American Journal of Science, Engineering and Technology
2017; 2(1): 39-49
http://www.sciencepublishinggroup.com/j/ajset
doi: 10.11648/j.ajset.20170201.17

Science PG
Science Publishing Group

TCP IP Header Attack Vectors and Countermeasures

Vincent O. Nyangaresi, Solomon O. Ogara, Silvance O. Abeka

School of Informatics and Innovative Systems, Jaramogi Oginga Odinga University of Science & Technology, Kisumu, Kenya

Email address:
vincentyoung88@gmail.com (V. O. Nyangaresi), solomon.ogara@gmail.com (S. O. Ogara), silvancea@gmail.com (S. O. Abeka)

To cite this article:
Vincent O. Nyangaresi, Solomon O. Ogara, Silvance O. Abeka. TCP IP Header Attack Vectors and Countermeasures. *American Journal of Science, Engineering and Technology.* Vol. 3, No. 1, 2017, pp. 39-49. doi: 10.11648/j.ajset.20170201.17

Received: November 10, 2016; **Accepted**: January 31, 2017; **Published**: February 27, 2017

Abstract: The TCP IP header has security vulnerabilities that make it prone to numerous kinds of attacks such as TCP SYN flooding, TCP RST, source quench, TCP session hijacking, TCP sequence number prediction, port scanning, CHARGEN and ECHO. The purpose of this paper was to investigate the attack vectors for various TCP IP header attacks and suggest possible countermeasures to curb these attacks. The goals were to gain an understanding on what makes the TCP IP header architecture vulnerable, so that appropriate countermeasures to address these shortcomings could be instigated; based on their performance in terms of their efficiency in curbing the various attack vectors exploiting these vulnerabilities. To achieve this, a combined experimental - simulation approach was employed using Wireshark network analyzer, Nmap, Ettercap, Aireplay-ng and Airodump-ng from Aircrack-ng suite software. A sample network utilizing the transmission control protocol was designed and some packets transmitted over it. The packet traffic volume, sequence numbers, acknowledgement numbers, associated protocols, TCP handshake and packets in flight were then studied. The results obtained indicate that the TCP IP header is indeed susceptible, most probably because the initial intent of the TCP was to share information and security was not a major concern at that time. However, as the internet is now open to the general public and not restricted to the department of defense where it was initially meant to serve, there is need to develop novel algorithms that could help mitigate the weaknesses inherent in the TCP architecture. This study is of help to network designers and administrators as it aids them to identify how to structure their networks for in-depth security by adding another layer of security at the TCP IP header level to support the network-based controls such as next generation firewalls.

Keywords: TCP IP Header, Attack Vector, Vulnerabilities, Countermeasures

1. Introduction

The TCP was initially designed to enable communications within the department of defense (DoD). In these networks, security as not a major a consideration as these were private networks and therefore there was some levels of trust among the communication parties. However, with the advent of electronic commerce, many organizations are moving their business enterprise activities online so that they could take advantage of global market reach, 24 hour economy and efficiency that the internet offers.

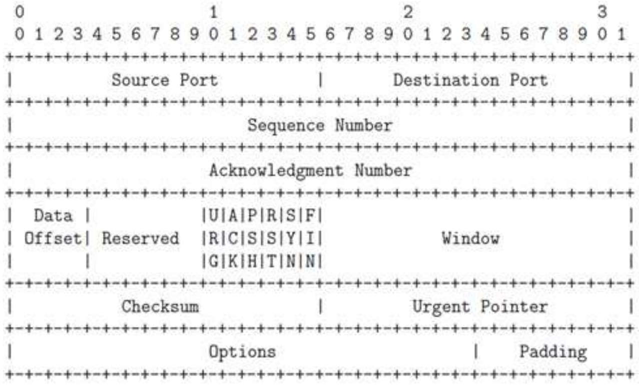
More and more people are getting connected using a variety of devices such as desktop computers, laptop computers, mobile phones, and tablets (Shaneel, 2014). This means that anonymity is on the rise as users do not know who is connecting to them, using what device and for what intentions.

This reduces the level of trust among the various networked entities and increases the attack vectors. Many solutions have been suggested ranging from next generation firewalls, intrusion detection systems, intrusion prevention systems, anti-malware, and anti-spy software among others.

However, all of these can be regarded as addressing the symptoms of the problem and not tackling the real problem, which is the weakness within the TCP architecture itself. In this paper, we take the security challenges and countermeasures at the TCP structure level.

