Network Security

Professor Gaurav S. Kasbekar
Department of Electrical Engineering
Indian Institute of Technology, Bombay
Week - 09
Lecture - 52
Firewalls and Intrusion Detection Systems: Part 2

Hello, in this lecture, we will continue our discussion of firewalls and intrusion detection systems. In the previous lecture, we introduced firewalls and then we started our discussion of some TCP/IP protocols. We need to understand TCP/IP protocols better to appreciate how firewalls operate. We discussed ICMP packets and the traceroute program. Now, we discuss the TCP protocol in some detail.

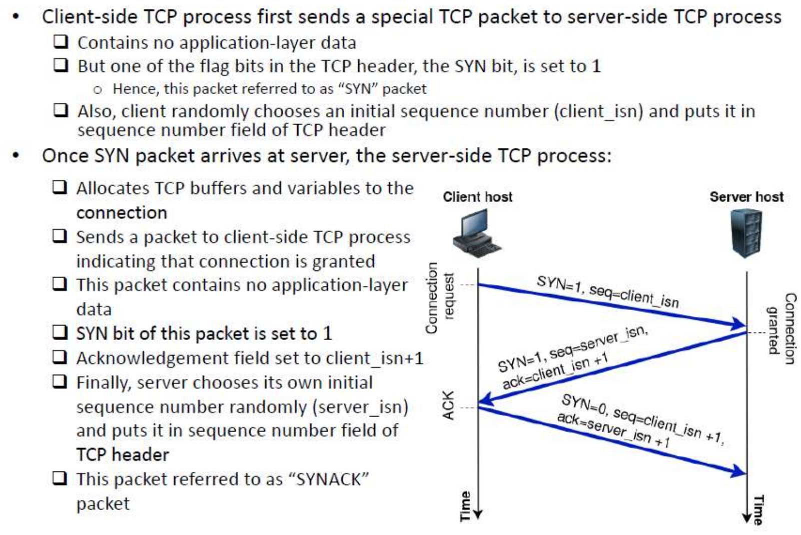
We'll discuss how a TCP session starts and how it ends. So, this knowledge will be used later to understand firewalls better. A TCP session starts using a TCP three-way handshake. That is, three messages are exchanged between the endpoints of the TCP connection. This figure shows how a TCP connection is established.

This is a client host which initiates a TCP connection with another host called the server host. So, the client host first sends a message to the server host in which a flag called SYN, which stands for synchronization, is 1. This packet has a sequence number, which is the initial sequence number of the client host that is denoted here by client_isn, client initial sequence number. So, this message has the client's initial sequence number, and when this message reaches the server host, it allocates resources such as buffers and initializes variables for this TCP connection. And then it sends a response packet to this client host.

The SYN flag is again 1. And this packet contains the initial sequence number of the server. That is server_isn. And the acknowledgement field in this packet acknowledges this packet. So, the acknowledgement number is client_isn +1.

Recall that the acknowledgement number is the expected next sequence number. Since the latest sequence number received by the server is client_isn, the expected next sequence number is client_isn+1. Hence, the acknowledgement field in this message is client_isn+1. So, when this message reaches the client host, it allocates buffers to the connection and also initializes variables. And then the client host sends another packet.

In this packet the SYN flag is 0. And the sequence number is client_isn+1 because this packet is sent after this packet. Hence, the sequence number is client_isn+1. And then this packet also acknowledges the previous packet, that is this packet. Hence, the acknowledgement field is server_isn+1.



- Client-side TCP process first sends a special TCP packet to server-side TCP process
  - ❑ Contains no application-layer data
  - ❑ But one of the flag bits in the TCP header, the SYN bit, is set to 1
    - o Hence, this packet referred to as "SYN" packet
  - ❑ Also, client randomly chooses an initial sequence number (client_isn) and puts it in sequence number field of TCP header
- Once SYN packet arrives at server, the server-side TCP process:
  - ❑ Allocates TCP buffers and variables to the connection
  - ❑ Sends a packet to client-side TCP process indicating that connection is granted
  - ❑ This packet contains no application-layer data
  - ❑ SYN bit of this packet is set to 1
  - ❑ Acknowledgement field set to client_isn+1
  - ❑ Finally, server chooses its own initial sequence number randomly (server_isn) and puts it in sequence number field of TCP header
  - ❑ This packet referred to as "SYNACK" packet

That is because the sequence number in the previous packet, which is this 1, was the server_isn. At this point, the connection is established, and data packets can subsequently be exchanged between the client and the server. Let's discuss this three-way handshake in some more detail. The client-side TCP process first sends a special TCP packet to the server-side TCP process. That is, this packet.

