Network Security I

Question 1 ***Introduction to Networking*** ()

(a) **TCP and UDP** The transmission control protocol (TCP) and user datagram protocol (UDP) are two of the primary protocols of the Internet protocol suite.

i. How do TCP and UDP relate to IP (Internet protocol)? Which of these protocols are encapsulated within (or layered atop) one another? Could all three be used simultaneously?

ii. What are the differences between TCP and UDP? Which is considered "best effort"? What does that mean?

Solution:

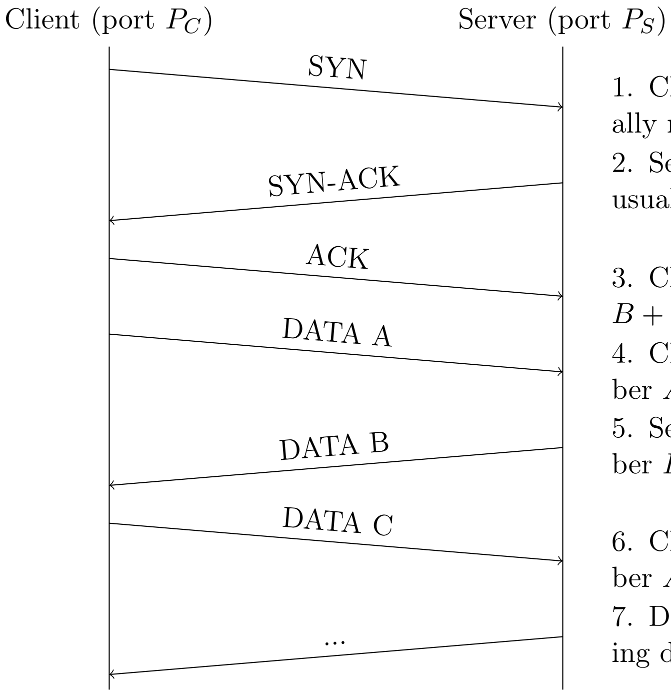
i. TCP and UDP both exist within the transport layer, which is one layer above IP (network layer). Either can be encapsulated in IP, referred to as TCP/IP and UDP/IP. TCP and UDP are alternatives; neither would normally be encapsulated within the other.

ii. TCP provides a *connection-oriented*, *reliable*, *bytestream* service. It includes sophisticated rate-control enabling it to achieve high performance but also respond to changes in network capacity. UDP provides a *datagram-oriented*, *unreliable* service. (Datagrams are essentially individual packets.) The main benefit of UDP is that it is lightweight.

"Best effort" refers to a delivery service that simply makes a single attempt to deliver a packet, but with no guarantees. IP provides such a service, and because UDP simply encapsulates its datagrams directly into IP packets with very little additional delivery properties, it, too, provides "best effort" service.

Question 2 *Attack on TCP* ()

Suppose that a client connects to a server, and then performs the following TCP handshake and initial data transfer:



1. Client sends initial SYN with sequence number A (usually random).
2. Server sends SYN-ACK with sequence number B (also usually random) and ACK $A + 1$.
3. Client sends ACK with sequence number A+1 and ACK $B + 1$.
4. Client sends DATA A of length L_A with sequence number $A + 1$ and ACK $B + 1$.
5. Server sends DATA B of length L_B with sequence number $B + 1$ and ACK $A + 1 + L_A$.
6. Client sends DATA C of length L_C with sequence number $A + 1 + L_A$ and ACK $B + 1 + L_B$.
7. Data exchange continues until both sides are done sending data.

(a) Assume that the next transmission in this connection will be DATA D from the server to the client. What will this packet look like?

Sequence number:	$B + 1 + L_B$	ACK:	$A + 1 + L_A + L_C$
Source port:	P_S	Destination port:	P_C
Length:	L_D	Flags:	ACK

(b) You should be familiar with the concept and capabilities of a *man-in-the-middle* as an attacker who **can observe** and **can modify** traffic. There are two other types of relevant attackers in this scenario:

1. *On-path* attacker: **can observe** traffic but **cannot modify** it.
2. *Off-path* attacker: **cannot observe** traffic and **cannot modify** it.