2. TCP IP Header Architecture

To understand how the various attacks against the TCP protocol are rationalized, we first of all study the TCP IP header as shown in Figure 1. This figure shows that the TCP IP header consist of the source port, the destination port, sequence number, acknowledgement number, data offset, reserved field, the various flags (URG, ACK, PSH, RST, SYN and FIN), the window size, checksum,, urgent pointer, options field and the padding field.



```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|          Source Port          |       Destination Port        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                        Sequence Number                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Acknowledgment Number                      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Data |           |U|A|P|R|S|F|                               |
| Offset| Reserved  |R|C|S|S|Y|I|            Window             |
|       |           |G|K|H|T|N|N|                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|           Checksum            |         Urgent Pointer        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Options                    |    Padding    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 1. *TCP IP Header.*

The source port field is 16 bits wide and represents the port that is the originator of the TCP segment while the destination port field is a 16 bit field that is a port of the destination computer which is the recipient of this TCP segment (Avi, 2016). The sequence number and acknowledgement number fields are both 32 bits wide and they play a vital role in connection initialization during the three way handshake procedure, and during payload exchange, to keep track of the byte count in the data streams that are exchanged among the communicating machines.

The data offset field is 4 bits wide and is the number of 32-words in the TCP header. The reserved field is 6 bits wide and as the name suggest, it is set aside for future usage. The TP IP header also includes six control bits, called flags, each of which is 6 bits wide. The URG flag stands for *urgent*. A data packet whose URG bit is set is an urgent data and can therefore act like an interrupt for the ongoing communication between the sender and the receiver. The ACK field means *acknowledgment* and is used to indicate receipt of data packets, signaling data or to indicate packet loss.

The PSH flag stands for *push*, meaning that when it is set, the corresponding TCP segment has to be put on the wire immediately. This is a fruitful requirement for highly petite messages and when echo-back is needed for individual characters. The RST flag means *reset* and when set, the interpretation is that the sender desires to reset the TCP connection. On the other hand, SYN flag stands for *synchronization* and when set, the consequence is that there is need for the synchronization of the sequence numbers. Lastly, the FIN control bit implies *finish* and when set, it means the source needs to lapse the connection.

The window field is 16 bits wide and is employed to specify the upper limit of the number of data bytes that the receiver's TCP is prepared to accept from the sender's TCP in a particular TCP segment. The Checksum field is 16 bits wide and is used for error detection and correction. In his study, Avi (2016) noted that it is computed by accumulating all 16-bit words in a 12-byte pseudo header, the TCP header, and the data. In situations where the data has an odd number of bytes, a padding consisting of a zero byte is appended to the data. The pseudo-header and the padding are not transmitted with the TCP segment.

The urgent pointer field is16 bits wide and is utilized in situations where urgent data is to be sent. In this scenario, the TCP header's URG bit set, and the consequences are that the receiving TCP engine has to momentarily suspend accumulating the byte stream that might be in the middle of and give higher priority to the sent urgent data.

The options field is of variable size. There are two dissimilar window related fields in this TCP header: the window field that is included in the header and the other one which is designated as the congestion window that only appears when data packet loss is detected via the failure to receive acknowledgements for the sent packets within the agreed time confines.

The congestion window field is placed in the *options* field in the TCP header. The window field is employed by the sender TCP and its size is determined by the information received from the receiving TCP whereas the congestion window is set by the sending TCP. In the following sections, the various attack vectors exploiting TCP header architectural vulnerabilities are discussed.

3. Research Approach

This study involved the practical investigation of the various TCP attack vectors. As such, various network monitoring tools (Wireshark network analyzer, Nmap, Ettercap and

Airodump-ng from Aircrack-ng suite) were employed for to achieve the paper objective. These tools included Wireshark and Nmap. The experimental setup consisted of three laptop machines designated as the server, attacker and user (client) as shown in Figure 2.

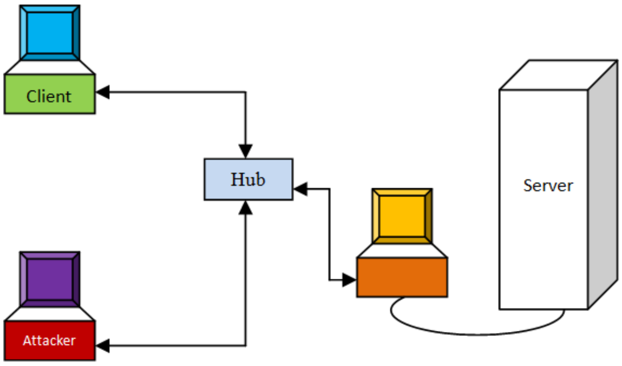


Figure 2. *Conceptual Set Up.*

The client machine was meant to transmit some packets to the server, via the hub. Since the client, server and attacker were linked together on the network through the hub, that attacker, using appropriate software, is able to observe the data being transmitted by the client. Therefore the network monitoring software were installed in the machine designated as the attacker. The practical set up that was employed is shown in Figure 3.

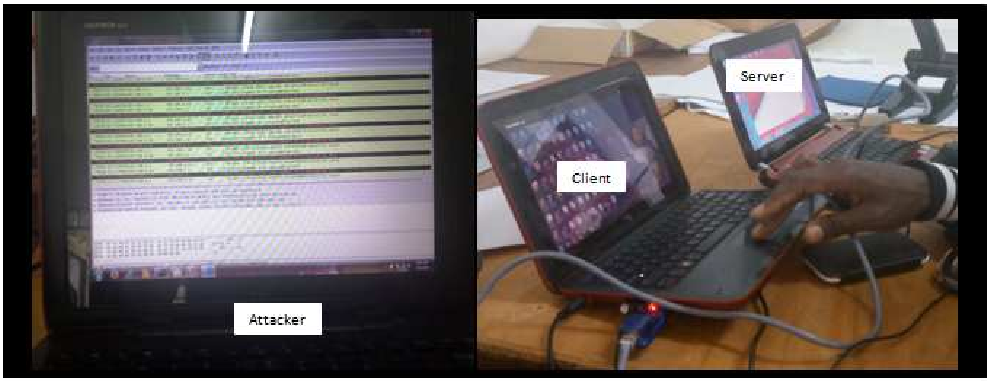


Figure 3. *Experimental Set Up.*

As shown in Figure 3, the machine designated as an attacker had Wireshark network analyzer, Nmap, Ettercap, Aireplay-ng and Airodump-ng from Aircrack-ng suite installed in it for network monitoring. These software were able to capture, insert and analyze packets over the network.

4. TCP Header Vulnerabilities

In this paper, seven attack vectors, Port Scanning, TCP Sequence Number Prediction, TCP Session Hijacking, Source-Quench Attacks, TCP RST Attack, TCP SYN Flooding, CHARGEN and ECHO formed the basis for investigation. The sub-sections below discuss these attacks as well as the possible remedies.

A. TCP SYN Flooding

According to Welian (2010), in TCP communications, all new connections are initiated by having the data source send a SYN segment to the receiver. The SYN segment is a data packet with its synchronization flag bit set. A TCP SYN flooding involves a malicious machine continuously transmitting a TCP SYN segment to every port on the receiver.

The receiver would then reply to the source's TCP SYN request by a SYN-ACK segment from its open ports. This is a data segment whose SYN and ACK control bits are set. However, for the closed ports, the receiver replies with the RST segment. In an ideal three-way handshake, the source responds to the SYN-ACK reply from the receiver with an ACK segment as shown in Figure 4.

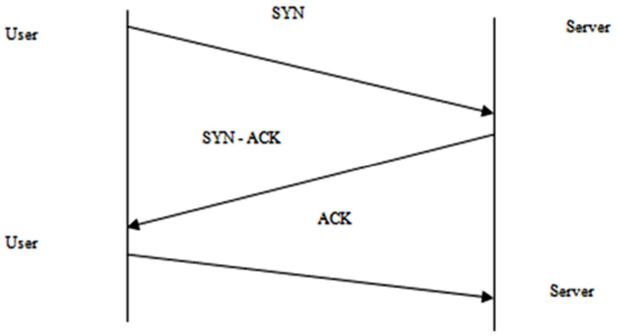


Figure 4. *Normal Three-Way Handshake.*

On the contrary, in TCP SYN flooding attack, a malicious machine never transmits back the expected ACK segment. Instead, once a connection for a specific port times out, another TCP SYN request will appear for the same port from the malicious machine (Guang, 2015).

When a source and a receiver gets into this scenario of a never-ending stream of TCP SYN-ACK segments from the receiver not being acknowledged by the source, there is a an everlasting half-open connections with the target machine as shown in Figure 5.

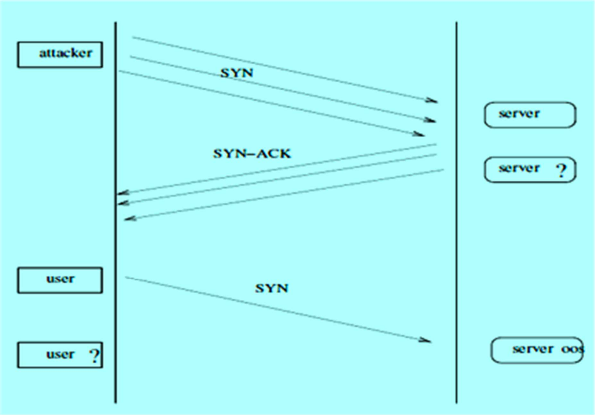


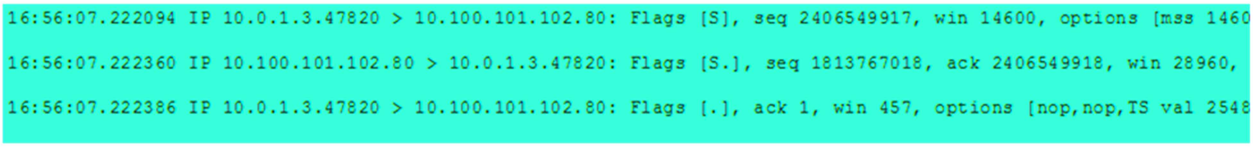
Figure 5. *TCP SYN Flooding.*

Using this attack vector, an intruder can flood the target's buffer that queues incoming requests for half-opened connections. These are connections that have completed SYN, SYN-ACK, but are yet to receive the final ACK from the requesting machine (Mirja, 2015). When this buffer queue is fully occupied by SYN floods, the target machine cannot take any additional connections.

B. TCP RST Attack

This attack can be employed to lapse an already established TCP communication between two target machines. This is common on telnet and secure shell (SSH) connections. Figure 6 that follows shows the Wireshark captured traffic that illustrates the normal way of starting a TCP connection.

In this captured traffic, *{S}* shows a SYN packet that is transmitted by the source machine of IP address 10.0.1.3 to destination, whose IP address is 10.100.101.102. To accomplish this, the source uses port 80. Additionally, {S.} displays a SYNACK packet that the server of IP address 10.100.101.102 sends back to the source machine on the port that the outbound request was originally made on.



