CNS, Spring 2020 HW2

B07902064 資工二 蔡銘軒

Handwriting

1. SSL/TLS and Tor

   (1) In a POODLE attack, the following two conditions must be met:

   a) the attacker must be able to control portions of the client side of the SSL connection.

   b) the attacker must have visibility of the resulting ciphertext.

   A common way to achieve these conditions is to act as MITM.
   To perform the attack, the attacker repeatedly drops the session, and eventually the client will fall back to older protocols such as SSLv3.
   In SSLv3, CBC mode is used to encrypt data. The attacker can retrieve the plain text by performing the padding oracle attack.
   To avoid the attack as a client, one should use the cipher **TLS_FALLBACK_SCSV**, which tells the server that although the client is downgrading to a lower SSL version, it is capable of higher version. This signals to the server that a MITM attack is probably going on and hence the conections would be dropped by the server. Also, disabling SSLv3 altogether could prevent the attack.

   (2) a) Most of the time, the following events take place to establish a secure connection:

   i The user sends an unsecured HTTP request.

   ii The server answers via HTTP and redirects the user to a secure protocol (HTTPS).

   iii The user sends a secure HTTPS request, and the secure session begins.

   In a SSL Strippinng attack, the attacker intercepts the HTTP request from the client to the server, and go on establishing the HTTPS connection with the server itself, while maintaining a HTTP connection with the client. Essentially, this is an MITM attack where the client's connection (with the attacker) is stripped of the encryption.
   The HSTS policy specifies a period of time during which the user shall access the server in a secure-only fashion. If the user's browser sees that the connection is over HTTP, it will try to switch to HTTPS or it won't connect at all.

   b) When a new visitor comes, I assign it a unique ID and represent it in binary. For example, I assign a visitor ID number 2, which is 10 in binary.
   To record the value, I can redirect (this could be done in the background by javascript) the visitor to serveral other domains that I control, and use 1 to represent that the domain requires HTTPS and 0 otherwise. For example, to store the value 2, I redirect the visitor to `0-bit.com` and `1-bit.com`. `0-bit.com` does not require HTTPS for future connection while `1-bit.com` does.
   Next time the visitor visits, I redirect it to the same domains using HTTP and see which of them requires HTTPS. This way, I can retrieve the ID of the visitor.

   c) We could clear the browser's HSTS settings, trading security for privacy. Another mitigation, proposed by Apple and specific to Safari's users, is that we limit HSTS State to the Hostname, or the Top Level Domain + 1. This prevents the attacker from efficiently setting HSTS across large number of different websites.

   (3) a) A Tor Entry Guard Relay is the entry point to the TOR network. These relays are often stable and reliable, so they will not be easily compromised by attackers. The guards rotate every 4-8 weeks.

   b) Tor Bridge, or Bridge nodes, are nodes that are not listed on public directory of TOR nodes. These nodes are used in circumstances where the TOR service is restricted by censorship. The ideal situation is that the censoring party does not realize the existence of such nodes, while the users under censorship are able to identify them and use them to join the TOR network.

2. BGP

   (1) AS 999 could announce 10.10.220.0/24 to hijack the traffic. Since BGP gives higher priority to longer prefix, 10.10.220.0/24 would be a more preferable choice than 10.10.220.0/22.

   (2) a) AS 999 could announce {10.10.220.0/24, {AS 999, AS 2, AS 1, AS 1000}} to achieve the result in Figure 3.

       b) For AS 3, AS 4, AS 5, the update from AS 999 appeared to have longer prefix, so the traffic would go to AS 999. For AS 1 and AS 2, the update would not be accepted due to **Loop Prevention**, so the traffic would go to AS 1000.

   (3) **Advantage**: Besides from mitigating BGP prefix hijacking, it also protects the router from overload.
   Disadvantage: The original design of matching the longest prefix routes traffic to the place with more specific and accurate address. Imposing MPL disables this property and makes routing less efficient.

3. Mix Network

   (1) r_1: uncertain, r_2: uncertain, r_3: is s_1's recipient, r_4: is s_1's recipient, r_5: uncertain
   I inspected the given data, and found that at some moment, r_3 and r_4 appeared alone on the receiver side. So it was certain that all packets went to the only recipient. As for r_1, r_2, r_5, they always appeared with other recipients, so I could not be sure whether r_1 was talking to them or not.