This packet has no application-layer data. But one of the flag bits in the TCP header, namely the SYN bit, which stands for synchronization, is set to 1. Hence, this packet is referred to as the synchronization packet or "SYN" packet. Also, the client randomly chooses an initial sequence number, which is denoted by client_isn. So, that is shown here.

This is the initial sequence number of the client. And the client puts this initial sequence number in the sequence number field of the TCP header of this message. Once the SYN packet arrives at the server, the server-side TCP process allocates TCP buffers and variables to the connection. And then the server-side TCP process sends a packet to the client-side TCP process indicating that the connection is granted. So, at this point, the connection is granted by the server host to the client host.

This packet contains no application layer data. So, this packet has the SYN bit in it set to 1. So, that is shown here. The SYN bit in this message is 1. And the acknowledgement field is set to client_isn+1.

That is because this message acknowledges this message. So, the acknowledgement field is set to client_isn+1. Finally, the server chooses its own initial sequence number randomly, and that is denoted by server_isn. So, we can see that the sequence number is server_isn and the server puts this initial sequence number, server_isn, in the sequence number field of the TCP header. So, we can see that the sequence number in the TCP header of this message is server_isn.

So, this packet is referred to as the “SYNACK” packet. So, that’s because the SYN flag is 1 and acknowledgement field contains a legitimate information. That is the value client_isn+1. So, this first message is known as the SYN packet. This second message is known as the SYNACK packet, and the third message is acknowledgement packet, or ACK packet, as we’ll see.

So, when the client receives the SYNACK packet that happens at this point. Then the client allocates buffers and variables to the connection. So, at this instant in time the client allocates buffers and variables to the TCP connection. Then the client sends the server another packet, which is shown here by the third packet. And this last packet acknowledges the server’s connection granted message.

That is done by the client by putting the value server_isn+1 in the acknowledgement field of the TCP header. So, we can see that the sequence number of this second packet was server_isn, and now that packet has to be acknowledged. So, the acknowledgement packet of the third message is set to server_isn+1. That is the expected next sequence number after this sequence number. Hence, the acknowledgement field is set to server_isn+1.

The SYN bit is set to 0 in this third packet. The reason the SYN bit is set to 1 in the first two packets but it is set to 0 in the third packet is the following. First two packets are used to inform the initial sequence number of the client and the server to the other party. That is, the client informs its initial sequence number to the server, and the server informs its initial sequence number to the client. So, this process is known as synchronization.

So, this packet synchronizes the client initial sequence number between the client and the server, and this packet synchronizes the server_isn from this server to the client. So, hence, these two packets are synchronization packets. They are used to inform the initial sequence numbers. Whereas at this point the initial sequence numbers are already known. Hence, this packet is not a synchronization packet.

Hence, the SYN flag is set to 0. So, this third stage of the three-way handshake may carry client-to-server data in the packet payload. We can piggyback some application data on this third packet, which is sent from the client to the server host. So, once these three steps have been completed, the client and the server host can send packets containing data to each other. This three-way handshake is used to set up a TCP connection.

Once it has been set up, the two parties can send application data to each other. In each of these future packets, the SYN bit will be set to 0. So, one observation we make is that the SYN flag is only 1 in the first two packets of a TCP connection. So, in all other packets the SYN flag is 0. Later on, we'll discuss how firewalls can make use of this information to defend against certain attacks.

Now, let's analyze this three-way handshake. So, what is the reason for using a three-way handshake instead of directly starting the exchange of data packets? So, each side randomly selects an initial sequence number (ISN), and needs to notify the other side of its selected ISN. So, this is a process of synchronization. That is, each party needs to inform its initial sequence number to the other party.

So, in the first packet, which is this one, the client notifies the server of its selected ISN, which is client_isn. And in the second packet, the server acknowledges the receipt of client_isn and notifies the client of its selected ISN, namely the server_isn. We can see that in this first packet, the client informs the server of its initial sequence number, namely client_isn. And in the second packet, the server informs the client of its initial sequence number, which is server_isn. In the third packet, the client acknowledges the receipt of server_isn.

