Chapter 6

TCP-SYN Flooding: Attack Structure and Prevention

6.1 Introduction

The TCP-SYN flooding is the most frequently executed DoS attack [19]. The basis of the attack lies in the design of the *TCP handshake*. In a normal scenario between a *client* and a *server* a TCP session starts with an agreement of session parameters between the two communicating parties. The *client* initiates the process by sending a TCP-SYN packet, requesting the *server* for some service. Upon arrival of the SYN packet, the *server* resources are allocated i.e. a record for *connection buffer*, client-info etc. with the SYN packet header, an initial sequence number is provided by the *client*, a unique number for every connection (used to keep track of data exchanged with the *server*, so missing data can be recognized and handled or keep track of any repeated data received). The *server* then replies with a SYN-ACK packet, notifying the client about the grant of connection with itself. The *server* basically acknowledges the *client's* sequence number and returns information about its own initial sequence number.

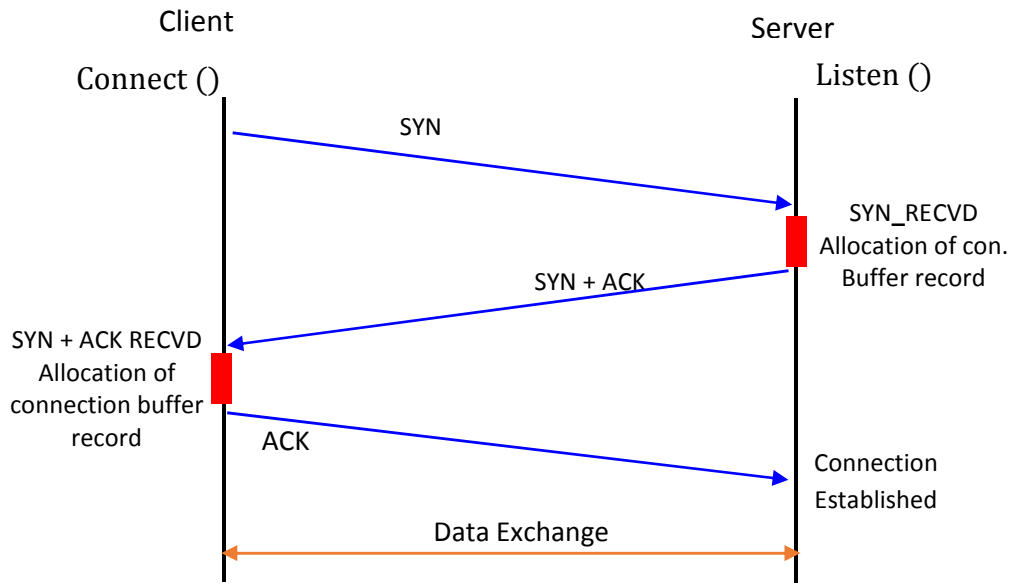

Figure 6.1: TCP connection three-way handshake.

Now the client machine upon the receipt of the SYN-ACK packet creates a connection buffer record. The client then send back the ACK to the server, thus completing the initial set-up of the connection with the server. This process is called a *three-way handshake* and is depicted in figure: 6.1.

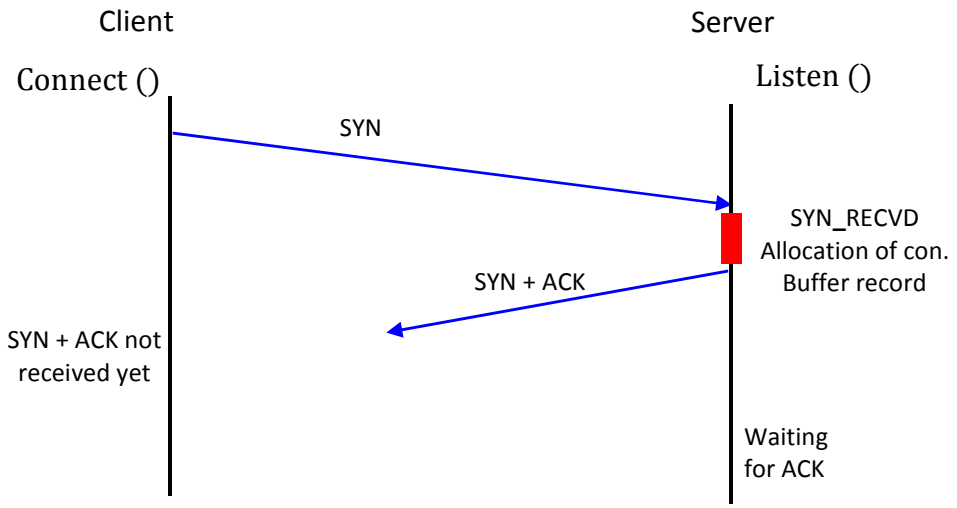

Figure 6.2: TCP connection: *half-open state.*

The promising situation for the exploit lies in the allocation of the server's resources (during connection setup). When the server obliges to the clients request by allocating connection buffer space and send back a SYN-ACK, the connection at this point is said to be half open. The allocated resources by the server are kept occupied for the client-request until an ACK is received from the client, until the connection timeout expires or the connection is re-set (by sending an RST packet) and the connection is terminated by the server after which the occupied resources are released. Now during the half-open connection phase, in the TCP-SYN flooding the attacker generates a huge number of connection requests and uses client IP spoofing (in order to eliminate the risk of being discarded by the server for a multiple connection requests from the same client). The throng of requests in a short span of time exhausts the TCP connection buffer space and sending the server in a state where no new connections can be established. As observed in chapter 3 the already established TCP