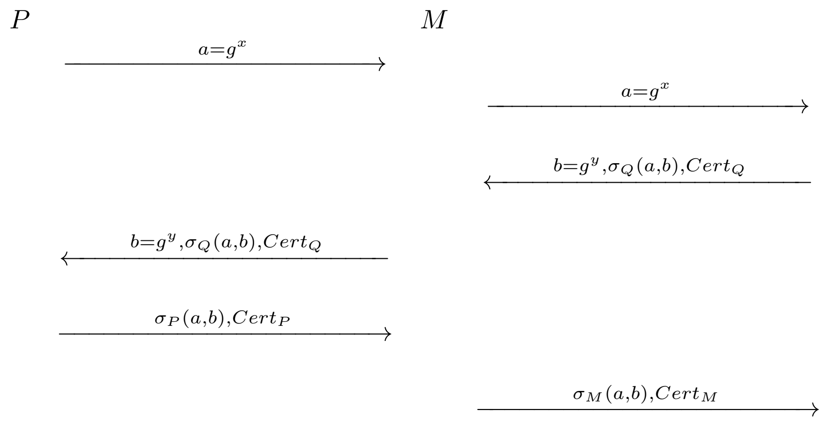
   (2) We could introduce some dummy traffic by arranging the network to always send another copy of the packet to the sender itself. This way, no single recipient will appear on the receiver side since the number of receivers will be at least as many as the senders. It is assumed that every user in the network can both send and receive packets. That is, a user can be a member in S and R at the same time.

4. STS Protocol

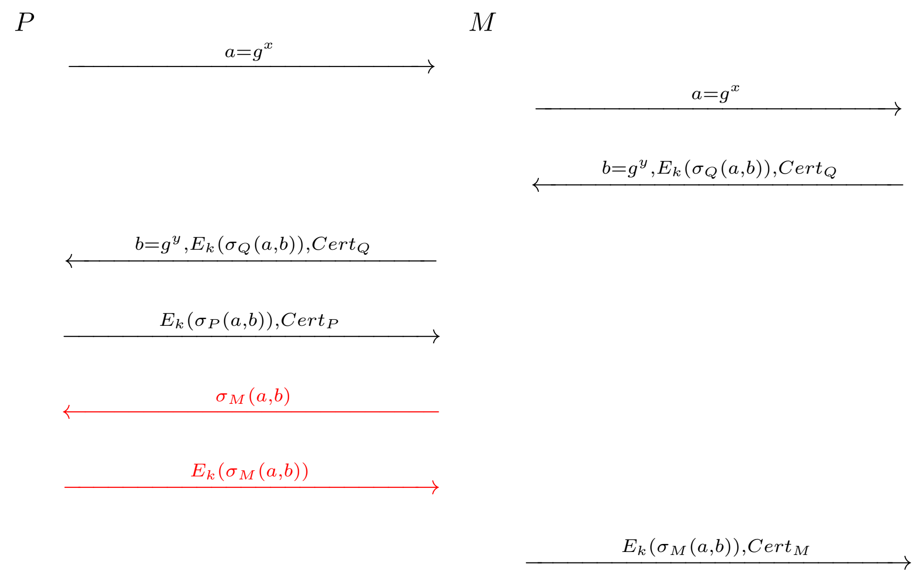
   Reference for this problem:
   Blake-Wilson, S., & Menezes, A. (1999, March). Unknown key-share attacks on the station-to-station (STS) protocol. In International Workshop on Public Key Cryptography (pp. 154-170). Springer, Berlin, Heidelberg.

   Diffie, W., Van Oorschot, P. C., & Wiener, M. J. (1992). Authentication and authenticated key exchanges. Designs, Codes and cryptography, 2(2), 107-125.