```
16:56:07.222094 IP 10.0.1.3.47820 > 10.100.101.102.80: Flags [S], seq 2406549917, win 14600, options [mss 1460
16:56:07.222360 IP 10.100.101.102.80 > 10.0.1.3.47820: Flags [S.], seq 1813767018, ack 2406549918, win 28960,
16:56:07.222386 IP 10.0.1.3.47820 > 10.100.101.102.80: Flags [.], ack 1, win 457, options [nop,nop,TS val 2548
```

Figure 6. *Normal TCP Connection Establishment.*

Finally, {.} shows an acknowledgement (ACK) packet that the source machine sends it to the destination machine on port 80 to signal that it has received the destination machine's ACK packet.

This essentially completes the TCP connection establishment phase. However, if after the connection establishment a reset (RST) packet is sent, this will serve to immediately exterminate the TCP connection and can be represented as shown in Figure 7. In TCP, an ongoing connection in which data transfer takes place between the two end points is in the *ESTABLISHED* state. However, at the start when a network interface is brought up on the local machine, the TCP connection is in the *LISTEN* state. When a local host wants to establish a connection with a remote host, it transmits a *SYN* packet to the remote host. This causes the *about-to-be established* TCP connection to transition into the *SYN SENT* state. The remote host normally responds with a *SYN/ACK* packet, to which the local should send back an *ACK* packet as the connection on the local transitions into the *ESTABLISHED* state. This is referred to as *a three-way handshake*.

On the other hand, if the local host receives a *SYN* packet from a remote host, the state of the connection on the local host transitions into the *SYN RECD* state as the local sends a *SYN/ACK* packet back to the remote. If the remote comes back with an *ACK* packet, the local transitions into the *ESTABLISHED* state. This is again *a three-way handshake*.

Regarding the state transition for the termination of a connection, each end must independently close its half of the connection. Suppose that the local host wishes to terminate the connection first. It sends to the remote a *FIN* packet, (which is the 6th flag bit in the TCP header of Figure 1) and the TCP connection on the local transitions from *ESTABLISHED* to *FIN WAIT 1*.

The remote must now respond with an *ACK* packet which causes the local to transition to the *FIN WAIT 2* state. Now the local waits to receive a *FIN* packet from the remote. When that happens, the local replies back with a *ACK* packet as it transitions into the *TIME WAIT* state. The only transition from this state is a timeout after two segment lifetimes to the state *CLOSED*.

On the contrary, when the remote host initiates termination of a connection by sending a *FIN* packet to the local host, the following happens: The local host sends an *ACK* packet to the Remote host and transitions into the *CLOSE WAIT* state. It next sends a *FIN* packet to remote and transitions into the *LAST ACK* state. It now waits to receive an *ACK* packet from the remote and when it receives the packet, the local transitions to the state *CLOSED*.

Since Wireshark enables intruders to monitor the TCP packet exchanges between the source and destination machines, it becomes possible for the intruder to send a RST packet with the proper values (, that is, a packet whose RST packet flag is set. To accomplish this, a spoofed source IP address is employed such that it matches that of the source machine, 10.0.1.3.

Moreover, proper source port has to be used (port 80 in this case), proper destination IP (address 10.100.101.102 in this case), proper destination port (port 80 for this case). Once constructed, it is launched to break an ongoing telnet connection between the two communicating parties (Robbie, 2015). Since video streaming is highly sensitive to delay and jitter, a TCP RST attack can have severe consequences. An intruder may then be interested in interrupting the TCP session established between the target and the video sharing website.

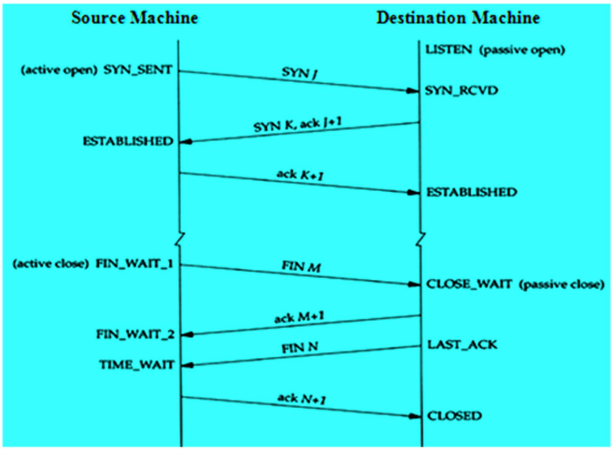
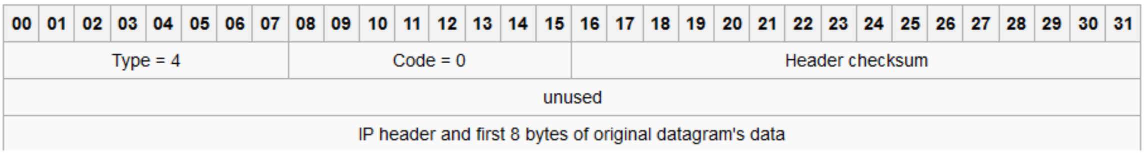


Figure 7. *TCP Establishment Phase.*

This takes advantage of the fact that in all video streaming applications, a TCP session is normally instituted between the client and the content server. If the intruder is able to send TCP RST packets over and over again, this will render a connection between the source and destination impossible to establish. Effectively, the TCP traffic is prevented from flowing between the source and destination.

C. Source-Quench Attacks

In his study, Bellovin (2013), illustrate that intruders can capitalize on the ICMP source quench signals that are ideally employed by routers overwhelmed by data traffic to instruct TCP data sources to decrease their transmission rate. Figure 8 shows a typical source quench message.



<table>
<tr><th>00</th><th>01</th><th>02</th><th>03</th><th>04</th><th>05</th><th>06</th><th>07</th><th>08</th><th>09</th><th>10</th><th>11</th><th>12</th><th>13</th><th>14</th><th>15</th><th>16</th><th>17</th><th>18</th><th>19</th><th>20</th><th>21</th><th>22</th><th>23</th><th>24</th><th>25</th><th>26</th><th>27</th><th>28</th><th>29</th><th>30</th><th>31</th></tr>
<tr><td colspan="8">Type = 4</td><td colspan="8">Code = 0</td><td colspan="16">Header checksum</td></tr>
<tr><td colspan="32">unused</td></tr>
<tr><td colspan="32">IP header and first 8 bytes of original datagram's data</td></tr>
</table>

Figure 8. *Typical Source Quench Message.*

For this attack to succeed, the Type field must be set to a value of 4, the Code field must be set to a value of zero (0) and the IP header and additional data must be employed by the intruder to match the reply with the associated request.