So, that is done by setting the ACK field to server_isn+1. This is the reason that these three messages are required. So, this message is used to inform the client_isn to the server. This message acknowledges the client_isn, and then it informs the server_isn to the client, and then this message acknowledges the packet containing the server_isn So, that's the reason that three packets are required in the TCP handshake.

So, now these initial sequence numbers have to be informed to the other party because they are random values. But what is the reason for using random initial sequence numbers instead of starting with a sequence number of 0 each time? The reason is the following.

Suppose initially there is a TCP connection between a client and the server. And then that connection ends, and some other connection is then established between the same client and the same server.

So, there may be some surviving packets from the old connection which got delayed in the network but which were not dropped. They reached the destination late. So, a random ISN for each connection prevents such surviving packets from the old connection from being confused with those from the new connection. So, that's because each TCP connection uses a random initial sequence number. So, the sequence number values which are used in the old connection are disjoint and different from the sequence numbers which are used in the new connection.

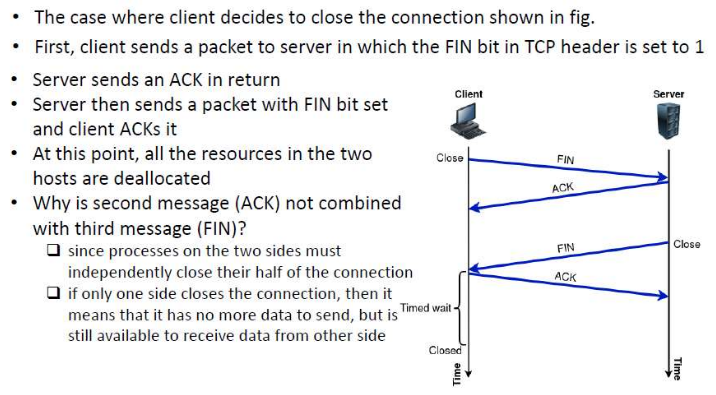
So, hence, the sequence number values of the old connection are not confused with the sequence number values of the new connection. So, whereas if every connection started with the same sequence number of 0, then it was possible that if there were some surviving packets from the old connection, then they would be confused with the packets of the new connection. So, to prevent this, a random initial sequence number is used for each TCP connection. So, we discussed the establishment of a TCP connection. Now we discuss, after the data has been exchanged by the client and the server, how the TCP connection is terminated.

So, either the client TCP or the server TCP process can end the TCP connection. So, when a connection ends, the resources—that is, buffers and variables in the hosts—are deallocated. So, they can be used for other connections. So, either the client or the server can decide to close the connection first. So, the case where the client decides to close the connection is shown in this figure.

The client sends a packet—a TCP packet—to the server in which the FIN flag is set to 1. FIN stands for finish. So, the FIN flag is set to 1, which indicates that the client wants to close the connection. Then, the server sends an acknowledgment for this FIN packet. Then the server sends a packet with the FIN flag 1 to the client, and then the client acknowledges the FIN packet.

The client ends its side of the connection by sending a FIN packet, and the server closes its side by sending a FIN packet. So, first, the client sends a packet to the server in which the FIN bit in the TCP header is set to 1. That packet is this one. Then, the server sends an acknowledgment in return. So, that packet is shown here.

Then the server sends a packet with the fifth bit set to 1, and the client acknowledges the packet. So, that is shown by these packets. So, this is the packet from the server with the FIN flag 1, and this is an acknowledgement for the FIN. So, at this point, all the resources in the two hosts are deallocated. So, once the FIN and ACK have taken place, then the resources in the two hosts are deallocated.



- The case where client decides to close the connection shown in fig.
- First, client sends a packet to server in which the FIN bit in TCP header is set to 1
- Server sends an ACK in return
- Server then sends a packet with FIN bit set and client ACKs it
- At this point, all the resources in the two hosts are deallocated
- Why is second message (ACK) not combined with third message (FIN)?
  - ❑ since processes on the two sides must independently close their half of the connection
  - ❑ if only one side closes the connection, then it means that it has no more data to send, but is still available to receive data from other side

Now, a question is: why is the second message, namely the acknowledgment, not combined with the third message, FIN? So, if we combine this packet and this packet, then the overhead will reduce slightly. The server will just need to send one packet instead of sending two packets. So, why is that not done? So, the reason is that the processes on the two sides, namely the client and the server, must independently close their half of the connection.