   (1) If the signatures are not encrypted, the *identity misbinding attack* can be performed as follows: (Let M be the adversary)

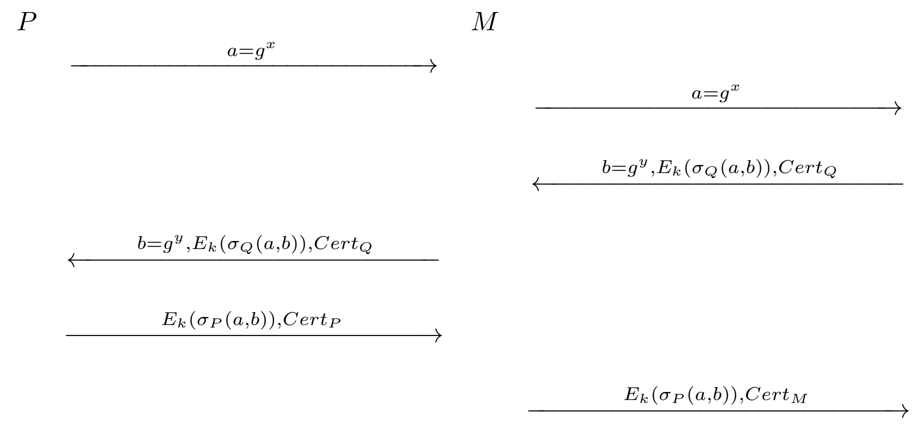


(2) The attack could be carried out as follows:


P
M
Q
$a{=}g^x$
$a{=}g^x$
$b{=}g^y, E_k(\sigma_Q(a,b)), Cert_Q$
$b{=}g^y, E_k(\sigma_Q(a,b)), Cert_Q$
$E_k(\sigma_P(a,b)), Cert_P$
$\sigma_M(a,b)$
$E_k(\sigma_M(a,b))$
$E_k(\sigma_M(a,b)), Cert_M$

The red part represents the query and the response of the encryption request.

(3) To perform the attack, the adversary M registers P's public key as his own. It is assumed that CA does not require the private key when one registers a public key. So M is able to register the same public key as P without knowing the private key. The attack takes place as follows:


P
M
Q
$a{=}g^x$
$a{=}g^x$
$b{=}g^y, E_k(\sigma_Q(a,b)), Cert_Q$
$b{=}g^y, E_k(\sigma_Q(a,b)), Cert_Q$
$E_k(\sigma_P(a,b)), Cert_P$
$E_k(\sigma_P(a,b)), Cert_M$

Since M's public key is the same as P's public key, when Q receives M's certificate and the public key, he could successfully decrypt the message from P, and thus be led to the illusion that the session key k is shared with M. A solution to this attack is to encrypt the certificate with the session key as well.

(4) The adversary M can send $x = 0$ to Q, and the session key between M and Q would be $g^{0\cdot y} = 1$. Then we have

$$\begin{aligned} E_k(\sigma_P(a)) =& E_1(\sigma_P(1)) \qquad (1) \\ =& E_1(1) \qquad (2) \end{aligned}$$

The first equation uses the fact that $a = g^0 = 1$ and the session key is also 1. And since signing in RSA is exponentiation, and $1^n = 1$ for all integers n, so $\sigma_P(1) = 1$. Now M can calculate $E_1(1)$ and send $E_1(1), Cert_P$ to Q to impersonates P.

Capture The Flag

5. Timmy & Amy

   (1) **Flag:** `CNS{this_is_TIMMinY_attack!!!!!}`
   Explanation: The key observation here is that the **Check()** function takes relatively longer time to run compared to others steps in the program.
   First of all, I had to determined the length of the secret number. This could be done by sending numbers with different number of digits to the server. If the length was correct, the server would run **Check()** against the first digit, which resulted in significant longer response time.
   Similarly, when trying to determine each digit, I enumerated from 0 to 9 and observed which one resulted in the logest response time.
   When I reached the last digit, I simply checked if the flag comes out when enumerating from 0 to 9.

   (2) **Flag 2:** `CNS{>Be_4ware_0f_the_$33D<}`
   Explanation: I used the fact that there are at most 128 different nonces that could be generated by the server. I made several connections to the server, each time with a random nonce N_a that was generated by the server in the previous connections.
   In each connection, the server would give me one of the possible 128 nonces, denoted by N_t, and the hash value determined by the pair (N_t, N_a). I checked if I had collected the hash value generated by the pair (N_a, N_t). If yes, I sent the value to the server and received the flag; otherwise, I recorded the hash value corresponding to the pair (N_t, N_a), inserted N_t into the list of the possible nonces, and moved on to the next connection.

