Raluca Ada Popa
Spring 2018

CS 161
Computer Security

Discussion 6

Week of February 26, 2018

Question 1 ***TLS threats*** **(10 min)**

An attacker is trying to attack the company Boogle and its users. Assume that users always visit Boogle's website with an HTTPS connection, using ephemeral Diffie-Hellman. You should also assume that Boogle does not use certificate pinning. The attacker may have one of three possible goals:

1. Impersonate the Boogle web server to a user
2. Discover some of the plaintext of data sent during a past connection between a user and Boogle's website
3. Replay data that a user previously sent to the Boogle server over a prior HTTPS connection

For each of the following scenarios, describe if and how the attacker can achieve each goal.

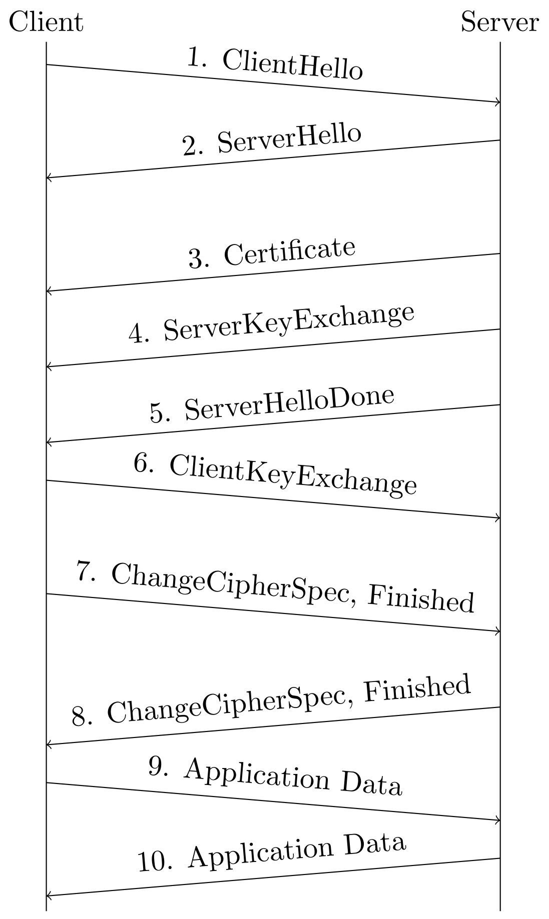
(a) The attacker obtains a copy of Boogle's certificate.

(b) The attacker obtains the private key of a certificate authority trusted by users of Boogle.

(c) The attacker obtains the private key corresponding to an old certificate used by Boogle's server during a past connection between a victim and Boogle's server. Assume that this old certificate has been revoked and is no longer valid. Note that the attacker does not have the private key corresponding to current certificate.

Question 2 *TLS protocol details* (20 min)

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 $\{g, p, g^a \bmod p\}_{K_{\text{server}}^{-1}}$

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?

(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.)

(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?

(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?

Question 3 *Lists and Trees of Hashes* (20 min)

BitTorrent splits large files into small file chunks which are then transmitted between peers in such a way that each peer eventually ends up with the whole file. Commonly, chunks are of size 2^8 KiB = 256 KiB.

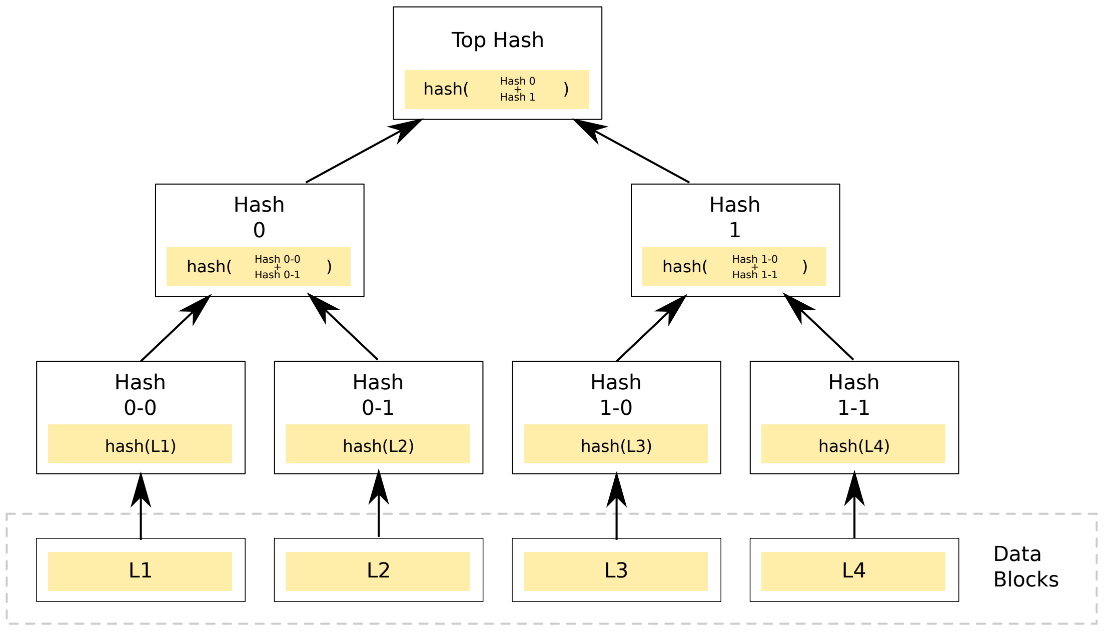
Because you cannot trust peers, you have to verify each chunk as you download them from a peer before you start providing them to other peers. Furthermore, you want to be able to do this as soon as possible and not wait for the whole file to be downloaded. You also want to be able to know which part of the file got potentially corrupted so that you do not have to re-download the whole file.

To achieve the above properties, BitTorrent uses a Torrent file. The file contains information describing the file (or files) to be transmitted, and their chunks. You must obtain this file from a trusted source.

(a) Initially, a Torrent file contained a list of SHA-1 hashes for each chunk. How large is such a list for a 10 GiB large file, if one SHA-1 hash takes 160 bits? (Note: 10 GiB = $10 * 2^{10} * (4 * 256)$ KiB)

(b) One way to make Torrent files smaller is to instead store only a hash of the hash list (top hash, or root hash) in the file and retrieve the hash list itself from a peer. Why would we want to make a Torrent file smaller? What is a downside of this approach?

(c) One approach to address the issue of the size of the hash list is to split it into chunks. However, you would then need a hash list of those chunks. A better approach is to generalize this idea and use a data structure called a hash tree or Merkle tree:



Now you do not need the whole hash list in advance to verify one chunk. Instead, you can ask your peer to provide you with some hashes along with the chunk just received.

Suppose you just received chunk L2 from a peer. Which and how many hashes do you need to verify if you correctly received chunk L2? How would you generalize which and how many hashes you need for each chunk? (Hint: This might be useful to implement efficient updates for part 3 of Project 2.)