In this scenario, the malicious machine falsifies such signals in an effort to employ them as a vector to launch a denial of service attack on the TCP sources.

Apart from source quench messages, other internet control management protocol (ICMP) messages are able to be maliciously employed to reset an ongoing communication between two or more parties. This is achieved by having the malicious machine transmitting to any of the participating parties an ICMP error signal claiming that there is a hard error.

Hard errors are ICMP error signals of type 3 that indicates *'destination unreachable'*, code 2 that shows *'protocol unreachable'*, code 3 that indicates 'port unreachable', or code 4 that shows *'fragmentation needed'* and the DF bit is set. Since RFC 1122 requires that a host machine should terminate the corresponding TCP connection upon receipt of an ICMP error message, this attack effectively tears down the communication among parties.

D. TCP Session Hijacking

The ultimate goal of this attack is to takeover an already established TCP connection session among communicating parties. *John & Barry* (2013) explain that this is achieved by infusing malicious contents into this TCP session. This is most common in telnet remote communications where the target machine can be made to execute the inserted malicious code. Figure 9 shows a normal three way handshake and data exchange among communicating entities.

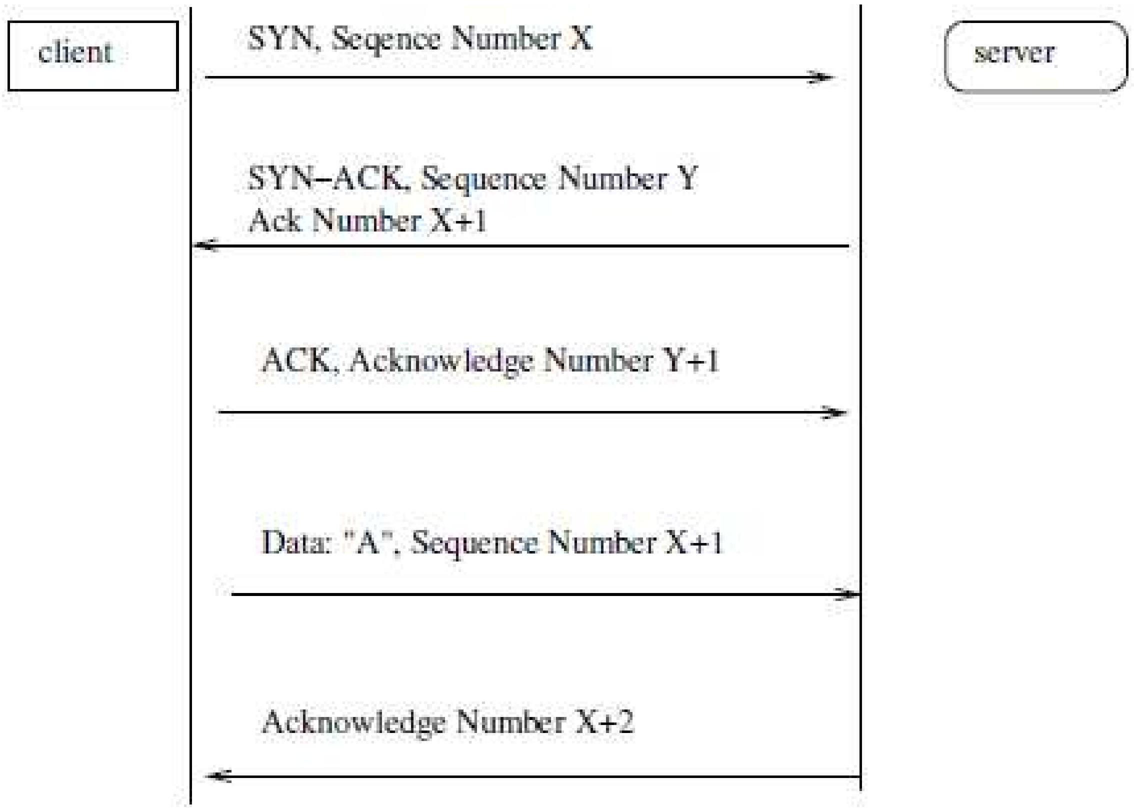

client
SYN, Seqence Number X
server
SYN–ACK, Sequence Number Y
Ack Number X+1
ACK, Acknowledge Number Y+1
Data: "A", Sequence Number X+1
Acknowledge Number X+2

Figure 9. *Ideal Three Way Handshake And Data Transmissions.*

This figure shows that the client initiates a three way handshake by sending a packet whose SYN control bit is set (Mahdavi, 2015). The server responds to this request by transmitting a SYN-ACK packet which is a packet whose SYN and ACK control bits are set. The client then transmits ACK to acknowledge the server's SYN-ACK response.

Afterwards, the two machines engage in a TCP communication until the source sends a packet whose FIN flag is set, to indicate to the server that it wants to terminate the connection. Figure 8 illustrates a different scenario where the TCP connection has been hijacked.

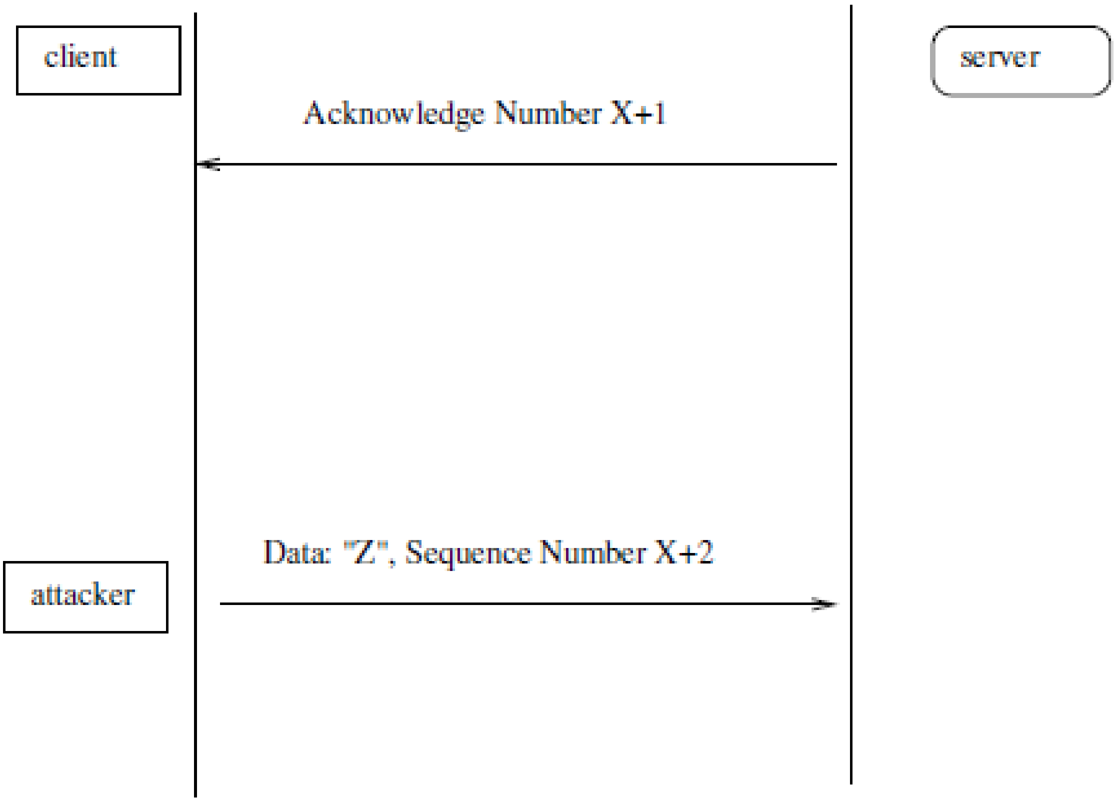


Figure 10. *TCP Session Hijack.*

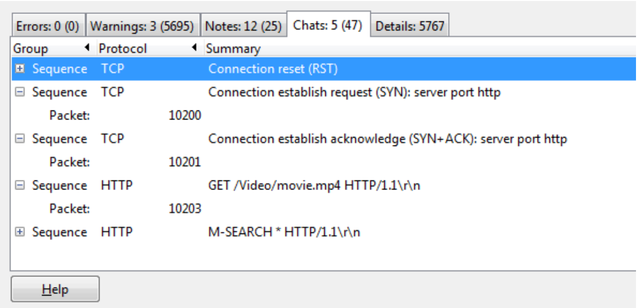
On the other hand, Figure 8 shows data exchange where a malicious machine has taken over the TCP communication among the communicating entities. In this scenario, the malicious machine has sent data segment 'Z' to the server, disguised as the legitimate client.

This attack is made easier by the fact that in most network monitoring tools such as Wireshark, the TCP sequence number displayed is the relative one, meaning that the actual sequence number can be computed by adding the initial sequence number to the present relative sequence number.

E. TCP Sequence Number Prediction

Gont (2015) noted that during the three way handshake, the communicating devices exchange startup packets containing sequence numbers. Considering these two machines as X (client) and Y (server), then X send to Y a SYN packet with a corresponding initial sequence number. The sequence number is a randomly generated number S, also known as the initial sequence number (ISN). The server X transmits to X a SYN-ACK packet as shown in Figure 11.

This packet holds what X anticipates to be the next sequence number from Y, the number *S+1*, in X's Acknowledgement Number field. The SYN-ACK packet must also hold in its sequence Number field a different randomly generated number T. upon receipt of SYN-ACK, X replies with an ACK packet with its Acknowledgement Number field holding its anticipated sequence number that Y will utilize in it next TCP transmission to X, which is *T+1*. This completes the three way handshake for initializing a TCP connection among the communicating entities. As this example illustrates, T and S are incremented in a predictable manner that can be easily determined by a malicious machine.



Group	Protocol	Summary
⊞ Sequence	TCP	Connection reset (RST)
⊟ Sequence	TCP	Connection establish request (SYN): server port http
Packet:	10200	
⊟ Sequence	TCP	Connection establish acknowledge (SYN+ACK): server port http
Packet:	10201	
⊟ Sequence	HTTP	GET /Video/movie.mp4 HTTP/1.1\r\n
Packet:	10203	
⊞ Sequence	HTTP	M-SEARCH * HTTP/1.1\r\n

Figure 11. *TCP Sequence Number Prediction.*

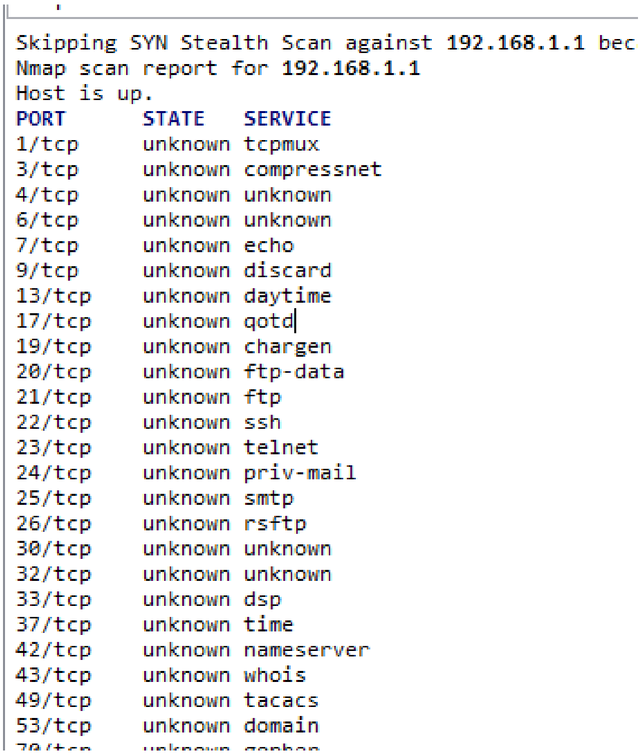
As Figure 9 demonstrates, the TCP packet with sequence number 10200 is the TCP SYN packet that is sent by the client to the server to request for the establishment of a TCP connection. The server responds with SYN-ACK packet of sequence number 10201.

Afterwards, the client acknowledges and goes on to request the required services, which is the downloading of a movie located on the server. This is request whose sequence number is 10203. This illustrates the ease with which the sequence numbers can be predicted to launch other attacks.

F. Port Scanning

This mechanism is used as an attack vector by allowing malicious users to determine open TCP ports as well as services running on these ports. This requires that a TCP connection be established to the victim machine. For example, in UNIX systems, may employ the *connect ()* call to open a connection with the victim's ports (Welian, 2010). This call will succeed if that particular victim is listening on that port. Port and service discovery can also be established using a SYN scanning technique.

In this technique, a packet is sent to the victim in pretence of the need to create a real TCP connection. A SYN-ACK will then be an indication of an active receiver while a RST message will be an implication of a port with no listening services. Figure 10 shows a sample data from a port scan.