For example, it can happen that the client has finished sending its data to the server, but the server still has some data that it wants to send to the client. So, in that case, the client will send a FIN packet to the server, and then the server will ACK it, but the server will not immediately send its own FIN packet. It will instead send the remaining application data to the client, and only after the server has finished sending all its application data will it send the FIN packet. So, processes on the two sides must independently close their half of the connection. If only one side closes the connection, then it means that it has no more data to send, but it is still available to receive data from the other side.

So, in our example, when the client finishes sending its data, it sends a FIN packet to the server, but the client is still available to receive data from the server in case the server has some additional data to send. So, once the server finishes sending its data, it will send its FIN packet. So, for this reason, the FIN and ACK are sent separately for the client and the server. Now, we have discussed TCP connection establishment and TCP connection closure. Now, there is an attack on TCP connection establishment known as the SYN flood attack, in which an attacker sends a large number of SYN packets to the victim.

Recall that a server allocates and initializes connection variables and buffers in response to a received SYN packet. We discussed that in this figure. So, when the server receives this SYN packet from the client—this is the SYN packet—the server allocates buffers and variables to the connection. So, that happens at this instant in time. So, a server allocates and initializes connection variables and buffers in response to a received SYN.

The server then sends a SYNACK in response and awaits an acknowledgment packet from the client. If the client does not send an acknowledgment to complete the third step in this three-way handshake, this will happen when the client is a malicious entity. In that case, eventually, after a minute or more, the server will terminate the half-open connection and reclaim the allocated resources. So, one possibility is that the client is some malicious entity which just wants to occupy the resources of the server. In that case, the client does not send an acknowledgment to complete the third step in the TCP three-way handshake.

Another possibility is that after sending the SYN packet, the client just crashed. In that case, that is another reason why the client may not have sent an acknowledgement to complete the third step. So, if this happens, then after a minute or more, the server will terminate the half-open connection and reclaim the allocated resources. This TCP connection management protocol is vulnerable to a Denial-of-Service (DoS) attack known as the SYN flood attack. In this attack, the attacker sends a large number of SYN packets to the victim and does not send the following ACK packets.

So, this results in a large number of half-open connections at the victim, and this blocks up the resources of the victim. So, in the attack, attacker or attackers send a large number of TCP SYN packets without completing the third handshake step. With this deluge of SYN packets, the server's connection resources become exhausted as they are allocated for half-open connections, but they are never used. So, legitimate clients are thus denied service because a lot of resources are allocated in response to this deluge of SYN packets from the attacker. So, there are no resources left to serve legitimate users.

So, this attack is quite easy for the attacker because the attacker just has to send series of SYN packets, and then it does not respond to the SYN-ACK packets sent by the server with an ACK packet. But this SYN flood attack occupies resources at the victim. Now there is a defense against SYN flood attacks using a mechanism called cookies. We discussed in the context of IPsec, we discussed the use of cookies. In particular, in the case of Internet Key Exchange Protocol, we discussed that cookies are used to prevent an attacker from forcing the victim to perform expensive computational operations.

So, this defense against the SYN flood attacks using cookies has some similarities with that mechanism used in the Internet Key Exchange Protocol. So, a defense against SYN flood attacks known as SYN cookies is now deployed in most major operating systems. It works as follows. When the server sends a SYN packet, it does not know if the packet is coming from a legitimate user or is part of a SYN flood attack. So, if it's part of a SYN flood attack, then the server should not allocate resources in response to the SYN packet.

But the server does not know if the packet is coming from a legitimate user or is part of a SYN flood attack. So, the server does not create a half-open TCP connection for this SYN. Instead, the server creates a cookie, which is created as follows. The server creates an initial TCP sequence number that is computed by applying a cryptographic hash function to a quantity derived from the following fields. One is the source IP address, another is the destination IP address, and the port numbers of the SYN packet, as well as the secret number that is only known to the server.

So, when the server receives a SYN packet, it does not create a half-open TCP connection for this SYN. Instead, it creates an initial TCP sequence number that is computed by applying a cryptographic hash function to a quantity derived from the source and destination IP addresses and port numbers of the SYN packet as well as secret number known only to the server. So, this carefully designed initial sequence number is called a cookie. Then the server sends the client a SYNACK packet with this special initial sequence number, namely the cookie. So, an important observation is that the server does not remember the cookie or any other state information corresponding to the SYN.