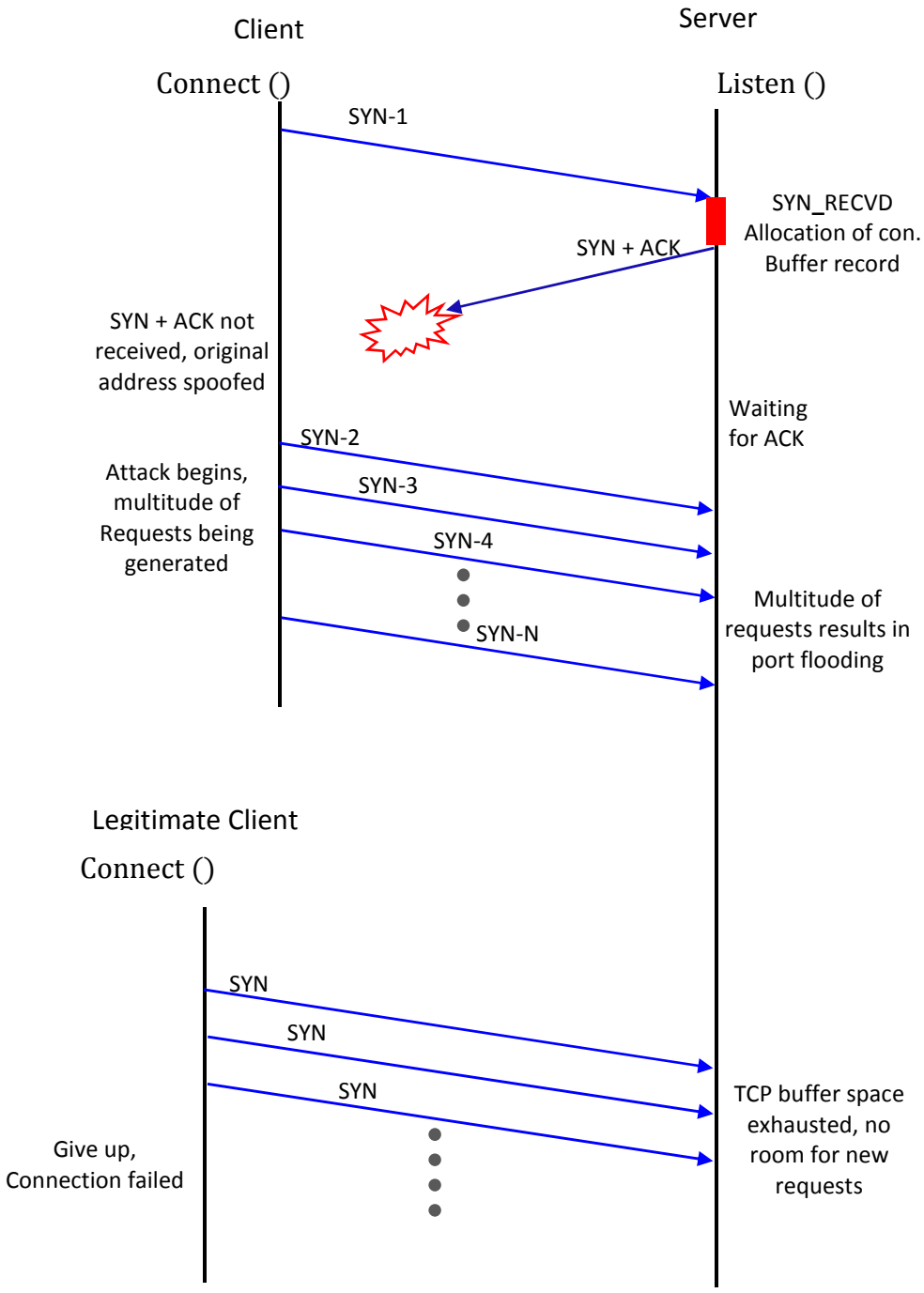

Figure 6.3: TCP SYN flooding during the *half-open state* and the connection request by the legitimate client.

connections experience delay in communicating with the server and in some cases the server may even crash (section 3.3). In order for a successful and a long term attack the attacker needs to send SYN packets at a uniform rate towards the victim. The motive basically is to attack the server and at the same time ensure the server is unaware of any attack going on, if we are targeting a particular listening port. Otherwise an ocean of requests in no time is always welcome. In practice if the attack is launched from a single source, then using IP spoofing is mandatory. While as if a number of compromised hosts (attack from distributed sources) are used to launch a TCP SYN attack, then the attacker does not need IP spoofing. A number of compromised hosts under the control of the attacker (botnet) launches a deadly TCP-SYN attack, where the traffic comes continuously at variable rates. As long as the rate of requesting new TCP-SYNs is greater than at which the TCBs are produced we are assured of a successful attack.