```
Skipping SYN Stealth Scan against 192.168.1.1 bec
Nmap scan report for 192.168.1.1
Host is up.
PORT      STATE   SERVICE
1/tcp     unknown tcpmux
3/tcp     unknown compressnet
4/tcp     unknown unknown
6/tcp     unknown unknown
7/tcp     unknown echo
9/tcp     unknown discard
13/tcp    unknown daytime
17/tcp    unknown qotd
19/tcp    unknown chargen
20/tcp    unknown ftp-data
21/tcp    unknown ftp
22/tcp    unknown ssh
23/tcp    unknown telnet
24/tcp    unknown priv-mail
25/tcp    unknown smtp
26/tcp    unknown rsftp
30/tcp    unknown unknown
32/tcp    unknown unknown
33/tcp    unknown dsp
37/tcp    unknown time
42/tcp    unknown nameserver
43/tcp    unknown whois
49/tcp    unknown tacacs
53/tcp    unknown domain
70/tcp    unknown gopher
```

Figure 12. *Data From Port Probing.*

This figure shows that this port scan led to discovery of port number, running protocol, state of the port as well as the services running on these ports. Another attack involving probing is the FIN scanning.

In this technique, FIN packets are sent and RST packets are waited from closed ports. The rationale is that an active listener will discard this packet silently without any need to reply. Hence if a response is received, it would b an indication of a closed port, upon which another set of ports are probed.

G. CHARGEN and ECHO

The UNIX systems implement CHARGEN services in their TCP/IP. This service executes on both UDP and TCP port numbers 19 and it works by having the receiver transmit back a packet with 0 to 512 randomly chosen characters for every incoming UDP packet.

In the case of ECHO that runs on both UDP and TCP port numbers 7, the requirements are that the receiver institutes a reply with whatever it has received for each incoming packet.

Although these two services are crucial for diagnostic purposes, they can easily be compromised and be used as attack vectors for denial of service (Guang, 2015).

To put this into context, we consider a case where the CHARGEN and ECHO services have established a chain between themselves. In this scenario, these two services will generate traffic continuously which in the final analysis will lead to a very high number of data traffic on the network. This is a denial of service (DOS) as illustrated by Figure 13.

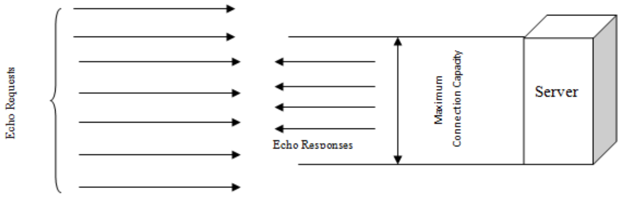


Figure 13. *CHARGEN and ECHO Initiated DOS.*

As an illustration, we consider two machines X and Y and an intruder Z. Using spoofed IP address, Z sends a UDP packet, whose source address field is set with Y's IP address, the destination IP address field is set as X, its source port set as 7, and destination port set as 19. Upon arrival at machine X, it will imagine that Y requires an access to the CHARGEN service.

Therefore, X will transmit back a packet to Y's ECHO port, leading to the creation of a chain and therefore the resulting high volumes of traffic generated on the network. Ultimately, this will lead to the legitimate users experiencing a reduction in the network data rates.

5. Discussions

In this section, the various TCP-based techniques that the researchers believe could help reduce or mitigate the TCP IP header attacks and vulnerabilities are elaborated. These mechanisms include TCP pacing, random early detection and selective acknowledgements.

However, it was noted that there is some kind of trade-off incurred in the utilization of these techniques. Notably present in all these is the notion of acknowledgements, which have been shown to facilitate other attacks such as denial of service attacks exploiting the three way handshake sequence number-acknowledgement exchanges over the transmission channels.

A. TCP Pacing

The rationale of this technique is that if the TCP packet sources can space the data packets to be sent away from each other, then it could be possible to mitigate the burst packet transmissions (Wei et al., 2010). The argument is that most TCP IP attacks in one way or another lead to denial of service attacks which lead to congestions and eventual packet losses.

Figure 14 gives an illustration of this mechanism. This figure shows three data packets in transit in the communication channel. Note that these data packets are separated from each other by a predetermined gap, called *'rest position'* for this study. Therefore, by preventing burst transmissions, the durations of congestions and dropping of packets by receivers will be reduced. Figure 15 compares paced against non-paged TCP connections.

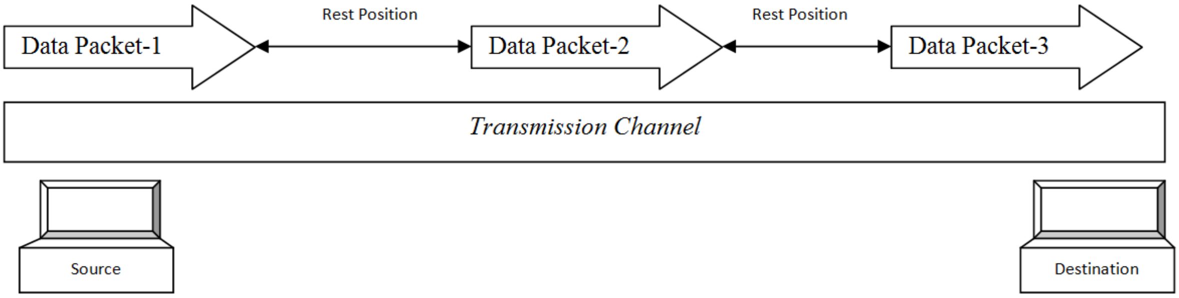


Figure 14. *TCP Pacing.*

This graph of latency against flows measured the latency of 16 parallel SACK flows in a local area network, where each flow sends one megabyte of data.

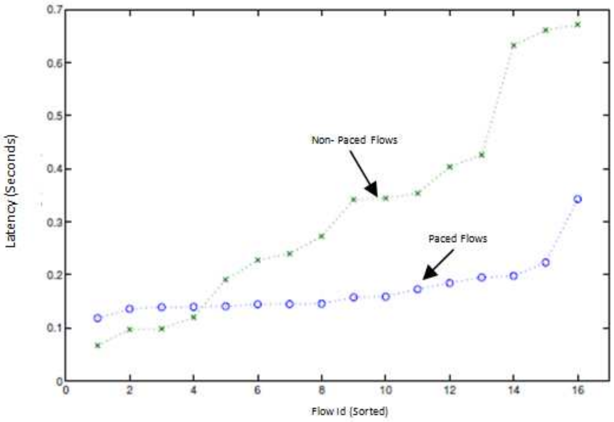


Figure 15. *Paced Versus Non-Paced TCP Connections.*

Figure 15 shows that TCP (Transmission Control Protocol) pacing evens out the transmission of a window of packets over a round-trip time (RTT), so that packets are injected into the network at the desired rate of *congestion_window_size/RTT*. In so doing, it reduces burstiness of TCP traffic caused by ACK compression.

Therefore, paced connections have generally low latency as seen in Figure 15. It works by creating some *'rest positions'* before the TCP sender could push some packets into the transmission channel. This leads to the transmission of discontinuous streams of data as opposed to the continuous bursts of data packets.

B. Random Early Detection

This algorithm manages router buffer queues and discards data packet during congestion based on a certain queue cut off limit (*Zhang et al., 2010)*. The effect of this is that congestion control is initialized just before any actual network congestion event occurs as shown in Figure 16.

Figure 16 demonstrates the fact that the congestion avoidance swings into action just before the slow start algorithm could reach its slow start threshold value. Therefore, the routers and other internetworking devices can react to denial service attacks before the actual denial of service attack is fully exploited.

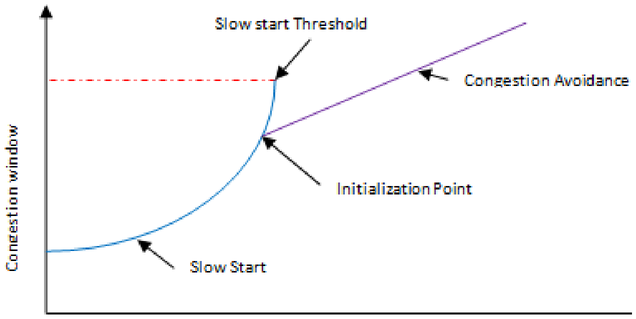


Figure 16. *Random Early Detection.*

In doing this, the receivers will monitor their buffer queues and start dropping packets with a predetermined probability once their threshold values have been exceeded. Effectively, the packet sources will initiate slow start and congestion avoidance algorithms long before the actual congestion occurs. In so doing, it prevents high bandwidth TCP connections from inhibiting transmissions for low bandwidth connections.

Moreover, it ensures TCP fairness in that it mitigates the chance that some unfriendly TCP implementations may gain unreasonable advantage. It does this by eliminating the reliance on TCP source and receiver trust relationships. In fact, this technique will help prevent advanced persistent threats which normally starve other TCP communications in their bid to gain high bandwidth for large files transfers. This is because it ensures that connections utilizing large share of the bandwidth have their packet loss probability higher than that of connections with low bandwidth usage.

C. Selective Acknowledgements

TCP's poor handing of congestion that leads to unnecessary retransmission of packets emanates from the fact that destination machine has no mechanism of instructing the source that it has received some packets which are currently in its buffer queue.

Therefore, when retransmission occurs, even the packets currently on the receiver's buffer are retransmitted. This is imposed by the requirement that the TCP destination send ACK only for the packets it has received correctly. In ideal situations, the source should transmit only the lost segments. This is enabled by selective acknowledgement (SACK), a technique implemented as a TCP option.

This requires that the source and receiver negotiate the use of this TCP option via the TCP header option fields (Stretch, 2010). In such cases then, the destination machine can provide the source with feedback in the form of selective acknowledgements option. The destination machine instructs the source on which blocks of data it has received. This takes the format shown in Figure 17.

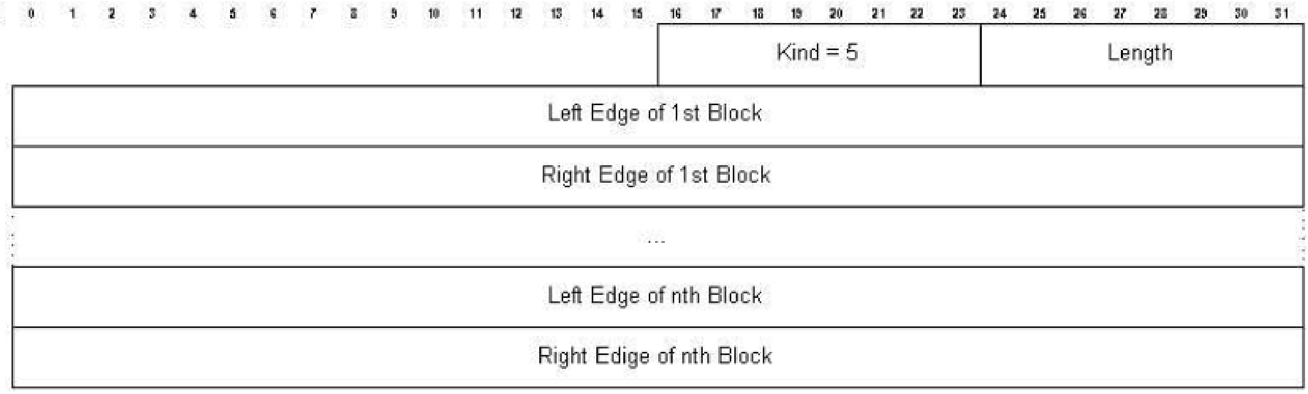


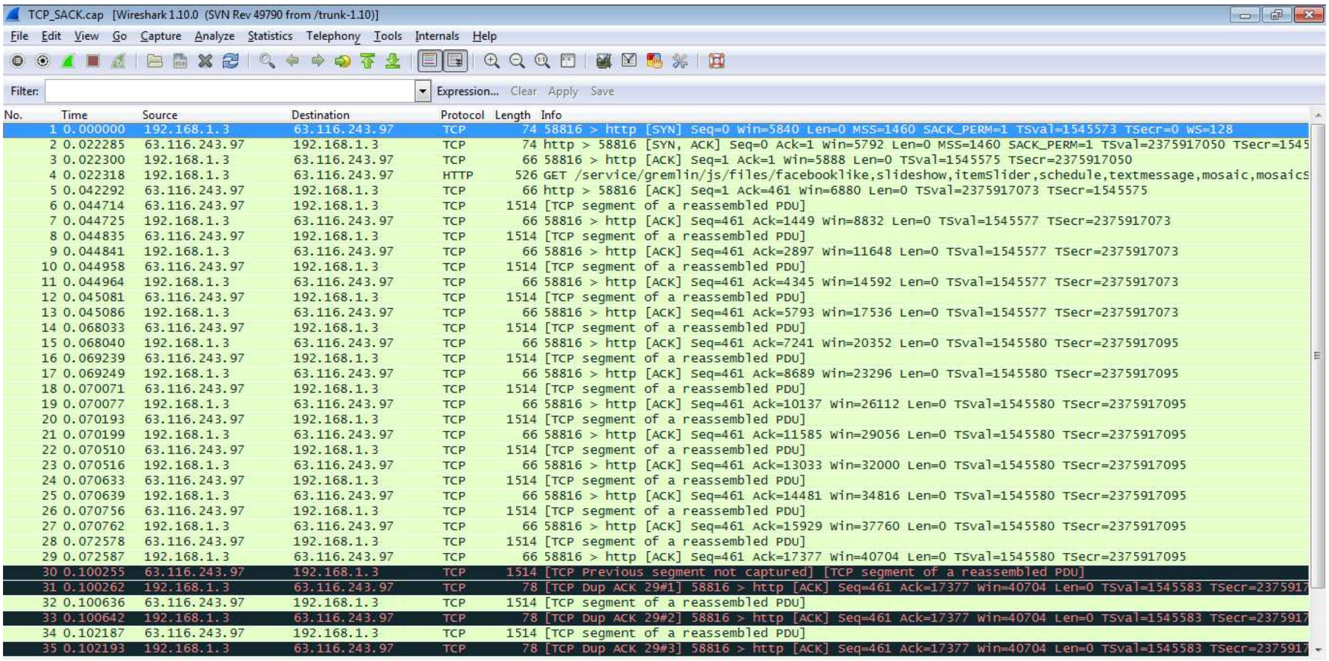
Figure 17. *TCP SACK Option.*

Figure 17 shows that SACK employs a list of blocks that serve to inform the source on the byte stream that has already been received successfully. However, the source does not have to depend on the destination machine to sustain the out of order packets.

If the receiver were to re-order and queue the packets that had been sent between the time when the segment was initially sent and the time when the retransmission window expired, will performance gain be achieved. This is because in this case, the source will limit its retransmissions to only those packets that have been expressly reported lost.

As an illustration, Figure 18 shows a Wireshark captured packet, which contains a demonstration of SACKs in action. The presence of the SACK permitted (SACK_PERM) option in the two SYN packets, packet number 1 and packet number 2 imply that both end hosts support selective acknowledgments.

Toward the end of the capture, it can be seen that packet number 30 was received out of order, and the client has sent a duplicate acknowledgment in packet number 31.

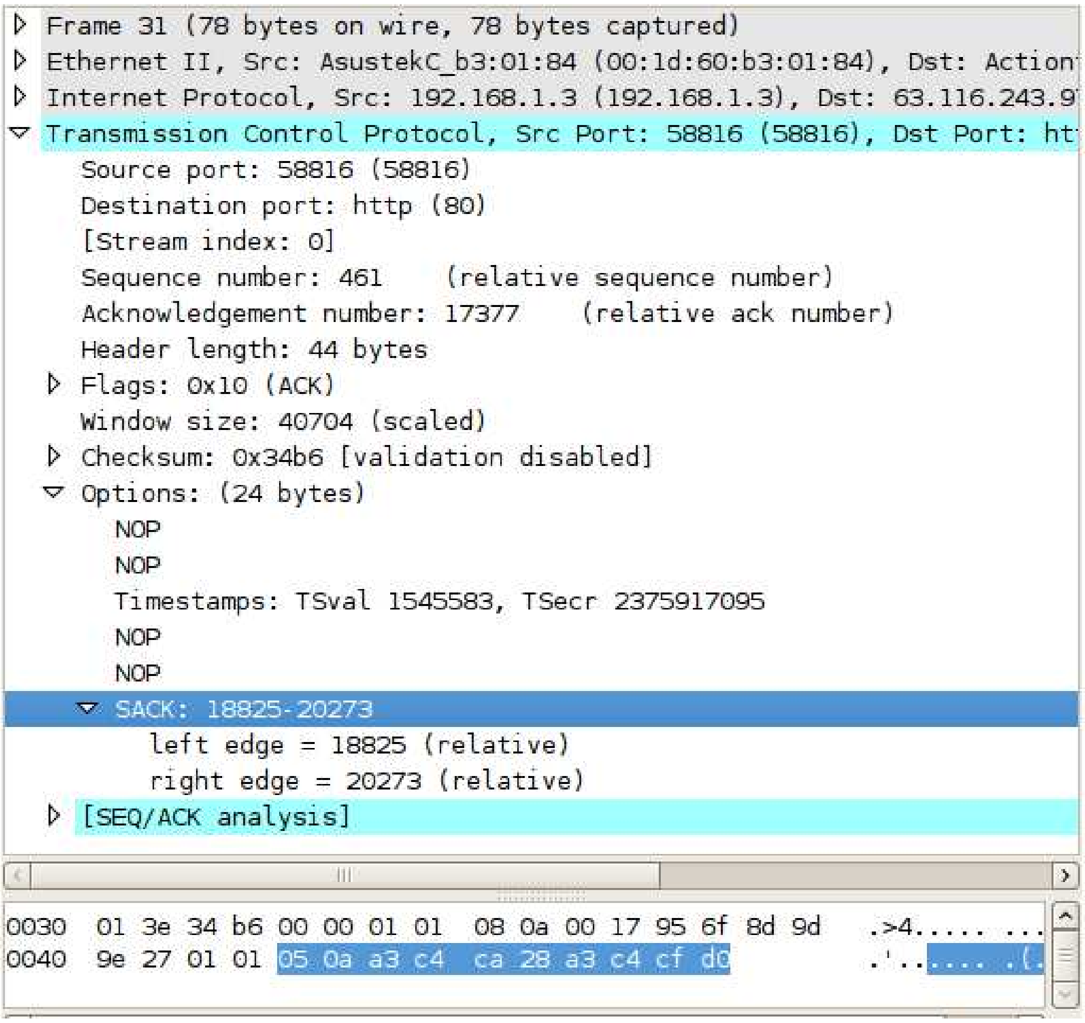

TCP_SACK.cap [Wireshark 1.10.0 (SVN Rev 49790 from /trunk-1.10)]
File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help
Filter:
Expression... Clear Apply Save

