C2107 Tutorial 6 (Protocol: Renegotiation attack)

School of Computing, NUS

March 15, 2021

1. Consider DNS spoofing. An attacker was able to trick a victim to send out 2^7 DNS queries asking the ip address of a particular domain, say `www.aaa.com`. The attacker was not able to sniff the query, but was able to send spoofed DNS responses to the victim. Since the attacker was unable to sniff, the attacker would not know the QID of each query and thus unable to construct valid responses.

   Nevertheless, this attacker, after tricked the victim to send out the queries, still sent 2^9 spoofed response to victim, each with a randomly generated QID.

   What would be the probability that the attack succeeded? (that is, at least of the responses matched one of the queries).

   Solution

   Using Birthday attack in tutorial 4 ($k = 2^7, m = 2^9, n = 16$).

   $$1 - 2.7^{-1} > 0.5$$

   How is this question related to the size of nonce in strong authentication?

   Solution

   Similar attack can be carried out on strong authentication. Attacker stores k past valid challenges in a database $\mathcal{D}$. Now, if victims issue m challenges to the attacker and size of nonce is n bits, by using the database to replay, the probability of succeed is at least 1 - 2.7^{-1}.

2. **Renegotiation attack.**
   This attack illustrates the subtly and difficulty in designing a security protocol. You have to prepare for this tutorial. Read Renegotiation attack on TLS :

   `http://www.educatedguesswork.org/2009/11/understanding_the_tls_renegoti.html`

   This attack exploits a vulnerability in how TLS handle the handshake, and it provides a way for the attacker to append any message. (see lecture note for TLS's handshake)

Solution

From the website:

```
Client                          Attacker                          Server
------                          -------                           ------
                                    <----------- Handshake ---------->
                                    <======= Initial Traffic ========>
<-------------------------  Handshake ============================>
<======================== Client Traffic ==========================>
```

Inserted by attacker:

```
GET /pizza?toppings=pepperoni;address=attackersaddress HTTP/1.1
X-Ignore-This:
```

Original request by victim:

```
GET /pizza?toppings=sausage;address=victimssaddress HTTP/1.1
Cookie: victimscookie
```

The combined request received by pizza shop:

```
GET /pizza?toppings=pepperoni;address=attackersaddress HTTP/1.1
X-Ignore-This: GET /pizza?toppings=sausage;address=victimssaddress HTTP/1.1
Cookie: victimscookie
```

(a) Is this a unilateral or mutual authentication? If unilateral, which entity, Server or the Client, carries out the verification?

Solution

Unilateral.
The Client carries out the verification.

(b) How the Client and Attacker get to know the server's public key?

Solution

From the Server's certificate, which is sent by the Server during the TLS handshake step.

(c) Let (k_1, t_1) and (k_2, t_2) be the sessions 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.

(d) Which key is used to encrypt the "Initial Traffic"?

Solution

Key k_1, which is derived in the first handshake.

(e) The second handshake is partially encrypted. Which key is used to encrypt the second handshake?

Solution

The previously-established key k_1, which is derived in the first handshake.

(f) Which key is used to encrypt the "Client Traffic"?

Solution

Key k_2, which is derived in the second handshake done before the Client Traffic is communicated.

(g) The vulnerability was discovered in 2009, and RFC 5746 was released in Feb 2010 to update TLS/SSL protocol specification. Suppose that, sometime during late 2009, Alice uploaded her report via Luminus. Later, Alice was very worried that someone would peep into her report. Could the confidentiality of the report be compromised?

Solution

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

(h) Bob was tasked to fix the vulnerability in the TLS handshake 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 x-th handshake".

where x can be `1`, `2`, and so on. Whenever the server notices inconsistency, it must immediately abort. Bob claimed that the above protocol modification would prevent the renegotiation attack. Show that Bob was wrong.

Solution

Attacker, as the man-in-the-middle (MITM), can simply change `first` to `second`.

(i) 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 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 x-th handshake".

Whenever the client notices inconsistency, it must immediately aborts. Show that Bob was still wrong.

Solution

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

(j) Based on Bob's second attempt, suggest a way to prevent renegotiation attack.

Solution

original TLS

(1). $C \rightarrow S$: hello
(2). $C \leftarrow S$: auth. key exchange data
(3). $C \rightarrow S$: auth. key exchange data
(4). $C \leftarrow S$: finish
(5). $C \rightarrow S$: finish

Note that step (4) and (5) are protected by the new session key.

fixed TLS method

(1). $C \rightarrow S$: hello
(2). $C \leftarrow S$: auth. key exchange data
(3). $C \rightarrow S$: auth. key exchange data
(4). $C \leftarrow S$: finish + this is the x-th handshake
(5). $C \rightarrow S$: finish + this is the x-th handshake.

Note that step (4) and (5) are protected by the new session key.
In the actual solution, a new field `Finished.verify_data` is included in the finish message, which serves similar purpose as above. The `Finished.verify_data` contains info of the pervious session (if any).
`https://tools.ietf.org/id/draft-mrex-tls-secure-renegotiation-00.txt`

(k) While this instance of TLS performs unilateral authentication, in many web applications, the clients still need to be authentication in some ways. In the pizza shop application, how is the client authenticated?

Solution

As mentioned in the article, most Web applications perform an initial client authentication via a client username/password pair, and then subsequently persist that authentication state with HTTP cookies (i.e. secret Server-generated tokens that are sent with any client's requests).

(l) 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 "harden" your web application by modifying your application? 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:

i. Two GET commands instead of one. The first GET command can just perform a dummy operation.

ii. The GET command includes (part of) the cookie for Server's verification.

iii. The GET command includes a value that is derived from the cookie, e.g. MAC of the GET command and the secret cookie, 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.

(m) When the Renegotiation attack was made known, a very simple and effective security was to disable renegotiation in TLS. Any implications and limitations to simply disabling TLS renegotiation?

Solution

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