6. TLS

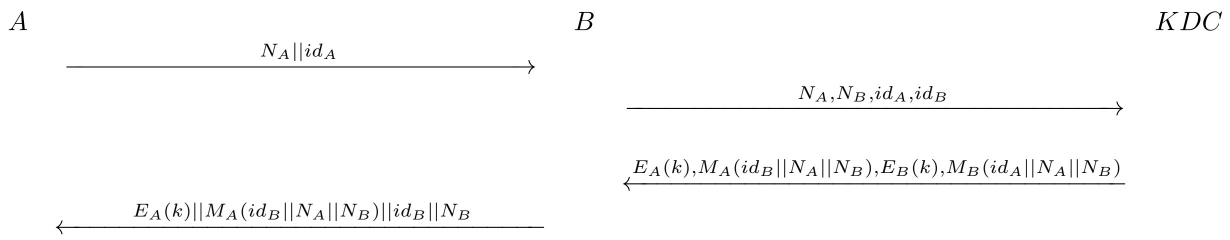
   (1) **Flag:** `CNS{Ch0053_CiPH3r_5UIt3S_C4rEfU1Ly}`
   Explanation: The problem was solved manually. I extracted the bytes of the certificate in the **.pcapng** file, and used the tool **openssl** to recover the certificate.
   With the n and e in the certificate, and the information that p, q were close to each other, I used the "Fermat's factorization method" to find p, q, and calculated d with p, q, e.
   After obtaining the private key, I turned the key into a PEM file and configured wireshark to use the file to decrypt the packets. Finally, I inspected each decrypted packets and found the flag.

   (2) The packets we can decrypt are encrypted with `TLS_RSA_WITH_AES_128_CBC_SHA`, but some other packages are encrypted with different cipher suite, e.g. `TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384`, so we could not decrypt them.

   (3) **Flag:** `CNS{LOSING_CA_PRIVATE_KEY_WILL_BE_A_DISASTER:(}`
   Explanation: In the previous problem, I obtained the certificate and the private key, which I turned into the files "cert.pem" and "key.pem", respectively.
   With the two files, I used the command
   ```
   ncat --ssl --ssl-cert cert.pem --ssl-key key.pem cns.csie.org 10223
   ```
   to impersonate the owner of the certificate and obtained the flag.

7. Key Exchange with KDC

   (1)


A
B
KDC
$N_A||id_A$
N_A, N_B, id_A, id_B
$E_A(k), M_A(id_B||N_A||N_B), E_B(k), M_B(id_A||N_A||N_B)$
$E_A(k)||M_A(id_B||N_A||N_B)||id_B||N_B$

(2) **Flag:** `CNS{M4n_1n_Th3_Middl3_4nd_r3pl4y_4tt4ck_t0_K3y_Exp0sur3_Att4ck}`
The flaw is that the protocol does not proctect the integrity of the session key. So the following relay attack allows other people to impersonate Bob.
Let C be the attacker.

$$\begin{tikzcd}
A \arrow[r, "N_A||id_A"] & C \arrow[r, "{N_A,N_C,id_A,id_C}"] & KDC \arrow[l, "{E_A(k),M_A(id_C||N_A||N_C),E_C(k),M_C(id_A||N_A||N_C)}"'] \\
{} & C \arrow[r, "{N_A,N_C,id_A,id_B}"] & KDC \arrow[l, "{E_A(k'),M_A(id_B||N_A||N_C),E_B(k'),M_B(id_A||N_A||N_C)}"'] \\
A & C \arrow[l, "E_A(k)||M_A(id_B||N_A||N_C)||id_B||N_C"'] & {}
\end{tikzcd}$$

(3) The protocol could include the key in the **mac** so that the attacker could not easily tamper with the key.

$$\begin{tikzcd}
A \arrow[r, "N_A||id_A"] & B \arrow[r, "{N_A,N_B,id_A,id_B}"] & KDC \arrow[l, "{E_A(k),M_A(E_A(k)||id_B||N_A||N_B),}"', "{E_B(k),M_B(E_B(k)||id_A||N_A||N_B)}"] \\
A & B \arrow[l, "E_A(k)||M_A(E_A(k)||id_B||N_A||N_B)||id_B||N_B"'] & {}
\end{tikzcd}$$

(4) **Flag:** `CNS{D0S_D00000S_D00000000S_T0_R4ce_C0nd1ti0n}`
Explanation: The main idea is to register the admin account before the server does. We know that the server resets all accounts every two minutes, so we keep trying registering the admin account, and if we get lucky, we will successfully register the admin account just at the moment when the server deletes all accounts, but has not yet registered the admin account. After we register the admin account, it is easy to obtain the flag.