```
No.   Time       Source          Destination     Protocol Length Info
    1 0.000000   192.168.1.3     63.116.243.97   TCP        74 58816 > http [SYN] Seq=0 Win=5840 Len=0 MSS=1460 SACK_PERM=1 TSval=1545573 TSecr=0 WS=128
    2 0.022285   63.116.243.97   192.168.1.3     TCP        74 http > 58816 [SYN, ACK] Seq=0 Ack=1 Win=5792 Len=0 MSS=1460 SACK_PERM=1 TSval=2375917050 TSecr=1545
    3 0.022300   192.168.1.3     63.116.243.97   TCP        66 58816 > http [ACK] Seq=1 Ack=1 Win=5888 Len=0 TSval=1545575 TSecr=2375917050
    4 0.022318   192.168.1.3     63.116.243.97   HTTP      526 GET /service/gremlin/js/files/facebooklike,slideshow,itemSlider,schedule,textmessage,mosaic,mosaics
    5 0.042292   63.116.243.97   192.168.1.3     TCP        66 http > 58816 [ACK] Seq=1 Ack=461 Win=6880 Len=0 TSval=2375917073 TSecr=1545575
    6 0.044714   63.116.243.97   192.168.1.3     TCP      1514 [TCP segment of a reassembled PDU]
    7 0.044725   192.168.1.3     63.116.243.97   TCP        66 58816 > http [ACK] Seq=461 Ack=1449 Win=8832 Len=0 TSval=1545577 TSecr=2375917073
    8 0.044835   63.116.243.97   192.168.1.3     TCP      1514 [TCP segment of a reassembled PDU]
    9 0.044841   192.168.1.3     63.116.243.97   TCP        66 58816 > http [ACK] Seq=461 Ack=2897 Win=11648 Len=0 TSval=1545577 TSecr=2375917073
   10 0.044958   63.116.243.97   192.168.1.3     TCP      1514 [TCP segment of a reassembled PDU]
   11 0.044964   192.168.1.3     63.116.243.97   TCP        66 58816 > http [ACK] Seq=461 Ack=4345 Win=14592 Len=0 TSval=1545577 TSecr=2375917073
   12 0.045081   63.116.243.97   192.168.1.3     TCP      1514 [TCP segment of a reassembled PDU]
   13 0.045086   192.168.1.3     63.116.243.97   TCP        66 58816 > http [ACK] Seq=461 Ack=5793 Win=17536 Len=0 TSval=1545577 TSecr=2375917073
   14 0.068033   63.116.243.97   192.168.1.3     TCP      1514 [TCP segment of a reassembled PDU]
   15 0.068040   192.168.1.3     63.116.243.97   TCP        66 58816 > http [ACK] Seq=461 Ack=7241 Win=20352 Len=0 TSval=1545580 TSecr=2375917095
   16 0.069239   63.116.243.97   192.168.1.3     TCP      1514 [TCP segment of a reassembled PDU]
   17 0.069249   192.168.1.3     63.116.243.97   TCP        66 58816 > http [ACK] Seq=461 Ack=8689 Win=23296 Len=0 TSval=1545580 TSecr=2375917095
   18 0.070071   63.116.243.97   192.168.1.3     TCP      1514 [TCP segment of a reassembled PDU]
   19 0.070077   192.168.1.3     63.116.243.97   TCP        66 58816 > http [ACK] Seq=461 Ack=10137 Win=26112 Len=0 TSval=1545580 TSecr=2375917095
   20 0.070193   63.116.243.97   192.168.1.3     TCP      1514 [TCP segment of a reassembled PDU]
   21 0.070199   192.168.1.3     63.116.243.97   TCP        66 58816 > http [ACK] Seq=461 Ack=11585 Win=29056 Len=0 TSval=1545580 TSecr=2375917095
   22 0.070510   63.116.243.97   192.168.1.3     TCP      1514 [TCP segment of a reassembled PDU]
   23 0.070516   192.168.1.3     63.116.243.97   TCP        66 58816 > http [ACK] Seq=461 Ack=13033 Win=32000 Len=0 TSval=1545580 TSecr=2375917095
   24 0.070633   63.116.243.97   192.168.1.3     TCP      1514 [TCP segment of a reassembled PDU]
   25 0.070639   192.168.1.3     63.116.243.97   TCP        66 58816 > http [ACK] Seq=461 Ack=14481 Win=34816 Len=0 TSval=1545580 TSecr=2375917095
   26 0.070756   63.116.243.97   192.168.1.3     TCP      1514 [TCP segment of a reassembled PDU]
   27 0.070762   192.168.1.3     63.116.243.97   TCP        66 58816 > http [ACK] Seq=461 Ack=15929 Win=37760 Len=0 TSval=1545580 TSecr=2375917095
   28 0.072578   63.116.243.97   192.168.1.3     TCP      1514 [TCP segment of a reassembled PDU]
   29 0.072587   192.168.1.3     63.116.243.97   TCP        66 58816 > http [ACK] Seq=461 Ack=17377 Win=40704 Len=0 TSval=1545580 TSecr=2375917095
   30 0.100255   63.116.243.97   192.168.1.3     TCP      1514 [TCP Previous segment not captured] [TCP segment of a reassembled PDU]
   31 0.100262   192.168.1.3     63.116.243.97   TCP        78 [TCP Dup ACK 29#1] 58816 > http [ACK] Seq=461 Ack=17377 Win=40704 Len=0 TSval=1545583 TSecr=2375917
   32 0.100636   63.116.243.97   192.168.1.3     TCP      1514 [TCP segment of a reassembled PDU]
   33 0.100642   192.168.1.3     63.116.243.97   TCP        78 [TCP Dup ACK 29#2] 58816 > http [ACK] Seq=461 Ack=17377 Win=40704 Len=0 TSval=1545583 TSecr=2375917
   34 0.102187   63.116.243.97   192.168.1.3     TCP      1514 [TCP segment of a reassembled PDU]
   35 0.102193   192.168.1.3     63.116.243.97   TCP        78 [TCP Dup ACK 29#3] 58816 > http [ACK] Seq=461 Ack=17377 Win=40704 Len=0 TSval=1545583 TSecr=2375917
```

Figure 18. *SACK PERMITTED.*

This packet includes a SACK option indicating that the segment in packet number 30 was received.

However, the SACK option cannot simply specify which segment(s) were received. For this reason, it specifies the left and right edges of data that has been received beyond the packet's acknowledgment number as shown in Figure 19. A single SACK option can specify multiple noncontiguous blocks of data (such as bytes 200-299 and 400-499).

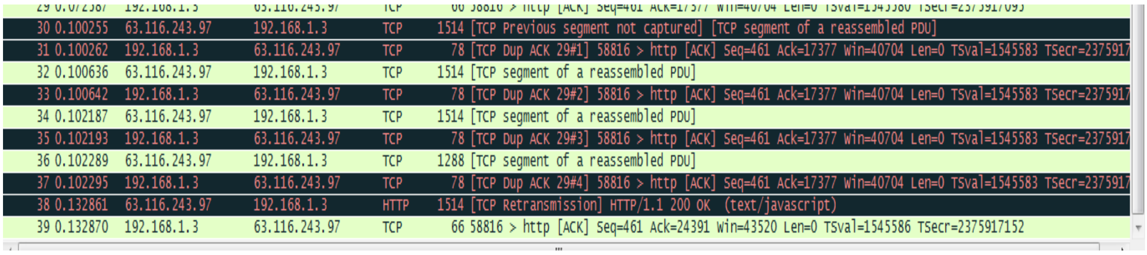


