Spring 2023 Cryptography and Network Security

Homework 3

Release Date: 2023/05/16

Due Date: 2023/06/13, 23:59

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

Instructions

- This homework set is worth 160 points, including 25 bonus points. You do not have to solve all the problems in order to get full points (100 points). Please take your interest into consideration and have fun!
- **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.

Fun Hands-on Projects

1. DDoS (40%)

In this problem, you are asked to analyze the packets under `hw3/ddos/` and answer the following questions.

We recommend you to use **Wireshark** to analyze the packets. Wireshark has many powerful analysis and statistics functions. If you prefer a command line interface, you can try **tshark**.

1. (5%) Observe the traffic in `ddos_1.pcapng`. When does the DDoS attack start? Find (1) the time of the first packet and (2) the victim's IP in this attack.

*Hint: You can use **I/O Graphs** to find the time that the flow starts to burst. Then you can find the first packet near there.*

2. (5%) Following the previous question, please find the protocol that the attacker exploits and the size of an attack packet.

   Hint: How to find attack packets if you know the victim?

3. (5%) DDoS attacks are usually launched with some specific intentions e.g., paralyzing portals, exhausting link bandwidth, etc. According to the flow statistics before and after the attack, please infer the victim type (server/client) and the intention of this attack.

4. (10%) Observe the traffic in `ddos_2.pcapng`. There are several victims in this DDoS attack. For each victim, find the (1) IP of the victim, (2) the number of the packets sent to the victim, and (3) three major amplifiers that send the most packets to the victim.

   *Hint: You can find some useful statistics in **IPv4 Statistics**.*

5. (10%) Now let's do an experiment to estimate the amplification factor. Choose an amplifier that has response in `ddos_1.pcapng`, and send a `monlist` query to it. Capture all the traffic and save it in a `ddos.pcapng` file. Remember to include this file in your submission. What is the amplification factor in your experiment?

   *Hint: You can use **nmap** or **ntpdc** to send a monlist query.*

   Note: An amplifier may change its status every time, and the amplification factor may differ. Also, different tools may send different payloads. So you just need to compute the amplification factor from your data.

   Note: Some of the amplifiers that have response in the data may not respond now. TA has got response from the following amplifiers:

   142.44.162.188
   91.121.132.146
   82.65.72.200
   81.23.0.110
   72.76.155.29
   61.216.81.26

   But we can't make sure that these amplifiers respond to you, so you may need to find other amplifiers from the pcapng file.

6. (5%) If you are the the operator of the amplifier (server), how can you prevent the type of attack analyzed in problem 1.2? If you are the network administrator of the victim's network, how can you protect the users from this attack?

2. Smart Contract (30% + 10% bonus)

We have prepared 6 fun challenges for you to get familiar with smart contracts on Ethereum. Enjoy!

Rules

- The source code of the challenges is provided in hw3/smart-contract.

- All challenges are included in a single contract named `CNSChallenge`, which is deployed on Sepolia testnet instead of Ethereum mainnet.

  `CNSChallenge` address: 0xc7e1A176060e0A92148845dae55F5614cDcE3Ca4

  `CNSToken` address: 0x9a125190981b318C9c855303978530D2c32487d2

  `CNSWallet` address: 0xfDa3E9Aab0bf395E176BFd56f04b09E9D6Fdf3Ff

- Please pass in your student ID (lower case, ex: r10922000) as a parameter while solving the challenges. Make sure your score is increased and recorded on the smart contract. DO NOT use others' student ID, which will lead to serious consequences. If your student ID is taken by others, please contact CNS TAs.
- There is no standard solution for each challenge. But cheating is not allowed (e.g., asking your classmate to give you CNS token). Remember that all transactions are recorded on the blockchain.
- Please describe your solutions and show your account address in the report. And submit your codes (if any) as code2-*.ext for any challenge you solved.
- If you have any questions, feel free to post them on the NTU COOL discussion forum. FAQ: `https://hackmd.io/@soyccan/cnshw3`.