Carol is an *on-path* attacker. Can Carol do anything malicious to the connection? If so, what can she do?

Solution: Yes, Carol can leverage the information she learns from your traffic to hijack the session.

In part (a), we identified the values of all of the fields of concern expected in the next data transmission in the connection. Say Carol wants to spoof a packet from the server to the client; Carol can create a packet with the source IP as the server's IP, the destination IP as the client's IP, and the payload as whatever she wants. To spoof traffic in the other direction, she swaps the sequence number/ACK, source port/destination port, and source IP/destination IP. The recipient of this data cannot distinguish it from legitimate traffic, so she has effectively hijacked the session from her victim, allowing her to inject arbitrary data.

(c) David is an *off-path* attacker. Can David do anything malicious to the connection? If so, what can he do?

Solution: No, there isn't much he can do.

In part (b), we demonstrated that we are effectively defenseless against an attacker that knows the *sequence numbers* and the *port numbers* of the connection. An off-path attacker, however, does not have the power to observe the traffic and find these parameters.

Even without prior knowledge of these parameters, though, an *off-path* attacker may attempt to guess them. In a typical TCP client-server connection, the client's port is an *ephemeral* port, with a maximum potential range of $[0, 2^{16}-1]$ (this varies, so we make an overestimation). The server's port is usually a *well-known* port for a specific service, such as port 80 for HTTP, which makes it much easier to guess. The sequence number and acknowledgement numbers are in the range $[0, 2^{32}-1]$. However, an attacker can just ignore the acknowledgement number or use a random number. The TCP connection will not be reset with an invalid acknowledgement number as long as it's within the window (out of scope). Thus, an attacker has a rough $\frac{1}{2^{48}}$ chance of successfully brute forcing the correct parameters.

If, for some reason, the initial sequence numbers are not properly randomized, David may be able to make educated guesses on the sequence numbers and significantly decrease the range of possibilities. However, assuming that they are properly randomized, this attack is theoretically possible but largely improbable.

We call this attack *blind hijacking*, as David has no concrete information when attempting to hijack the session.

(d) The client starts getting responses from the server that don't make any sense. Inferring that David is attempting to hijack the connection, the client then immediately sends the server a **RST** packet, which terminates the ongoing connection. David wants to impersonate the client by establishing a new connection. How would he go about doing this?

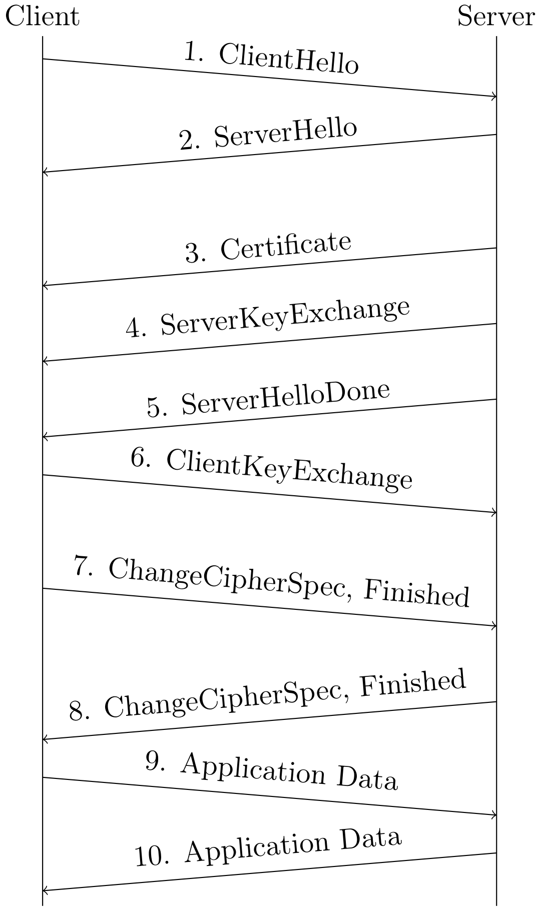
Solution: If David attempts to start a new connection, he can *choose* the source port (the *ephemeral* port) and the source sequence number to be whatever values he wants. The values of these parameters for any subsequent transmissions in the connection will then be predictable. The server's port remains a well-known port; the only remaining unknown is the server's sequence number. Because David is still an off-path attacker, he still has to guess this field, with an overall probability of $\frac{1}{2^{32}}$ of success, which we note is higher than the *blind hijacking* approach.