An important thing here to note is the TCP-SYN attack does not basically target a host directly or the network. But rather the attack is aimed at a particular open port or random open ports in the victims system. The initial phase always in a TCP-SYN attack is scanning for open ports on the victim machine. A tool like Nmap [129] gives us the list of open ports on a target machine. After the open port discovery the attacker may launch an attack on a specific port or randomly on a number of ports.

6.2 Common Defense Mechanisms

Does an information system or a network exist today that is completely secure, unbreakable and can never be attacked by hackers or any other thing? Yes such a system does exist, it is the system not connected to any network, wrapped up in concrete and lying deep down at the bottom of the deepest ocean on earth [145]. Needless to mention such a system is of no use. Now if we can't have a complete secure system, what can we have then? Security professionals and practitioners have been trying to figure this out for the past three decades, since the inception of the first computer virus [74] and have come to the conclusion that

the security of a system can't be 100% but a system can try and achieve if not 100% but nearer to that.

Similarly the attacks discussed in the preceding section are unavoidable and can't be prevented 100% but certainly can be mitigated to a certain level where a legitimate user can be assured of the services as promised to be delivered by the information system. Since in the middle chapter 3 we discussed and demonstrated the exploitation of the TCP protocol for the attack on the server, the thesis focuses on mitigating the problem of TCP-SYN only. Before going into the solution, this section will discusses the existing solutions that exist in practice for the prevention of TCP-SYN Denial of Server attacks happening in *Internet* or any other network based resource. Some of the very well-known defence mechanisms [146] that prevent DoS attacks from happening are as follows;

6.2.1 Filtering

Since majority of the TCP-SYN attacks (direct attacks) employ *IP spoofing*, therefore filtering mechanisms are widely deployed in the networks to check for traffic containing spoofed packets. The filtering techniques *Ingress filtering* and *Egress filtering* (147,148 and 149) represent the most common practices for IP address based filtering. *Ingress filtering* is a restrictive mechanism to drop traffic with IP address that does not match a domain prefix connected to the ingress router [19]. *Egress filtering* is an outbound filter, which ensures that only assigned or allocated IP address space leaves the network. *History based IP Filtering* [150] is based on checking a pre-constructed IP address database for the legitimacy of the packets arriving at the router. If a barrage of packets is entering the router, the connection history is checked and if the barrage is not seen anywhere in the history then the arriving packets are labelled as suspicious and appropriate steps are taken to avoid any catastrophe in the network. *Route-based packet filtering* [151] is another filtering mechanism used in IP networks. Route-based filtering uses the route information of the IP packets to filter out spoofed IP packets. The main disadvantage of this filtering mechanism is that it requires universal knowledge of the network topology and which may lead to scalability issues. To overcome the issues with route-based filtering Li et al. proposed

Source Address Validity En-forcement (SAVE) protocol [152]. SAVE constantly propagates messages containing valid source address information from the source location to all destinations. Thus, each router along the way builds an incoming table that associates each link of the router with a set of valid source address blocks [153]. An attacker can duplicate any field in the header of the IP packet but cannot forge the number of Hops an IP packet makes during its journey from source to destination [153]. In other words the TTL (time to live) value in the header of the IP packet cannot be falsified. *Hop Count filtering (HCF)* [154] is based on the value contained in the TTL part of the header. In the Hop Count Filtering method a mapping table is constructed, containing hop-count to IP address mapping. Upon the arrival of the new packet new mapping calculations are carried out at the victim site. The newly calculated values are then compared to the already stored values in the mapping table and the packets whose address is spoofed are identified and discarded in the process.

6.2.2 Shortening SYN-RECEIVED Timer

It's a victim oriented defence technique in which the timer is scaled down when a TCB enters the SYN-RECEIVED state [146, 155 and 156]. The time period is shown in figure 6.1 (red bar on the server side). With this the curtailed-timer will keep the illegitimate SYN attempts from persisting for as long in the backlog and therefore freeing up the resources for legitimate SYN's. Decreasing the timer for TCB's can also prove flawed in some cases, some legitimate connections may be prevented from getting established with the server. Another vulnerability in the defense method is it just needs the attacker to increase the attack rate and make huge number of TCP's flock the server in lesser time periods. For the above reasons this method of dealing with the TCP-SYN attacks is least preferred.

6.2.3 Increasing TCP backlog

TCP Backlog is considered to be the limit of the queue for the incoming TCP connections to the server. If the limit is overflowed with a barrage of illegitimate requests, the result is a TCP SYN attack. Now the mitigation strategy in this case is to increase the number of TCP backlog connection sockets so that the server tolerates the TCP SYN attack and still

is able to provide services to the legitimate users. This step by itself should not be seriously considered as a means to defend against SYN flooding attacks—even in operating systems that can efficiently support large backlogs—because an attacker who can generate large attack segments will most likely be able to scale to much larger orders than the backlog supportable by a host [155].

6.2.4 SYN Cache

In the SYN-cache approach full TCB is not allocated immediately for the incoming TCP connections [157]. But when the opening request is received a minimal state is allocated to the requesting host. The full state allocation is done only when the full connection is established. During the TCP handshake process secret bits are selected from the incoming SYN segments. Hashing is done on these secret bits along with the socket (IP address + TCP port) of the segment. The calculated hash determines a location in the global hash table where the incomplete TCB is stored. There is a bucket limit for each hash value, and when this limit is reached, the oldest entry is dropped [146].

