Spring 2023 Cryptography and Network Security

Homework 2

Release Date: 2023/04/10

Due Date: 2023/05/15, 23:59

TA's Email: `cns@csie.ntu.edu.tw`

Instructions

- This homework set is worth 115 points, including 15 bonus points.
- **Submission Guide:** Please submit all your codes and report to NTU COOL. Please refer to the homework instructions slides for information.
- You may encounter new concepts that haven't been taught in class, and thus you're encouraged to discuss with your classmates, search online, ask TAs, etc. However, **you must write your own answer and code**. Violation of this policy leads to serious consequences.
- You may need to write programs in the Capture The Flag (CTF) problems. Since you can use any programming language you prefer, we will use a pseudo extension code**.ext** (e.g., code.py, code.c) when referring to the file name in the problem descriptions.
- In each of the CTF problems, you need to find out a flag, which is in `CNS{...}` format, to prove that you have succeeded in solving the problem.
- Besides the flag, you also need to submit the code you used and a short write-up in the report to get full points. The code should be named **code{problem_number}.ext**. For example, code3.py.
- In some CTF problems, your solution may involve human-laboring or online tools. These are allowed as long as you get the flag not by cheating or plagiarism. If your code does not directly output the flag (e.g., it requires the user to do manual filtering on some messages), please specify the execution process of your code in **readme.txt** file.
- In some CTF problems, you need to connect to a given service to get the flag. These services only allow connections from 140.112.0.0/16, 140.118.0.0/16, and 140.122.0.0/16.

Handwriting

1. SYN Cookies (10%)

1) (3%) Explain why SYN cookies can mitigate SYN flooding attacks.
2) (4%) Explain why the cookie needs to contain a timestamp and the client's IP address.
3) (3%) Explain why using a Message Authentication Codes (MAC) is better than using a hash function to calculate SYN cookies.

2. BGP (20%)

In this problem, we would like you to explain and analyze some attacks against the BGP routing protocol. In both attacks, the attacker has control over AS999 and wants to attack AS1000. Figure 1 shows the routing paths in the normal state after AS1000 has announced 10.10.220.0/22. Each circle represents an Autonomous System (AS). A solid line indicates a link over which two neighboring ASes can exchange control messages such as BGP update messages. A dashed line indicates an established AS path to 10.10.220.0/22.

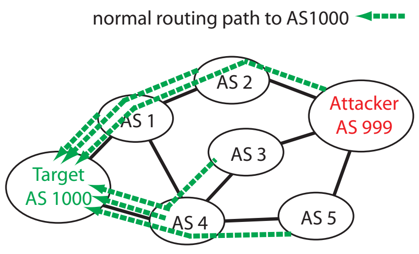

Figure 1: Normal scenario.

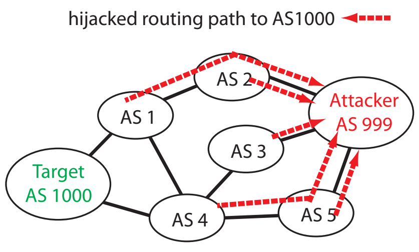

Figure 2: BGP hijacking.

1) (5%) Describe the most likely scenario that could explain the result of Figure 2. Specifically, what did AS999 announce?

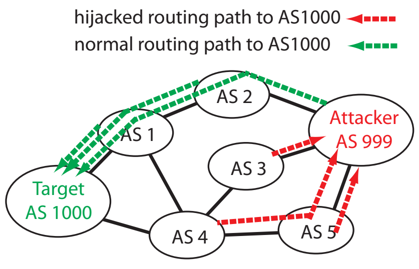
hijacked routing path to AS1000
normal routing path to AS1000
AS 2
Attacker
AS 999
AS 1
AS 3
Target
AS 1000
AS 4
AS 5

Figure 3: BGP man-in-the-middle.

2) In the second type of attack illustrated in Figure 3, the attacker can silently redirect the hijacked traffic back to the victim along the path indicated by green lines. This attack exploits **AS Path Prepending**, where an AS inserts AS numbers at the beginning of an AS path to make this path less preferable for traffic engineering purpose, and **Loop Prevention**, where AS x drops any BGP update with itself (i.e., AS x) in the AS-Path attribute to prevent routing loops.

   a) (5%) Instead of announcing the ownership of an address block, AS999 announces a fake BGP update. Specify a BGP update message (in the form of {IP prefix, [AS x, AS y, $\cdots$]}) that could cause the result of Figure 3.

b) (5%) Briefly explain how the attacker misuses path prepending and loop prevention for malicious purpose.

3) (5%) To mitigate BGP prefix hijacking, some propose the idea of BGP Maximum Prefix Limit (MPL). If a prefix length of an advertisement is longer than the MPL, the advertisement will be discarded. For example, if MPL is set to 24 in a BGP speaker, the announcement of 10.10.220.0/25 will simply be ignored. List one advantage and one disadvantage of MPL.

