initial sequence number by following the rules[3] below:

- top 5 bits: t mod 32, where t is a 32-bit time counter that increases every 64 seconds.
- next 3 bits: an encoding of an MSS selected by the server in response to the client's MSS;
- bottom 24 bits: a server-selected secret function of the client IP address and port number, the server IP address and port number, and t.

B. *Drawbacks*

SYN cookies is compatible with all TCP implementations but the fact that the server needs to drop packets with large window size can be considered as a drawback. The other drawbacks of using this technique is that we cannot use TCP options since the server discards the original SYN message, but also equally important, this technique hinders future compatibility with TCP extensions and changes. Moreover, some other vulnerabilities can be observed. For example, if the client's connection-finalizing ACK packet is lost, the application layer requires from the server to act first (for example in SMTP and SSH). Consequently, the client waits for the server's response which apparently never sent from the server because of the server's ignorance about the existing session. Eventually the client will reach a timeout and he will abort the connection, but this timeout is usually too long.

VI. IMPLEMENTATION

A. *Topology*

We assume we have 3 machines, one is the server, one is the client attempting to connect to the server, and finally we have a malicious client (hacker) which has the role of the attacker. These machines are connected via a switch, as seen in the picture below taken from our GNS3 project.

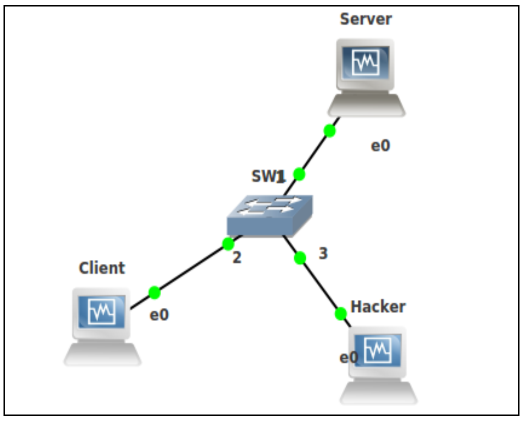

Figure 2. Topology

B. *Configuration*

For IPv4 addressing, we will use the private IPv4 address space 10.10.14.0/24.

- For the local network connecting Server to Switch Port, we use the server's eth0 and the port 1 of the switch. The IP given to the server is 10.10.14.4.
- For the local network connecting Hacker to Switch Port, we use the hacker's eth0 and the port 3 of the switch. The IP given to the server is 10.10.14.5.
- For the local network connecting Client to Switch Port, we use the client's eth0 and the port 2 of the switch. The IP given to the server is 10.10.14.6.

C. *Server Configuration*

Our server is a Slitaz Linux node with minimal configurations besides setting the interfaces correctly. The server runs NodeJS and we have implemented a simple webserver to run on this. Our webserver runs on the port 5000, accepts TCP connections which it stores in a list, and for each connection, when it receives data, will respond with a simple fixed string.
In the beginning we didn't change the value of the SYN_Cookies but later during the tests we tested with and without.

D. *Client Configuration*

Our client is a Slitaz Linux node, also with minimal configurations besides the interfaces.
Our client runs two different python scripts.
One python script constantly attempts to create a new TCP connection with the server, send a hello message and waits for a fixed size string to return. Upon receiving this string it will terminate the connection and rerun the above. The other script keeps the original connection and sends and receives in every loop.

E. *Hacker Configuration*

Our hacker is a Kali Linux node, because it provides hping3 by default, and we set it up with the correct interface configurations.

Our hacker simulates the SYN flood attack by running the following command:

```
hping3 -a 10.10.14.30 -p 5000 -S -flood 10.10.14.4
```

VII. Tests and Results

A. Default Case

In this case, we have the client and the server running only. The SYN_Cookies is set to 1. When running the server and then the client, the connection works is exactly as expected *since no attack is happening.*

B. SYN_Cookies Enabled

In this case we have all three computers active. Because the SYN cookies are enabled, the hacker in unable to disrupt the client-server connection.

During this execution we also test the effect of the cookies on the server CPU and memory. When cookies were disabled and the client was successfully connecting with the server, the CPU was about 11%. When the flood started, the client stopped connecting, but the CPU dropped to 1%, the memory remained approximately the same.

On the other hand, when the cookies were enabled, after running the client and running the attacker, we noticed the CPU spike to above 20%. We didn't see any large changes in memory.

C. SYN_Cookies Disabled

In this case we have all three computers active. Because the SYN cookies are disabled, the hacker disrupts the client-server connection and the client never establishes a connection. During this run, we also tested after how long is the effect of the attack perceived after the flooding has stopped and it was about 2.1 s.

D. SYN_Cookies Disabled and 2 types of clients

In this case we have all three computers active. The client runs two different scripts, one which tries constantly to open a new connection and exchange data, and another that opens the connection once and exchanges data.

There are two different executions of this case. In the one case, the hacker runs before the clients. In this case, both client scripts do not work. In the other case, the two clients start first, then the hacker is run. In this case, the client which constantly changes connection is blocked, whereas the other client with the one connection works correctly. This worked exactly as the theory said it would, since the connection has been made before the attack started.

E. Changing SYN_Cookies on the fly

If we start the server with cookies disabled, and then run the client and the hacker, like before the client will be blocked. If during this execution we change, on the fly, the value of the cookies, after a short period of time, the client will stop being blocked and create connections. When the cookies are enabled, the time it takes for the effect of the attack to vanish is 2.5s.

Likewise, if we start with cookies enabled and on the fly disable them, the client will work and then shortly after will block, likewise this change takes approximately 2s.

F. Number of TCP Connections

One final test we ran is the simple test, with and without cookies, and counted the number of TCP connections at various points. From our tests we saw that with no client and no hacker, the server has zero connections, which is logical. With cookies enabled, we run the client, the server now has one connection. With cookies enabled, we run the client and the hacker, the server now has 256 connections, which is normal since it fills the queue(this is the max backlog size), but the client continues to communicate correctly using one of those connections. Finally, with cookies disabled, we run the client and the hacker, the server now has 256 connections and the client has no connection. This is correct, the queue fills up and the client cannot connect.

We also monitored the state of the TCP connections after the attack, and at various points in time we saw the number of connections falling, from 256 to 215 to 121 to 1 finally..

VIII. CONCLUSION

In summary, we have analyzed the way TCP connections are implemented on servers, how hackers can use this to their advantage and launch SYN_Flood DoS attacks against them and finally the possible countermeasures, focusing on the use of SYN_Cookies. By implementing the above attack in a controlled environment, we were able to see the mitigation of the attack using SYN_Cookies. However, even though this approach mitigates the problem to a certain degree it comes with some drawbacks and vulnerabilities, many of which are known and some that are under investigation, and this technique is not entirely sufficient to block all forms of SYN_Flood attacks without additional measures.

References

[1] J. Postel. RFC 793: Transmission control protocol, September 1981.

[2] Cisco,. 'Defenses Against TCP SYN Flooding Attacks - The Internet Protocol Journal - Volume 9, Number 4'. Web. 8 Jan. 2015.

[3] Eddy, W., "TCP SYN Flooding Attacks and Common Mitigations", RFC 4987, August 2007.

[4] Carl, G.; Kesidis, G.; Brooks, R.R.; Rai, S., "Denial-of-service attack-detection techniques," in Internet Computing, IEEE , vol.10, no.1, pp.82-89, Jan.-Feb. 2006.

[5] Cert.org,. 'CA-1996-21'. N.p., 2000. Web. 8 Jan. 2015..