6.2.5 SYN Cookies

SYN-cookie [157,158 and 160] also follows the similar approach of SYN-cache by not allocating the full TCB immediately for the incoming TCP connections. Unlike SYN-Cache no state is allocated at all for all the incoming SYN's, but instead, the sequence number used in the SYN-ACK is filled with compressed information created from the basic data of the connection state. The data is encrypted into the sequence number and transmitted in the SYN/ACK packet. The ACK packet that completes the handshake can be used to reconstruct the state to be put into the backlog queue. One problem with SYN cookies is not able to encode all the TCP options, and the other is that TCP protocol with SYN cookies would never retransmit the unacknowledged SYN/ACK packet [159].

6.3 Attack Prevention Mechanism

The main objective is to differentiate between legitimate packets and the illegitimate packets. In order to separate the attack packets from the normal ones we make use of co-relational patterns. The phenomenon of existence of a co-relational pattern among the various parts of an IP packet header [168] is used in the mitigation technique to thwart TCP-SYN DoS attacks. The mitigation technique separates spoofed IP packets from the normal ones, based on the co-relation between TTL and IP address of the incoming packet and on top of this the destination port number reserved on the victim machine is also included in the mitigation technique. The mitigation technique is inspired from *Hop Count Filtering* [154], as explained in section 6.2.1. Before going into the proposed *Victim Based Statistical Filtering* we first discuss the existing *Hop Count Filtering* and the HCF based techniques used in the prevention of TCP-SYN DoS attacks in the next section.

6.3.1 Related work

Wang et al proposed the TTL based HCF algorithm [154 and 161] that separates spoofed IP packets from the normal ones with capturing rate of 90% of spoofed packets. Their HCF algorithm creates IP to HC mapping table and stores the mapping in an IP2HC table. Upon arrival the packets HC is compared to the HC stored for this IP. If the HC values match, then the packet is legitimate. Otherwise the packet is dropped. Wang et al also analysed the strengths and weaknesses of his filtering method in his work.

Xia Wang et al. [162] proposed a modification in the HC Filtering method. Instead of applying the HCF at the victim site, their technique emphasised on applying the HCF at in-between routers. With this technique they emphasised on not only protecting the victim but the entire network. Their results outperformed the Wang's HCF technique.

StackPi detects the spoofed packets using a routing mechanism known as “path markers”. The StackPi marking scheme consists of two new marking methods that substantially improve Pi’s incremental deployment performance: *Stack-based* marking and *Write-ahead* marking [163]. The scheme performs 2–4 times better than the original Pi scheme in a sparse deployment of Pi-enabled routers.

An implementation of HCF inside the Linux kernel [164] presents a flexible solution against DoS attacks. A hash table is used to construct the IP2HC table in order to hide the IP to HC mapping of every single IP address from the machine

Work in [165] presents a simplistic 3-layer defense mechanism based on web servers against DoS attacks. And at the application layer a traffic limit is used for DoS attacks using legitimate IP. All the transport layer malicious traffic is filtered by the algorithm of SYN Proxy Firewall. A majority of malicious traffic is filtered on network layer using HCF.

Work in [166] proposes a modification in the HCF technique and increases the accuracy of filtering by further 9%. This technique includes all valid HC's seen in the learning phase. This variation enhances the overall accuracy compared to the original HCF and its variations.

The technique presented in [167] works by checking the packets until 'n' malicious packets have been received. Then, 'm' packets are allowed to go unchecked. Their analysis is based on probability of packet arrival 'p', number of malicious packets 'n', and number of legitimate packets 'm'.

6.3.2 Prevention Technique: Victim Based Statistical Filtering

The mitigation framework *Victim Based Statistical Filtering (VBSF)* presented here is basically a variation in the *HOP count filtering* (HCF) technique presented in [161]. The logic behind the hop count filtering is the fact that the packets upon arrival at the victim site do not contain consistent hop count values matching with the spoofed IP addresses. The technique works by analysing the incoming packet's IP address and hop count and if any inconsistency is found, such a packet is deemed as spoofed and discarded accordingly. Hop Count Filtering works by building an accurate IP-to-hop-count (IP2HC) mapping table. The table is built during the *alert* phase, during which every IP packets header is analysed for any abnormality in the TTL field. During this phase certain legitimate packets

may also get incorrectly identified as spoofed ones. Upon DoS attack detection the HCF changes its phases to *action* phase and every incoming IP packet is analysed for any mismatch using the IP2HC table and the mismatching ones are dropped accordingly.