Note that there's now a time constraint on David's attack: if the client receives a response from the server based on his spoofed *SYN*, it will send a **RST** and terminate the connection, putting David back at step 1.

We call this attack *blind spoofing*, as David has no concrete information when attempting to spoof a new session.

Question 3 *TLS protocol details* ()

Depicted below is a typical instance of a TLS handshake.



1. Client sends 256-bit random number R_b and supported ciphers

2. Server sends 256-bit random number R_s and chosen cipher

3. Server sends certificate

4. DH: Server sends $\{g, p, g^a \bmod p\}_{K^{-1}_{\text{server}}}$

5. Server signals end of handshake

6. DH: Client sends $g^b \bmod p$
   RSA: Client sends $\{PS\}_{K_{\text{server}}}$

Client and server derive cipher keys C_b, C_s and integrity keys I_b, I_s from R_b, R_s, PS

7. Client sends MAC(dialog, I_b)

8. Server sends MAC(dialog, I_s)

9. Client data takes the form $\{M_1, \text{MAC}(M_1, I_b)\}_{C_b}$

10. Server data takes the form $\{M_2, \text{MAC}(M_2, I_s)\}_{C_s}$

Figure 1: TLS 1.2 Key Exchange

(a) What is the purpose of the *client random* and *server random* fields?

Solution: They act as nonces to prevent replay attacks.

NOTE: Previous versions of the discussion mistakenly claimed that these values were used to add entropy to the key - this is incorrect.

(b) ClientHello and ServerHello are not encrypted or authenticated. Explain why a man-in-the-middle cannot exploit this. (Consider both the Diffie-Hellman and RSA case.)

Solution: The use of either public key encryption (RSA handshake) or a Diffie-Hellman exchange prevents an eavesdropper from learning the Premaster Secret.

A MITM attacker who alters any of the values will be exposed, as follows.

For the RSA handshake case, the MITM will be unable to read the Premaster Secret sent by the client because it is encrypted using the server's public key. When the client and server exchange MACs over the the handshake dialog, the MITM will be unable to compute the correct MACs for their altered dialog because they will not know the corresponding integrity keys derived from the Master secret.

For the Diffie-Hellman case, the MITM will be unable to alter the value of $g^a \mod p$, because the client requires that the value have a correct signature using the server's public key. Because the MITM cannot alter the value, they cannot substitute $g^{a'} \mod p$ for which they know a'. Without knowledge of the exponent, the MITM cannot compute $g^{ab} \mod p$ to obtain the Premaster Secret.

(c) Note that in the TLS protocol presented above, there are two cipher keys C_b and C_s. One key is used only by the client, and the other is used only by the server. Likewise, there are two integrity keys I_b and I_s. Alice proposes that both the server and the client should simply share one cipher key C and one integrity key I. Why might this be a bad idea?

Solution: Vulnerable to **reflection** attacks: a man-in-the-middle can send a client's application data back to them. It will still verify the appropriate checks, but the user will think that this is the response from the server. Likewise, an attacker can reflect a server's response back to the client. Note that due to the existence of sequence numbers in the TLS specification, the actual attack would be a little more complicated.

(d) The protocol given above is a simplified form of what actually happens. After step 8 (ChangeCipherSpec), the protocol as described above is still vulnerable. What is the vulnerability and how could you fix this?

Solution: An attacker can perform a **replay** attack, where they send the same application data twice. The solution is to add sequence numbers (which is what the actual TLS specification does, with some extra details involved).