So, in particular, when the server receives the SYN packet from the client, it does not allocate any buffers or variables to the connection. It just calculates this cookie and sends a SYNACK packet, but it does not remember any state information for this half-open connection. So, hence, the resources at the server are not occupied. So, no buffer space or variables are allocated to the connection. So, a legitimate client will then return an acknowledgment packet.

When server receives this acknowledgement packet, it must verify that the acknowledgement corresponds to some earlier SYN which was sent earlier. But recall that the server maintains no memory about SYN packets. So, since the server maintains no memory about SYN packets, this is done using the cookie. So, the server verifies that the

ACK corresponds to some previously sent SYN using cookies. Recall that for a legitimate acknowledgment, the value in the acknowledgment field equals the initial sequence number in the SYNACK packet, which is the cookie value in this case, plus 1.

So, the server has not stored the cookie value, but the server can recalculate the cookie value. So, the server can run the same cryptographic hash function using the source and destination IP addresses and port numbers in the ACK packet, which are the same as in the original SYN, and the same secret number. So, the server recalculates the cookie value in this way. So, if the result of this function plus 1 is the same as the acknowledgment number in the client's ACK packet, then the server concludes that the ACK corresponds to an earlier SYN packet and, hence, the ACK is valid. So, this informs the server that the cookie value it had sent earlier to the client has reached the client, and the client has sent an acknowledgment in response to the received cookie value.

Hence, the client is a legitimate client. It wants to actually set up a TCP connection and not just attack the victim by sending a lot of bogus SYN packets. The server then creates a fully open connection and allocates resources. So, resources are allocated only after the entire TCP connection is established, that is, after the completion of the three-way handshake. So, resources are not allocated after the SYN packet is received by the server.

On the other hand, if the client does not return an acknowledgment packet, then the original SYN packet has done no harm to the server because the server hasn't yet allocated any resources in response to the original bogus SYN packet. So, this original SYN packet caused the server to calculate a cookie and send the cookie to the client. But apart from this, there is no blockage of resources at the server because the server does not allocate resources to the connection in response to the SYN packet. So, this is how the use of a cookie helps in defending against a SYN flood attack. Later on, we'll study an alternative technique to defend against a SYN flood attack.

This alternative technique is based on an intrusion detection system. This concludes our discussion of TCP connection establishment and connection closure. Now, we will discuss the Domain Name System, which is another important protocol of the TCP/IP protocol stack. Internet hosts are addressed in multiple ways. One kind of address for an internet host is a hostname, such as www.google.com or timesofindia.indiatimes.com.

So, these are names which are easy to remember by humans. Internet hosts are also addressed by IP addresses such as 121.7.106.83, which are useful for machines, and they are hierarchically assigned for facilitating routing. For example, a network of an organization may have IP addresses starting with 121.7, and then a particular part of the network may have IP addresses starting with 121.7.106.something, and so on. So, these IP addresses are assigned hierarchically to facilitate routing. DNS, or the Domain Name System, is a directory service that translates hostnames to IP addresses.

So, it can be used to find the IP address corresponding to a particular hostname. So, it has the following components. One is a distributed database implemented in the hierarchy of DNS servers. So, there's a hierarchy of many DNS servers which perform this translation from hostnames to IP addresses. But this database must be contacted, and the IP address corresponding to a hostname must be obtained.

So, the other component of DNS is an application-layer protocol that allows hosts to query the distributed database. So, this application-layer protocol allows hosts to query one of the servers in this DNS database and obtain the IP address corresponding to a certain hostname. So, this is DNS. We provided a very brief overview of DNS. It performs the service of translating between hostnames and IP addresses.

So, we'll use the knowledge of DNS later during our discussion of firewalls and intrusion detection systems. Another fact about the DNS protocol is that it runs over UDP and uses port number 53. So, the DNS server runs on port number 53, and a host that wants to contact the DNS server should send a packet to port number 53. So, this concludes our brief review of DNS. So, in summary, we are discussing firewalls and intrusion detection systems, and we have discussed various protocols that are part of the TCP/IP protocol stack.

For example, we discussed the TCP connection handshake. We discussed the TCP connection establishment process, which consists of a handshake of three messages. And we discussed how TCP connections are closed. And then we discussed the SYN flood attack and the use of cookies to defend against the SYN flood attack. We also discussed the DNS protocol briefly.

So, later on, we'll use this knowledge of TCP/IP protocols to recall our discussion of firewalls and intrusion detection systems. Thank you.