The variation that we are proposing is adding the monitoring information of the usage levels of port numbers on the destination machine, in the *HCF* Algorithm. Recalling from chapter 5 (during the setting up of the DoS attack) the victim machine is scanned for open TCP ports. The attacker either targets a particular port in this list or a group of random port numbers in the list. The point we are making here is that the DoS attack is going to be based on these open ports that resulted from the port scan. The attacker would keep on hitting these port numbers with SYN's and at the same time keep on changing (spoofing) the IP address as well. Now the defense framework *VBSF* presented here will monitor the usage levels of these port numbers. Other than the well-known ports numbers all the hosts IP address requesting to SYN with the server on these port numbers will be suspected as spoofed ones and then a check would be made for the respective port numbers from the server constructed port-frequency-monitoring table and all originating requests from IP addresses trying to SYN with these ports numbers will be dropped accordingly. The server here would analyse the port scan detection [170,171] and maintain a list of open ports that it returned to such requesting hosts of the open port numbers.

In order to separate legitimate packets from the illegitimate ones, *Victim Based Statistical Filtering (VBSF)* employs correlation pattern. The concept of correlation here points out to the situation, when a legitimate packet travels from source to destination, certain number of interior characteristics take place at the same time. Such a pattern can be seen between the requesting hosts IP address and the hop count. For example majority of the users (students) of Kashmir University website "www.uok.edu.in" are from the Kashmir region. During normal days a large number of users visit the website for the service required. During certain days a flash crowd can also be seen. But almost every time the majority (more than 90%) of the traffic is going to come from within Kashmir only, also taking into account the distribution of Global IP addresses. Therefore the Website of

Kashmir University will have more IP packets containing correlations between visits of webpage and the IP addresses from Kashmir. A barrage of traffic originating outside of Kashmir will always be suspicious and is easy to filter out using the HCF technique, but if the gigantic traffic originates from Kashmir only then the VBSF technique comes into play.

6.3.2.1 VBSF Algorithm

The Filtering algorithm extracts hop count, IP address from the IP header and destination port number from the TCP header of every incoming packet. Hop count is the number of router traversals made by the IP packet when it moves from source to destination. Hop count can be calculated using the Time to Live (TTL) field of the IP header. TTL is an 8-bit field in the IP header. The value of TTL is decremented by 1 every time the packet traverses through a router; if the value reaches 0, the packet is dropped by the router. Hop count is calculated by subtracting the currently received TTL value of the IP packet, from the initial TTL of the IP packet, which is set by the Operating System. These initial TTL values are OS dependent and vary from OS to OS. Potential initial values of TTL across various operating systems (variants of Linux and Microsoft Windows) are 30, 32, 60, 64, 128, and 255 [161]. Given the relatively limited hop counts of 1-30 [169] between any two hosts on the internet it's not that hard to guess the initial TTL value. After inferring the initial TTL value, hop-count is calculated by subtracting the final TTL value from the initial TTL value. After this the IP address of the packet is searched in the IP2HC table for the stored hop count of the packet. If there is a mismatch, the packet is deemed as spoofed, otherwise the packet is put forward for further processing. The destination port number is extracted from the packet now and is searched and compared to the port numbers in the port-frequency-monitoring table. If there is a match, the packet is dropped. If there is no match the packet is a legitimate one and is granted entry. Figure 6.4 and 6.5 present the structure of *IP-Header* and *TCP-Header*. The algorithm is as follows:

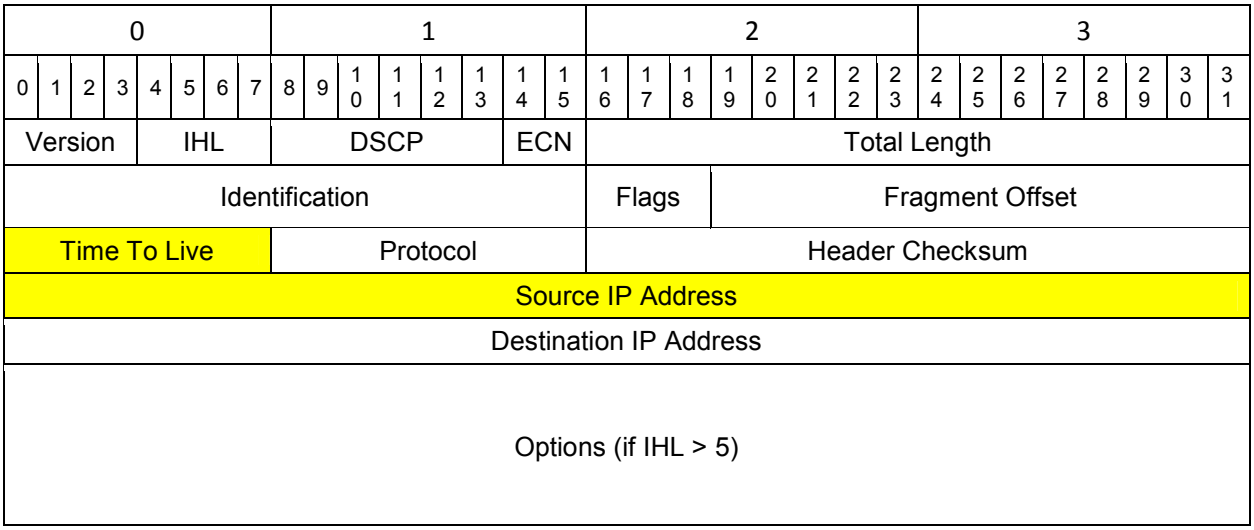

Figure 6.4: TTL and hosts IP address extracted from IPV4 Header.