```
▷ Frame 31 (78 bytes on wire, 78 bytes captured)
▷ Ethernet II, Src: AsustekC_b3:01:84 (00:1d:60:b3:01:84), Dst: Action
▷ Internet Protocol, Src: 192.168.1.3 (192.168.1.3), Dst: 63.116.243.9
▽ Transmission Control Protocol, Src Port: 58816 (58816), Dst Port: ht
    Source port: 58816 (58816)
    Destination port: http (80)
    [Stream index: 0]
    Sequence number: 461    (relative sequence number)
    Acknowledgement number: 17377    (relative ack number)
    Header length: 44 bytes
  ▷ Flags: 0x10 (ACK)
    Window size: 40704 (scaled)
  ▷ Checksum: 0x34b6 [validation disabled]
  ▽ Options: (24 bytes)
      NOP
      NOP
      Timestamps: TSval 1545583, TSecr 2375917095
      NOP
      NOP
    ▽ SACK: 18825-20273
        left edge = 18825 (relative)
        right edge = 20273 (relative)
  ▷ [SEQ/ACK analysis]

0030  01 3e 34 b6 00 00 01 01  08 0a 00 17 95 6f 8d 9d   .>4..... ...
0040  9e 27 01 01 05 0a a3 c4  ca 28 a3 c4 cf d0         .'.......(.
```

Figure 19. *Left And Right Edges Of Data.*

Figure 18 illustrates that there is duplicate acknowledgment repeated in packets number 33, 35, and 37. In each, the SACK is expanded to include the noncontiguous segments the server has continued sending. Finally, the server retransmits the missing segment in packet number 38, as shown in Figure 20.



