Paxson
Spring 2017

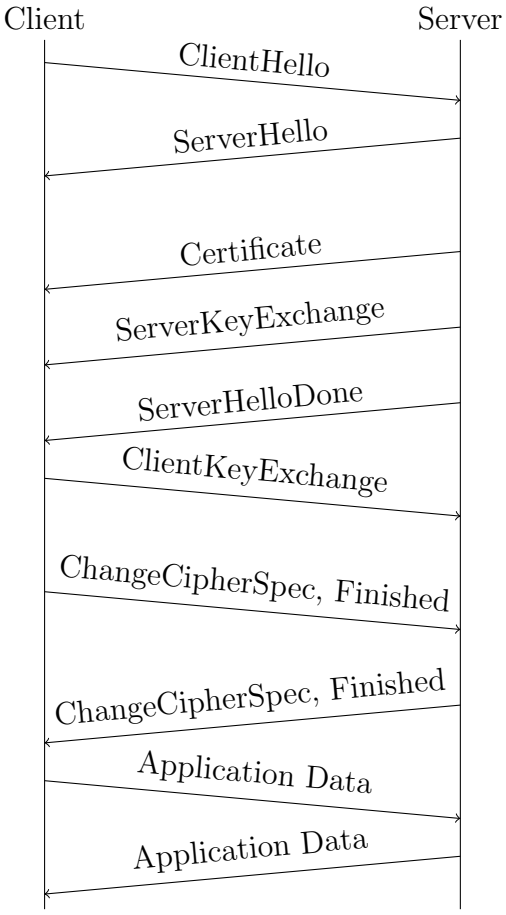
CS 161
Computer Security

Discussion 10

Week of April 10, 2017

Question 1 ***TLS protocol details*** **(25 min)**

Depicted below is a typical instance of a TLS handshake. Use the image to help answer the questions.



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 $\text{MAC}(dialog, I_b)$
8. Server sends $\text{MAC}(dialog, I_s)$
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:** Because the master secret depends on these, they act as nonces that prevent replay attacks.

(b) ClientHello and ServerHello are not encrypted or authenticated. What would hap-

pen if somebody eavesdropped on them? What about an active man-in-the-middle?

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) Unlike the ServerKeyExchange, the ClientKeyExchange is sent without a signature. Why is this the case, and what are the implications?

Solution: In this handshake, the client does not provide a verifiable public key (since it, unlike the server, does not hold a certificate), and so cannot sign messages. This means the client is un-authenticated: the server does not know who it is talking to. This generally works fine for HTTP, since it is the server for which authentication is the #1 concern.

That said, TLS does have support for client-side certificates as well, which can be used for authenticating clients to servers. The potential use of such certificates is part of the initial negotiation in the Client/Server Hello exchange.

(d) TLS provides end-to-end authentication, integrity, and confidentiality guarantees. Is that enough to make online commerce safe and secure? Why or why not?

Solution: TLS provides secure communication between a client and server, but was not specifically designed for online transactions. For instance, the browser checks the name in the certificate against the site's domain name, but this gives no assurance that the site is a bona fide merchant. Similarly, the online merchant has no way to check that the person making the purchase is authorized to use the credit card. Customers can repudiate purchases, claiming their credit card number was stolen.

Question 2 ***TLS threats*** **(15 min)**

An attacker is trying to attack the company Wahoo and its users. Assume that users always visit Wahoo's website with an HTTPS connection, using RSA and AES encryption (no Diffie-Hellman). You should also assume that Wahoo does not use certificate pinning. The attacker may have one of three possible goals:

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

> **Solution:** None of the above. The certificate is public. Anyone can obtain a copy simply by connecting to Wahoo's webserver. So learning the certificate doesn't help the attacker.

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

> **Solution:** The attacker can impersonate the Wahoo web server to a user. The attacker can't decrypt past data, because the attacker doesn't learn Wahoo's private key—only the CA's private key. All that the CA's private key can be used for is to create bogus certificates, which can be used to fool the client into thinking it is talking to Wahoo—but doesn't allow learning past data. Replays aren't possible, due to the nonces in the TLS handshake.

(c) The attacker obtains the private key that was used by Wahoo's server during a past connection between a victim and Wahoo's server, but not the current private key. Also, assume that the certificate corresponding to the old private key has been revoked and is no longer valid.

> **Solution:** The attacker can discover all of the plaintext of data sent during a *past* connection (one where the old private key was used) between a user and Wahoo's website. Since the server is using RSA, an attacker who learns the RSA private key can decrypt past sessions (the attacker can decrypt to learn the premaster secret, derive the symmetric keys, and decrypt all of the data).
>
> This can't be used to impersonate a Wahoo server, however, because the attacker

doesn't have a fresh valid certificate corresponding to the stolen private key, and can't use the previous certificate for that key because it's been revoked.

Question 3 ***Denial-of-service on the web*** **(10 min)**

Your friend has just launched `SiteTester.com`, a cool web service that helps website developers see how their web site will look when rendered with Chrome vs. how it will look in Internet Explorer.

The service is pretty simple. If you visit a URL like `http://sitetester.com/?u=http://berkeley.edu/`, the SiteTester server launches a process running the Chrome browser, loads `http://berkeley.edu/` in Chrome, and takes a screenshot of the Chrome window after the site finishes loading. In parallel, SiteTester starts up Internet Explorer, loads `http://berkeley.edu/` in Internet Explorer, and takes a screenshot of Internet Explorer after the page loads. After both screenshots are available, the SiteTester server serves you a dynamically-generated HTML document that shows both screenshots side-by-side. (Note that, while the SiteTester is handling this HTTP request from the user, it will to issue two separate HTTP requests to `berkeley.edu`, one for each browser.) The SiteTester service can be used on any web page you specify; everything after the `?u=` is treated as a URL and loaded into both browsers. This makes SiteTester very useful to web developers for testing how portable their website is.

How could an attacker mount a denial-of-service attack against the SiteTester server, simply by visiting a single URL? Show in your answer the malicious URL that causes so much trouble. You can assume the SiteTester developers haven't taken any special precautions against denial of service.

Solution: `http://sitetester.com/?u=`$\underbrace{\texttt{http://sitetester.com/?u=}\cdots}_{\text{repeat many times}}$

With n repetitions, SiteTester will make 2^n HTTP requests. If n is large, this will likely overload the SiteTester server.

Your answer should not assume that SiteTest's server has XSS vulnerabilities or other software bugs.