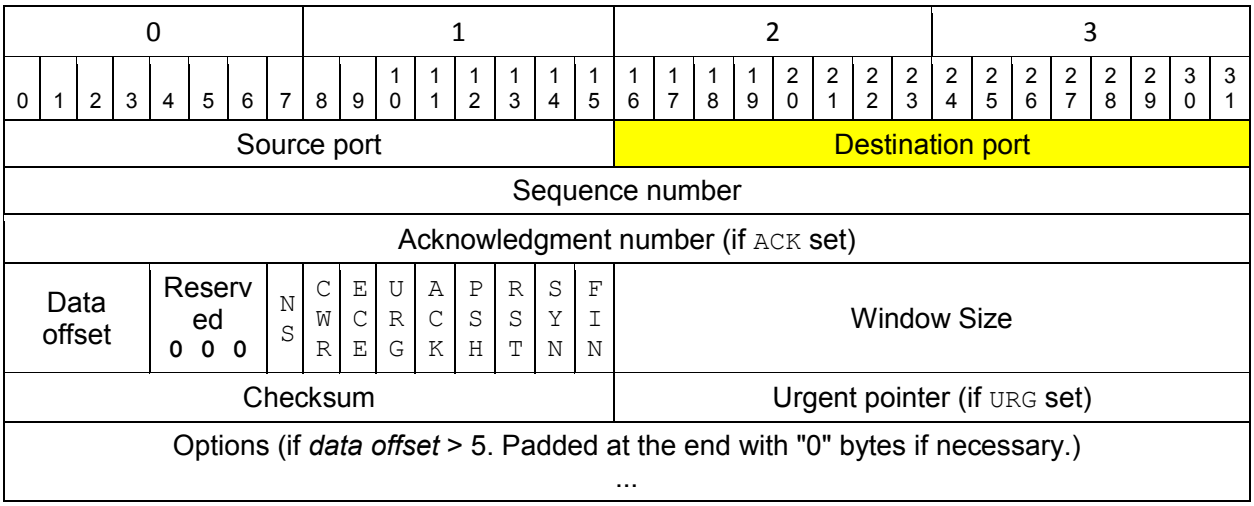

Figure 6.5: Destination port number extracted from the TCP Header.

Algorithm 3 Victim Based Statistical Filtering (VBSF)

Input: IP packet, version 4.
Output: legitimate IP packet or illegitimate IP packet.

1. **for** each IP packet:
2. Extract the f_{TTL}, the IP address S and destination port P_x; // f_{TTL} = final TTL.
3. Figure out the i_{TTL}; // i_{TTL} = initial TTL.
4. Calculate hop count $H_c = i_{TTL} - f_{TTL}$.
5. Search HP2HC table for stored hop count H_s *of host* S.
6. **if** ($H_c \neq H_s$) **then**
7. Drop packet; IP packet spoofed;
8. **else if** ($P_x == P_s$) //P_s stored destination port no.
9. Drop packet; IP packet spoofed;
10. **else**
11. Accept packet; IP packet legitimate.
12. **end for**

end Algorithm

6.4 Empirical Evaluation and Analysis

For the illustration of the *victim based statistical filtering,* a small scale experiment was conducted using a simple network topology given in figure 6.6. The experiment focuses more on the evaluation and analysis to the proposed modification in the hop count filtering method [161], as the HCF part of the algorithm already stands as a well-established filtering method present in the research and industry today.

6.4.1 Experiment Setup

The experiment is carried out under controlled conditions on a Local Area Network consisting of a server, three client computers and an attacker. The configuration of the machines is presented in the table 6.1. The server is the victim machine which is at the receiving end of the traffic generated by the attacker machine.

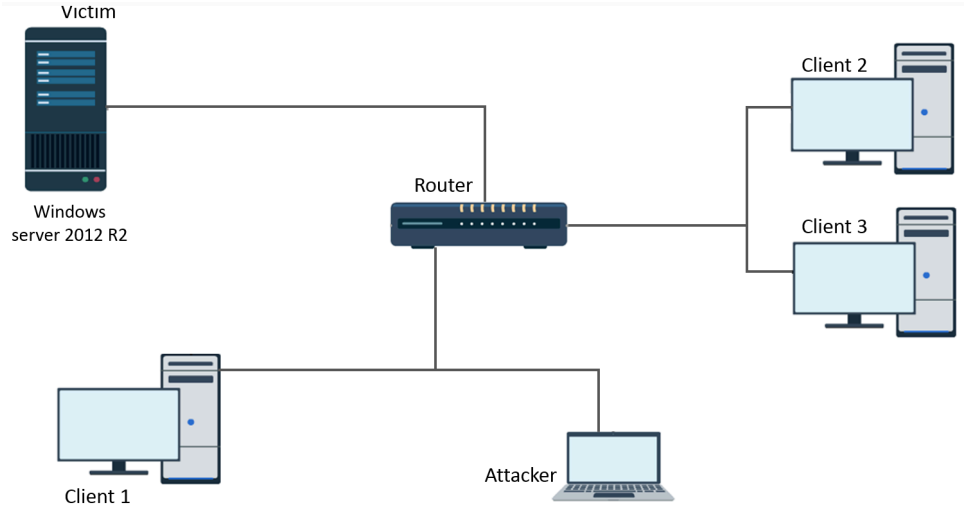

Figure 6.6: Experiment setup.