3. Knowing What I Am, Not Knowing Who I Am (15%)

Users want anonymity in the Internet. However, web services often require authenticated users to access. Can these paradoxical goals be achieved simultaneously? Following is a solution for *anonymous authentication* based on asymmetric cryptography:

- Notations: $E_{pk}(w;r)$ denotes the encryption of a string w using public key pk and a random nonce r. $D_{sk}(c)$ denotes the decryption of ciphertext c using secret key sk.
- Setup: Users $U_1, \cdots, U_l$ and a server S all have common l public keys $pk_1, \cdots, pk_l$ for an encryption scheme E, and U_j owns a private key sk_j for each $j \in [1, l]$.
- Input: U_i requests to authenticate himself to S
- Protocol:
  1. S chooses a random string w of length n and random nonces $r_1, \cdots, r_l$ such that each r_j is of the length needed to serve as randomness for encrypting w under E. S computes $c_j = E_{pk_j}(w; r_j)$ for every $j = 1, \cdots, l$ and sends $c_1, \cdots, c_l$ to U_i.
  2. Upon receiving $c_1, \cdots, c_l$, U_i computes $w' = D_{sk_i}(c_i)$ and sends back to S.
  3. Upon receiving w', S grants access if and only if $w' = w$. If access is granted, S sends $r_1, \cdots, r_l$ to U_i
  4. U_i verifies that $c_j = E_{pk_j}(w; r_j)$ for every $j = 1, \cdots, l$. If not, it outputs $\texttt{cheat}_S$ and halts. Otherwise, it is granted access and continues.

In this protocol, anonymity is ensured as the server cannot distinguish which of the ciphertexts the user chose for decryption, as long as the same w is used for every ciphertext. The user can verify if the server honestly use the same w, after he sends his identity. Since the anonymity can only be verified after the authentication process, this is called *verifiable anonymity.*

1) (5%) Please prove that this protocol achieves **verifiable anonymity**: The server should not know which user it is interacting without being detected cheating by the user.
Let's prepare some definitions for the proof:
   - $\Pi = (G, S, U)$ denotes a **protocol**, including a set of instructions for a user U, a server S and a public key generation algorithm G.
   - **Verifiable anonymity experiment** $\texttt{Expt}^{\texttt{anon}}_{\Pi,\mathcal{A},l}(n)$
     (a) The key generation protocol G is run l times and outputs l public-private key pairs $(pk_1, sk_1), \cdots, (pk_l, sk_l)$, associated with a set of authorized users $U_1, \cdots, U_l$ respectively.
     (b) A random i is chosen uniformly from $\{1, \cdots, l\}$.

(c) The adversarial server $\mathcal{A}$ is given $(pk_1, \cdots, pk_l)$ and interacts with U_i running Π and using private key sk_i.

(d) At the end of the experiment, $\mathcal{A}$ outputs an index $j \in \{1, \cdots, l\}$. The experiment result $\texttt{Expt}^{\texttt{anon}}_{\Pi,\mathcal{A},l}(n) = 1$ if and only if $j = i$ and U_i does not output $\texttt{cheat}_S$. In this case $\mathcal{A}$ is said to **succeed**.

- A protocol $\Pi = (G, S, U)$ is said to achieve **verifiable anonymity** if for every adversary $\mathcal{A}$ and every l,

$$\Pr[\texttt{Expt}^{\texttt{anon}}_{\Pi,\mathcal{A},l}(n) = 1] \leq \frac{1}{l}$$

2) (5%) The computation complexity of this protocol grows linearly with respect to the number of users. Can you improve the efficiency by perhaps trading off anonymity? *Hint: think of* ***k-anonymity***.

3) (5%) It is assumed that public keys are safely distributed among users and the server for this protocol to work. However, this can be more challenging than one thought. As one solution, we can let users upload their public keys onto the server, and let server publish those public keys so that any user can download. Assume users do not reveal their identities when uploading their public keys, this approach can still potentially compromise anonymity in the presence of a malicious server. Can you point the reason out? You can make necessary assumptions about the details of the key distribution process when answering this problem.

Capture The Flag

4. TLS (15%)

Former NTU CNS members operate a TLS-protected service at `cns.csie.org:12345`. They suspect that an attacker has successfully impersonated legitimate users to gain access to their service, but they cannot determine how the attacker bypassed the server's TLS protection.

As a current NTU CNS member, you are requested to aid in the investigation by examining their packet capture (pcapng) file, which is provided at `hw2/tls/tls_session`. You can open pcapng files with `Wireshark`. Once you have identified useful information from the pcapng file, you can use it to pass the server's identity check during the TLS handshake process.

Note that Ncat does not support TLS, so you may need to use other tools like OpenSSL to establish a connection to the server.

Hint: What if the two prime factors p and q of an RSA modulus n are too close to each other?

