Homework2

葉富銘　工海四 B08505045

1 SYN Cookies

1. SYN flooding attacks exploit the problem that server needs to kept state after receiving an initial SYN, so attacker can send several SYNs to fill server's connection table, hence deny other clients before timeout. SYN cookies compute SYN cookie per SYN request based on connection state and send back to client, so it needs no state kept.

2. Timestamp is added to ensure that the cookie is valid and has not expired, while client' s IP address helps to identify the user's device or computer, ensuring that the cookie is only sent to the device that originally requested it.

3. Everyone can use hash function to calculate SYN cookies, that is, SYN cookies can be forged by everyone. By using MAC, only server that owns the key can calculate SYN cookies.

2 BGP

1. AS999 can simply announce the same ip with longer prefix, such as 10.10.220.0/24 and advertises it to AS2, AS3 and AS5, and AS2 and AS5 will also propagate this advertisement to AS1 and AS4. Since 10.10.220.0/24 is longer than 10.10.220.0/22, thus all ASes will finally choose AS999 rather than AS1000.

2. a. AS999 sends update message {10.10.220.0/24, [AS999]} to AS3 and AS5.
   AS999 sends update message {10.10.220.0/24, [AS2, AS1, AS999]} to AS2 and AS1.

   b. Attacker first exploits longest prefix match property to let AS3, AS5 and AS4 forward their message to him rather than AS1000. To avoid AS2 and AS1 also forward their message to him, attacker prepends AS1 and AS2 on the route path to exploit loop prevention, let AS1 AS2 drop the advertisement.

3. The Maximum Prefix Limit feature can prevent BGP route hijacking and prevent a router from receiving more routes than the router memory can take.
   The disadvantage is that it can lead to suboptimal routing. If a BGP speaker is unable to receive all the available routes, it may not be able to select the best path for a given destination. This can result in increased latency, reduced throughput, and potentially suboptimal routing.

3 Knowing What I Am, Not Knowing Who I Am

1. Assume that public keys are safely distributed among users and the server. Even adversarial server has all public keys, it then computes $c_j = E_{pk_j}(\mathrm{w};r_j)$ for every j = 1, ... ,l and sends c_1, ... ,c_l to U_i, it will only receive $\mathrm{w}' = \mathrm{w}$ and can't get more information. Hence it can only guess U_i among l users, and since U_i is randomly and uniformly chosen from $\{U_1, \ldots, U_l\}$, so $\Pr[Expt^{anon}_{\prod,A,l}(\mathrm{n}) = 1] = \frac{1}{l}$, verifiable anonymity is achieved.

2. Divide users into (logl) groups and let users in the same group share the same key, hence the computation complexity will grows logarithmicly. The downside is the decreasement of anonymity since the number of anonymity set decreases.

3. I come up with two ways:

   (a) Server can perform passive eavesdropping. When server publish all public keys. Most user may not be so cautious and will only download the public keys which belong to others, so malicious server can simply observe which public key each user doesn't download to get the identity information.

   (b) Upon receiving all public keys, server can choose a pk_i, $1 \leq i \leq l$, to not publish, and compute (l - 1) ciphertexts without using pk_i. By this way, there must be a user which can't send response or wrong w′ back to server.

4 TLS

First use wireshark to observe packet, find server's public key and N. According to the hint, Fermat's factorization method factors N into p and q very efficiently if p and q of are too close to each other (p and q share half of their leading bits). I hence compute p and q, then compute private key and add it to wireshark to see the decrypted content, which contain root's private key.
After that, By following the procedure in this article, I then use root's private key to forge root's certificate, and use it to sign my personal key and generate valid certificate. Finally by obsersving data in decrypted packet to get user's name and password,I successfully impersonate legitimate users to gain access to the service.
flag : CNS{ph4Ul7y_K3y_93n3R471oN_15_D4N93rOU2!}