There are three legitimate clients as well who want to access the services of the server machine. The server is running on *VMware Player V7* , hosted on windows 8.1 machine (6.3 build 9600) with Intel® Core™ i5 2.8 GHz, 4 GB RAM.

Table 6.1: System Configurations used in the Experiment.

Machine	Operating System	Hardware Configuration
192.168.0.10 (victim)	Windows server 2012 R2 (6.3 build 9600)	Intel® Core™ 2 Duo 2GHz, 1 GB RAM
192.168.0.100 (client 1)	Windows 8.1 (6.3 build 9600)	Intel® Core™ i3 2.4 GHz, 2 GB RAM
192.168.0.101 (client 2)	Back Box 4.1	Intel® Core™ i3 2.4 GHz, 2 GB RAM
192.168.0.102 (client 3)	Back Box 4.1	Intel® Core™ i3 2.4 GHz, 2 GB RAM
192.168.0.120 (Attacker Machine)	Kali Linux 1.1.0	Intel® Core™ i5 2.8 GHz, 4 GB RAM

6.4.2 Results and Discussion

The attacker launches the DoS attack from his machine which is running Kali Linux using a socket-stress testing framework *Sockstress.* The following configuration is used to bring havoc on the target machine:

```
./Sockstress -A -c -1 -d 192.168.0.10 -m -1 -Ms -p
,13,19,20,21,23,25,53,80,137,445,32001,32132,32145,48150,48151 ,49157 ,49158 -r
100000 -s 192.168.0.150/175 -vv
```

Sockstress floods the victim with TCP-SYN requests by randomly picking up the port numbers mentioned in the above configuration. All these port numbers (reserved and others) where initially discovered on the victim machine using the port scanner Nmap [129]. For the VBS filtering to work the victim system also has to maintain the list of open ports returned to the calling port scanner software. We did this by using the port scan firewall logs on the windows server and network port scan analyser software wire-shark [172], the ports with maximum occurrence were filtered out and the frequency of occurrence was recorded in the port-frequency-monitoring table. The art of extracting information from the port scan logs on the victim machines can be found in [170] [171]. Coming back to the system configurations used in the experiment in table 6.1 and the experiment setup in figure 6.6, we make the following assumptions:

1. Since we have used various variants of Windows and Linux operating systems, we take the initial TTL as 255 in all the Operating Systems.
2. All the machines (server, legitimate clients and the attacker) are kept in the same network on purpose, to see how the filtering is done if the hop-count and IP addresses values match with the legitimate clients.

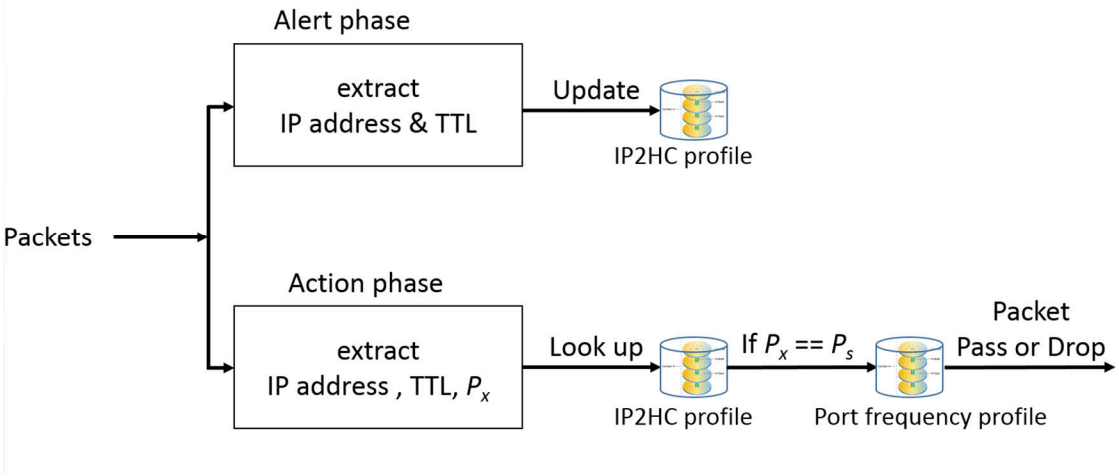

Figure 6.7: Outline of *VBS Filtering* Technique.

Now during the *alert phase* of the HCF method, in absence of the attack the *IP2HC* table looks like as follows:

Table 6.2: *IP2HC* table during the alert phase[1].

Machine	i_{TTL}	f_{TTL}	H_s
192.168.0.100 (client 1)	255	254	1
192.168.0.101 (client 2)	255	254	1
192.168.0.102 (client 3)	255	254	1
192.168.0.120 (Attacker)	255	254	1

During the action *phase* of the HCF method, in presence of the attack the *IP2HC* table looks like as follows:

Table 6.3: *IP2HC* table during the action phase.

Machine	i_{TTL}	f_{TTL}	H_s
192.168.0.100 (client 1)	255	254	1
192.168.0.101 (client 2)	255	254	1
192.168.0.102 (client 3)	255	254	1
192.168.0.120 (Attacker)	255	254	1
192.168.0.159 (Attacker)	255	254	1

[1] f_{TTL}, values were extracted from the packet header using wireshark.