5. Little Knowledge Proof (10% + 5% Bonus)

Given a group $\mathbb{G}$ and a generator g, the CDH assumption states that given (g, g^a, g^b), it is hard to compute g^{ab} without knowing a or b. Diffie-Hellman Key Exchange and ElGamal are based on this assumption. Here, we proposed a zero-knowledge proof scheme based on this assumption.
Alice has an ElGamal private-public key pair $(x, y = g^x)$ in group Z_p^*. Now she want to prove to Bob that she owns the private key x. Here is how she does:

1. Alice: choose random r and send $a = g^r \pmod{p}$ to Bob.

2. Bob: choose a random challenge $c \in Z_p$ and send c to Alice.
3. Alice: compute $w = cx + r$ and send w to Bob.
4. Bob: verify that $g^w = y^c a \pmod{p}$. If it is correct, Bob believes that Alice knows x.

You can access Alice by `nc cns.csie.org 12346` and access Bob by `nc cns.csie.org 12347`. The challenge source is provided in `hw2/little-knowledge-proof`.

a) (5%) flag1: Please draw a sequence diagram to explain how they perform the zero knowledge proof (3%) and get the flag (2%).
Hint: for sequence diagram, you can use any drawing tools, or some markdown language, reference: https://hackmd.io/@codimd/extra-supported-syntax#Sequence-Diagram.

b) (5%) flag2: After learning Linear Congruential Generator in her Algorithm Course, Alice loves to implement random number generators herself. What is the problem with her random?
Hint: Linear Congruential Generator (LCG) is a pseudorandom number generator algorithm that generates random numbers. However, are these random numbers random enough? If you are interested in learning more about LCG, you can visit the following link: `https://en.wikipedia.org/wiki/Linear_congruential_generator`.

c) (Bonus 5%) flag3: Finding that you have reconstructed her secret, Alice decides to pick a new secret and makes the proof non-interactive. However, do you believe in CDH? You can find it out and get the flag.
Hint: find the order of the group. Can you factorize the order?

6. Clandestine Operation II (15% + 5% Bonus)

(This stroy continues from hw1...)
After entering Sanctuary of Surasthana, you found Nahida imprisoned in a huge tulip-shaped space! In addition to this gorgeous prison, there is a control panel that you can use to open its door. Nahida hinted you that it adopts NTLMv2 authentication, a challenge-response authentication mechanism used in Windows system. Also, it uses NTLM2 session security for subsequent communication. For more details, please refer to `hw2/clandestine-operation-II/README.md`.

You can access the control panel by `nc cns.csie.org 44397`. Some source files are provided in `hw2/clandestine-operation-II`.

a) (3%) Please explain how NTLMv2 authentication works. (You do not need to provide the technical details, e.g., how exactly the message is generated.)

b) (3%) Please explain how NTLMv2 signing with key exchange scheme works.

c) (9%) flag1: Umm... let's try to communicate with the control panel first. Also, by some means you successfully cracked the database and some data leaked. Can you take advantage of it?

d) (Bonus 5%) flag2: There is only one step away from success. Create the correct signature and rescue Nahida!

Important: you should upload your code, which will also be a grading criterion.
Note: We will not provide the server's complete code as that is what we want you to implement.

7. So Anonymous, So Hidden (15% + 5% Bonus)

a) (5%) You learned what mix network is in class. Suppose you are running a mix. Alice and other users will give you some packets, and it is your job to decrypt the packets and forward the packets to the next mixes. Access the service at `nc cns.csie.org 12804`, where you will get more instructions on how to interact with the service. You are also a great security engineer, and you know that timing analysis on the mix network is bad, so you want to make sure that there is less than a 1% chance that the adversary correctly guesses which incoming message corresponds to a specific outgoing message.

   Note: The packets come aperiodically, and the plaintext message starting with `CNS` would be sent when the challenge is over, either with a flag or a fail message.

b) (5%) Suppose you are Alice. You want to ask Bob to give you the flag. Finish the TODOs in `hw2/mix-tor/lib.py` and access the service at `nc cns.csie.org 12805`. The server source code is available as `hw2/mix-tor/main2.py`.

   Note: Despite its name, what the function `add_next_hop` actually does is "prepend" a hop to path. The first hop the packet is sent to is the last hop added to the packet. The TA is a terrible security engineer, so no delay is applied in this subproblem.

   For subproblem (b), besides the script used for generating packets, also upload your implementation of `lib.py` and rename it to `code6b-lib.py`. You may not change any other files.

c) (Bonus 5%) Eve, a sneaky hacker, has somehow intercepted a message between Alice and the first mix. But she fell asleep in class so she knows nothing about cryptography. Can you decrypt the message for her? Eve is at `nc cns.csie.org 12806` waiting for you to contact her. You will receive partial credit for pointing out possible weaknesses in the writeup without being able to crack the ciphertext.

   Note: There is no expected solution to this problem, although there are some intentional bad practices or designs in the implementation. You can perform any attack other than denial of service attacks.

d) (5%) Alice and Bob eventually learn that it is a bad idea to implement the security protocol themselves, so they decide to use a robust anonymous network with a sophisticated implementation. They decide to use Tor. Bob sets up a hidden service. The domain name looks so random that Bob thinks no one will ever find his cool service. Moreover, there are 65,535 ports, no one has time to try them all. He also published his public key online (at `hw2/mix-tor/tor-pubkey`); after all, it is a *public* key. Find out about Bob's cool hidden service and steal his flag!