5 Little Knowledge Proof

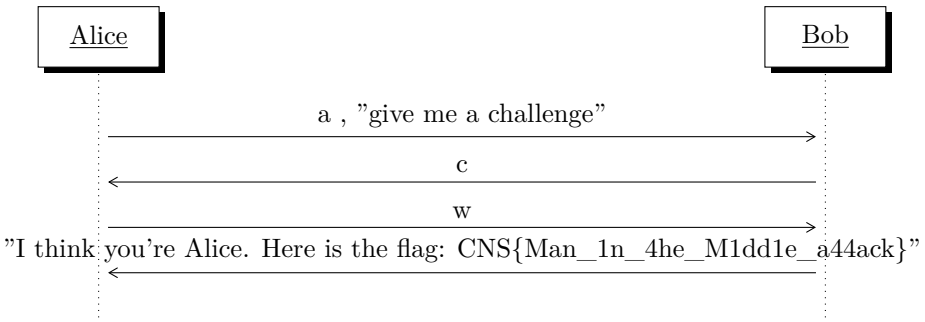

Figure 1: A simple sequence diagram

(a) The procedure is illustrated above, I simply implement Mitm, forward a to Bob, and forward c to Alice, and finally forward W to Bob to get the flag.
flag : CNSMan_1n_4he_M1dd1e_a44ack

(b) Since the implementation of LCG is deterministic(same seed produces same sequence of output) and it always uses the same seed, I can simply try c = 0 and c = 1 respectively to get x with formula w = cx + r.
flag : CNS{r&omne$$$hould_B_unp#ff0000ic\\tle}

(c) Since the p is prime, the order of Z_p^* is (p - 1). Then I factorize the order. Finally according to Pohlig–Hellman algorithm, it can be used to get the private key x. But I haven't implemented this part yet.

6 Clandestine Operation II

(a) 1. The client sends a request to the server.

   2. The server responds with a challenge, which is a random nonce generated by the server.

3. The client generates a response by hashing the challenge together with the user's password, then sends it to the server along with the user's username and domain name.
4. The server receives the response and verifies it by comparing it with the expected response, If the response is valid, the server grants access to the request.

This scheme hides user's password in response by combine it with other information such as challenge, target name and target info, and hash it together. So it can be authenticated without revealing its password.

(b) There are four steps for key derivation :

1. Master key negotiation :
   Either the LMv2 or NTLMv2 User Session Key will be employed as the master key.
2. Key exchange :
   The client selects a secondary master key, RC4-encrypts it with the base master key, and sends the ciphertext value to the server in the Type 3 "Session Key" field. This key will be used in following steps.
3. Key weakening :
   The key is simply by truncating the master key in step 2 to the appropriate length based on the flag.
4. Subkey generation :
   Up to four subkeys are established by the master key, either for server/client to create signatures for messages or encrypt messages.

NTLM2 signing with key exchange scheme will use both signing key and sealing key, which is created in subkey generation.

1. A sequence number is obtained; this starts at zero and is incremented after each message is signed. The number is represented as a long (32-bit little-endian value).
2. The sequence number is concatenated with the message; the HMAC-MD5 message authentication code algorithm is applied to this value, using the appropriate Signing Key. This yields a 16-byte value.
3. An RC4 cipher is initialized using the appropriate Sealing key. This is done once (during the first operation), and the keystream is never reset. The first eight bytes from the HMAC result are encrypted using this RC4 cipher.
4. A version number ("0x01000000") is concatenated with the result from the previous step and the sequence number to form the final signature.

(c) I just follow the procedure of NTLMv2 Authentication and rule of type1, type2 and type3 message stated in document, instead that I add the Negotiate Key Exchange flag and replace Negotiate 56 into Negotiate 128.
flag : CNS{p@33_tHe_ha3H!}

(d) I have implemented client signing and sealing key generation and signing procedure in Clandestine.py, but I can't pass the security check yet.

7 So Anonymous, So Hidden

(a) Use buffer with size 100 to store received packets, when the number of packets reachs 100, shuffle and send them to the destination according to the result of decrypt_server(self.sk), which is provided in lib.py.
flag : CNS{H3Y_Y0u_Ar3_A_m1x3R_ma573R}

(b) I modify create(message, send_to, pk) and add_next_hop(self, target, pk) in lib.py based on the mechanism in decrypt_server(self, sk) and decrypt_client(self, sk), and use them to create packet and prepend a hop to path in the reverse order of route.
flag : CNS{1_8r04dc457_7H15_m59_51nCe_y0U_D1dn7_91vE_7He_re7uRN_4ddRE55}

(c)

(d) First convert tor-pubkey into address to connect the hidden service, and use binary seach to search through all ports to find the flag. (I haven't implemented this yet.)