Notice the spoofed IP address of the attacker, changed from 192.168.0.120 to 192.168.0.159 (as configured in the Sockstress, the IP is spoofed randomly between 192.168.0.150-175). In spite of the changed IP address of the attacker the spoofed address still falls in the same subnet as others. Therefore the simple HCF algorithm would treat the traffic from the attacker as legitimate just like others. The VBS filtering is a step ahead than HCF, now after initial policing the packet, the algorithm looks into the port-frequency-monitoring table for any matching destination port numbers. Upon arrival of the packet at the server premises the destination port number P_x is hooked off and then searched for any matching port number (P_s) in the port-frequency-monitoring table. The table 6.4 contains the values:

Table 6.4: Port-frequency-monitoring table[2].

S No.	Destination Port Number	Frequency	Monitoring Time (seconds)
1.	48151	5218	60
2.	32132	3045	60
3.	49157	2824	60
4.	48150	1761	60
5.	49158	647	60

Clearly there is a match and all the entries in the table 6.4 would point the finger of suspicion in the direction of the IP packet of the attacker, because it's from the attackers IP and TCP header that the information matched in the port-frequency table. Such matching packets will be discarded immediately by the *Victim Based Statistical Filtering* Framework. In other cases had the attacker been from a different area the HCF would have shown a different profile in the table and would have discarded the packet much before than reaching for a comparison in the port frequency monitoring table.

[2] Well known port numbers are excluded from the table.

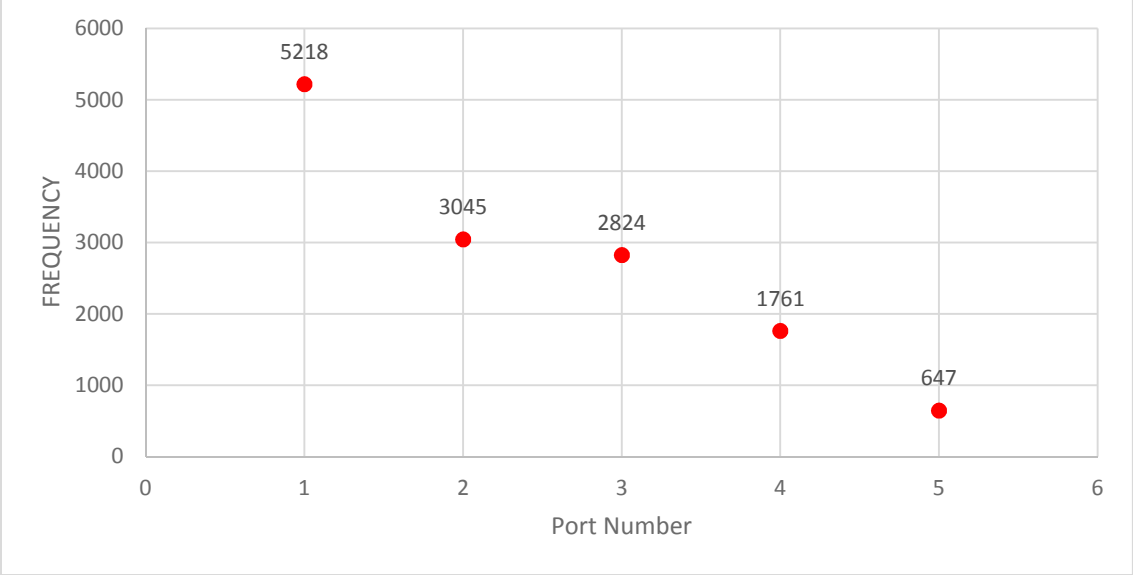

Figure 6.8: Port Frequency Monitoring levels.

6.4.3 Conclusion

With the modification of using the destination port numbers with the HCF method, the Victim Based Statistical Filtering framework shows promising scope to thwart TCP-SYN exploits, but the framework needs to be implemented, tested and analysed in a live scenario first, to see its actual performance over the already effective HCF filtering and other filtering techniques. VBS filtering mechanism relies heavily on two things:

1. Monitoring the activity levels of destination port numbers of the victim.
2. The time quantum under which the activity levels of the port numbers would be analysed.

In order for the VBS filtering technique to show better results than the HCF technique, it's mandatory to extract the port scan logs efficiently from the victim machine and make the data available for storage and further analysis. Generally the victims are very reluctant to show and distribute the information in logs to third parties (for testing purpose) due to their sensitive nature or potential for violation of any privacy laws, but given the dependence on VBS Filtering technique on the information in logs, we need to come up with schemes and

methods that would ensure the security of log information as well as help the information in making filtering of malicious traffic more efficient.

For the time quantum of analysing a port number for activity levels is still an un-answered question in the thesis. Our observation is based on recording the activities for 60 seconds only (just an assumption), while as an optimum time scale needs to be developed keeping in view certain factors like, capturing for longer durations might result in legitimate port use falling in as *false positives* and much smaller time quantum's might result in *false negatives*. Therefore an appropriate time quantum needs to be developed for the successful working of the technique.