```
29 0.072587   192.168.1.3    63.116.243.97  TCP    66 58816 > http [ACK] Seq=461 Ack=17377 Win=40704 Len=0 TSval=1545580 TSecr=2375917095
30 0.100255   63.116.243.97  192.168.1.3    TCP  1514 [TCP Previous segment not captured] [TCP segment of a reassembled PDU]
31 0.100262   192.168.1.3    63.116.243.97  TCP    78 [TCP Dup ACK 29#1] 58816 > http [ACK] Seq=461 Ack=17377 Win=40704 Len=0 TSval=1545583 TSecr=2375917
32 0.100636   63.116.243.97  192.168.1.3    TCP  1514 [TCP segment of a reassembled PDU]
33 0.100642   192.168.1.3    63.116.243.97  TCP    78 [TCP Dup ACK 29#2] 58816 > http [ACK] Seq=461 Ack=17377 Win=40704 Len=0 TSval=1545583 TSecr=2375917
34 0.102187   63.116.243.97  192.168.1.3    TCP  1514 [TCP segment of a reassembled PDU]
35 0.102193   192.168.1.3    63.116.243.97  TCP    78 [TCP Dup ACK 29#3] 58816 > http [ACK] Seq=461 Ack=17377 Win=40704 Len=0 TSval=1545583 TSecr=2375917
36 0.102289   63.116.243.97  192.168.1.3    TCP  1288 [TCP segment of a reassembled PDU]
37 0.102295   192.168.1.3    63.116.243.97  TCP    78 [TCP Dup ACK 29#4] 58816 > http [ACK] Seq=461 Ack=17377 Win=40704 Len=0 TSval=1545583 TSecr=2375917
38 0.132861   63.116.243.97  192.168.1.3    HTTP 1514 [TCP Retransmission] HTTP/1.1 200 OK  (text/javascript)
39 0.132870   192.168.1.3    63.116.243.97  TCP    66 58816 > http [ACK] Seq=461 Ack=24391 Win=43520 Len=0 TSval=1545586 TSecr=2375917152
```

Figure 20. *Server Packet Re-Transmission.*

Note the *"HTTP 1514 [TCP Retransmission] HTTP/1.1 200 OK (text/javascript)"* for packet 38. This information confirms that the server has indeed retransmitted the TCP packet. After this retransmission, the client updates its acknowledgment number appropriately in packet number 39.

6. Conclusions

This paper studied the various TCP header vulnerabilities such as Port Scanning, TCP Sequence Number Prediction, TCP Session Hijacking, Source-Quench Attacks, TCP RST Attack, TCP SYN Flooding, CHARGEN and ECHO, that could be exploited as attack vectors to launch other attacks. It has also suggested techniques such as selective acknowledgements, random early detection and TCP pacing could help mitigate the various TCP Header attacks. All these mechanisms are geared towards proper handling of congestion so that it could not be used by malicious users to unfairly gain their bandwidth at the expense of other network users. However, there is some form of trade-offs involved, especially for random early detection where TCP connections with high bandwidth usage have their packets dropped with high probability. The consequences are that high bandwidth data transfers such as video traffic, for example, will have more of their packets dropped compared to text documents transfer. This will effectively lead to more buffering, an unwanted scenario in video streaming.

With SACK, the source is able to reduce its retransmissions significantly. However, packets that are out of order and still in the receiver's buffer have to be resent since the destination machine has no mechanism of distinguishing the source the packets that have been lost and the ones that are out of order. TCP pacing will lead to long network delays since the data packets are distantly spaced hence leading to poor network performance. All these shortcomings point to the requirement for novel approaches either from the TCP header architecture perspective or the countermeasure perspective to effectively mitigate TCP header vulnerabilities.

References

[1] N. Shaneel (2014). *Improving Network Performance: An Evaluation of TCP/UDP on Networks*. Department of Computing UNITEC Institute of Technology Auckland, New Zealand.

[2] K. Avi (2016). *Lecture 16: TCP/IP Vulnerabilities and DoS Attacks: IP Spoofing, SYN Flooding, and The Shrew DoS Attack.* Purdue University.

[3] D. Welian (2010). *Attack Lab: Attacks on TCP/IP Protocols*. Syracuse University.

[4] Y. Guang (2015). *Introduction to TCP/IP Network Attacks.* Department of Computer Science Iowa State University.

[5] K. Mirja (2015). *Mitigating TCP ACK loop.*

[6] M. Robbie (2015). *Attacks on TCP/IP Protocols*. Computer Network Security.

[7] M. Bellovin (2013). *Security Problems in the TCP/IP Protocol Suite.*

[8] T. John and E. Barry (2013). *TCP veto: A novel network attack and its application to SCADA protocols.* Innovative Smart Grid Technologies (ISGT), IEEE PES.

[9] A. Mahdavi (2015). *A DDoS Attack Explained: TCP ACK*. Staminus.

[10] F. Gont (2015). *On the Validation of TCP Sequence Numbers.* TCP Maintenance and Minor Extensions.

[11] D. Wei, P. Cao and H. Steven (2010). *TCP Pacing Revisited.*

[12] C. Zhang, J. Yin, C. Zhiping and C. Weifeng (2010). *RRED: robust RED algorithm to counter low-rate denial-of-service attacks.* IEEE Communications Letters. 14 (5): 489–491.

[13] J. Stretch (2010). *TCP Selective Acknowledgments (SACK).*