CS2107 Tutorial 6
(Security Protocol: TLS and Its Renegotiation Attack)

School of Computing, NUS

11–15 October 2021

Objective

This tutorial looks into attacks on a protocol. The discussed attack below illustrates the subtlety and difficulty of designing a secure network protocol. As you will see, even a well-designed and widely-used protocol like TLS still suffers from several serious past attacks, which are summarized in `https://datatracker.ietf.org/doc/html/rfc7457`. (Note that you don't need to know all these attacks for our introductory module.)

Background on TLS and Its Renegotiation Attack

As discussed in our lecture, TLS client and server perform a *handshake* to establish cipher settings and also a session key for securing their subsequent communication. A vulnerability of the renegotiation procedure in the TLS' handshake protocol was discovered in 2009. An attack exploiting this vulnerability can cause *message injections*. It can allow the attacker, for instance, to splice/include his own requests into the beginning of the conversation that the TLS client has with the web server. (Note, however, that the attacker can't actually decrypt the TLS client–server communication. Hence, it is slightly different from a *typical/full* man-in-the-middle attack, who can also discover the communication content. Yet, in this particular attack, the attacker still act as a man-in-the-middle in relaying messages from a client in addition to injecting his own's.)

You have to prepare this tutorial by first reading the article "*Understanding the TLS Renegotiation Attack*" by Eric Rescorla from `https://web.archive.org/web/20191016205558/http://www.educatedguesswork.org/2009/11/understanding_the_tls_renegoti.html`. In the article, you can first see a description of the TLS handshake protocol, which has been described in the class. (Yet, we do not need to know the full details on the handshake to complete this tutorial.) The described attack allows an attacker to inject traffic into a legitimate client-server exchange, so that the TLS server will accept it as if it came from the client.

Answer the following questions related to the described attack.

1. What is the role of the server's public-private key pair? Note that there is no involvement of the client's public-private key pair in the authentication process. Why?

Solution

The server's public-private key pair is used to authenticate the server.
The client's public-private key pair is *not* involved in the authentication since it is a *unilateral authentication*.

2. How the attacker and client get to know the server's public key?

Solution

From the *server's certificate*, which is sent by the server during the TLS handshake.

3. What is the main purpose or result of the handshake protocol?

Solution

The purpose is to derive a set of secret keys that will be subsequently used for securing message communication between the client and the server.

4. Let k_1 and k_2 be the symmetric keys established during the first and second handshake, respectively. Does the attacker knows k_2?

Solution

No, because the second handshake is performed directly between the client and the server.

5. Which key is used to encrypt the "Initial Traffic"?

Solution

Key k_1, which is established in the first handshake.

6. The second handshake is partially encrypted. Which key is used to encrypt the second handshake?

Solution

The previously established key k_1 (from the first handshake).

7. Which key is used to encrypt the "Client Traffic"?

Solution

Key k_2 established in the second handshake, which was completed before the Client Traffic is communicated.

8. The vulnerability was discovered in 2009, and RFC 5746 was released in Feb 2010 to update the TLS/SSL protocol specification. Suppose that, sometime during late 2009, Alice uploaded her report via LumiNUS (during the "Client Traffic" in the renegotiation attack). Would the confidentiality of the report being compromised under the attack?

Solution

No. The renegotiation attack on TLS does *not* compromise confidentiality. However, it may breach the integrity of TLS client-server communication.

9. Bob was tasked with fixing the found vulnerability of the TLS protocol. Bob suggested the following:

In the original TLS's handshake, the very first message was:

- Client $\rightarrow$ Server: "Hello, I want to connect".

Bob wanted to change the above to:

- Client $\rightarrow$ Server: "Hello, I want to connect and this is the $\langle x \rangle$ handshake".

where $\langle x \rangle$ can be `first`, `second`, and so on. Whenever the server notices an inconsistency, it must then immediately abort. Bob claimed that the above protocol modification would prevent the renegotiation attack. Show that Bob was wrong.

Solution

The attacker, as the man-in-the-middle (MITM), can simply change `first` in the unencrypted message to `second`.

10. Bob realised the mistake. He noticed that in this attack, the client was the victim. So, he should give the victim the power to control the situation. In his second attempt, he then suggested the following:

In the original TLS's handshake, the second message was:

- Server $\rightarrow$ Client: "Ok, I am happy to connect. Here is my certificate and other information".

Bob wanted to change the above to:

- Server $\rightarrow$ Client: "Ok, I am happy to connect. Here is my certificate and other information. This is our $\langle x \rangle$ handshake".

where $\langle x \rangle$ can be `first`, `second`, and so on. Whenever the client notices an inconsistency, it must then immediately aborts. Show that Bob was still wrong.

Solution

Again, the attacker, as the MITM, can also change `second` to `first`.

11. Based on Bob's second attempt, suggest a way to prevent the renegotiation attack.

Solution

The server can sign the new message in Question 10 using its private key. Alternatively, if a secret key was previously established with the client, the server can MAC the new message in Question 10.

(Additional note: A *Renegotiation Indication Extension* was proposed for TLS. It requires the client and server to include and verify information about previous handshakes in any renegotiation handshakes. This extension has become a proposed standard as RFC 5746. It has been implemented by several libraries as well.)

12. While this instance of TLS performs a unilateral authentication, in many web applications, the clients still need to be authentication in some way. In the pizza-shop application, how does the client get authenticated?

Solution

As also mentioned in the article, most web applications perform an initial client authentication using a *client's username-password pair*, and then subsequently persist that authentication state using a *Session ID* sent as a *web/HTTP cookie*.

(Additional note: A web/HTTP cookie is a server-generated token. Once it's received by a browser, the browser will automatically attach the cookie in its subsequent requests to the server. This HTTP cookie will be discussed in our lecture on Web Security later.)

13. You were the web application developer who built the pizza-shop online site. When the renegotiation attack was made known, you knew that your application was vulnerable to the attack. Should you rely on the web server to fix and mitigate the attack (and thus you don't have to do anything), or should you "harden" your web application by modifying it? If you choose the latter, how should you prevent the attack? Recall that in the original application, the following is sent from the client's browser:

```
GET /pizza?toppings=pepperoni;address=attackersaddress HTTP/1.1
Cookie: victimscookie
```

What should be sent instead?

Solution

The following are possible new message formats to be sent by Client:

(a) Two GET commands instead of one. The first GET command can just perform a dummy operation to be omitted.

(b) The GET command includes (part of) the Session ID for server's verification.

(c) The GET command includes a value that is derived from the Session ID, e.g. MAC of the GET command and the Session ID, for server's verification.

Notice that any remedial solution from web developer would be messy.
Also notice that, even if it works, the application now has to take care of communication security. This is not desirable since it is against the *modularity principle* of a good system design. That is, communication security is supposed to be handled by the layer *below* the application layer.

14. When the renegotiation attack was made known, a very simple and effective suggested *short-term fix* was to disable the renegotiation in TLS. Are there any implications/limitations to simply disabling the TLS renegotiation feature?

Solution

It will affect *availability* since some applications may need the renegotiation feature of the TLS protocol.

— End of Tutorial —