Resources

- Remix - The recommended IDE to develop, deploy and interact with smart contracts. Choose "Injected Provider" to sign the transactions using the private key from your crypto wallet (e.g. MetaMask).
- MetaMask - A crypto wallet that keeps a user account's private keys for signing transactions on behalf of the user account.
- You can get FREE ethers from a Sepolia faucet:
  - Alchemy Sepolia faucet
  - QuickNode Sepolia Faucet
  - PoW faucet
- CryptoZombies - Learn Solidity by making a game.
- (Optional) Libraries for scripting: web3.js, ethers.js or web3.py
- (Optional) For the bonus challenge, you may need to understand basics of Ethereum Virtual Machine (EVM):
  - evm.codes: Reference manual of EVM opcodes
  - EVM Deep Dives: The Path to Shadowy Super Coder - Part 1
  - Ethereum EVM Illustrated (2018)
  - The EVM Chapter in the Mastering Ethereum book

Hint: All the data in the smart contract is public to everyone, even it's declared as a private variable.

3. Web Authentication (35% + 5% Bonus)

You must have used some type of web authentication method before. To download this document from NTU COOL, you will need to log in to your NTU account, which will authenticate that you are a legitimate student in this course. This challenge requires you to implement, attack, and analyze some web authentication methods.

a) (6%) In this problem, you need to implement three authentication methods: The 'Basic' HTTP Authentication Scheme (RFC 7617), cookie-based authentication, and JWT-based authentication. An automatic user will attempt to access your service, and your service should grant access to endpoint `/`, i.e., return status code 200, if the user has a legitimate token or correct password, and deny access, i.e., return status code 401, otherwise. The automatic user may attempt to access `/` with an invalid password or token. Access the judge at `nc cns.csie.org 17501`. You are **allowed** to use publicly available open source libraries and frameworks, but if you do, make sure you reference them properly in the writeup.

   Note: The automatic user is located at `140.112.31.97` and will not access any endpoints other than `/`. The automatic user only checks the status code. The TA will read your code. You will not get credit for trying to bypass the automated check without proper implementation.

b) (5%) Discuss how three authentication methods used in subproblem (a) work. And then compare them, providing at least one pro and con for each method. Also, justify your statements.

c) (8%) Alice implemented a great web service that you can access at http://cns.csie.org:17503/. It uses the JWT stored in the cookie to authenticate users. You can only use `guest / guest` to access the service, but one of the flags is hidden in the account with the username "admin" and another is stored somewhere else that Alice mistakenly thought was private. Try to get two flags.

d) (8%) Time-based one-time password (TOTP) defined in RFC 6238 is a standard algorithm that generates a one-time password (OTP) based on a timestamp. It is widely used in two-factor authentication (2FA) systems. In this challenge, you will implement the TOTP algorithm. You can access the challenge at `nc cns.csie.org 17504`. In addition, briefly explain your implementation.

   You are encouraged to implement the TOTP algorithm yourself. Using publicly available open source TOTP libraries is allowed, but would result in some points being deducted. Using them for reference is allowed and encouraged.

e) (8%) In addition to directly attacking the authentication protocol, brute force is another common attack. In a brute-force attack, an attacker submits many passwords in the hope of eventually guessing one correctly. There are some common ways to prevent brute force attacks, such as adding a captcha challenge, bot detection, or rate limiting. Bob has implemented a great web service that you can access at http://cns.csie.org:17505/. However, only `admin` has the flag. Try to brute force the service with this word list. The correct password is guaranteed to be in the list.

   Hint 1: There are strings in the cookie that look like hashes, what could they be?
   Hint 2: If you failed to figure out what hint 1 means, here's another method. It's the era of Machine Learning. Even babies know what Convolutional Neural Network is.
   Hint 3: What are some common ways to get the user's IP when the web service is behind a reverse proxy? Are these common practices secure?

f) (Bonus 5%) Pick a modern authentication method not mentioned in the class and in this problem, explain it in detail, and discuss how it is better than traditional methods, or how it was supposed to help but failed. Your score on this question will depend on how comprehensively and accurately you explain it.

4. Accumulator (30% + 10% Bonus)

In the class, we have already learned about Merkle Trees as a way to provide a proof of membership. However, when verifying the proof using Merkle Trees, we need to verify the nodes along the tree path, resulting in a time complexity of $O(\log n)$. In this question, we will introduce another proof of membership technique called Accumulator, which can achieve constant-time verification.
The concept of the Accumulator is as follows:

1. Let's assume that the set of elements we want to prove is a set of primes. We want to provide a digest that represents these primes, and this digest should be able to prove membership and non-membership. A naive way is to provide the product of these primes, and the proof of non-membership/membership only requires proving whether a prime is factor of the digest.

2. Although this method can provide correct proofs, further optimization is needed. If we simply use the product of each number, it is obvious that as the number of members to be proved increases, our digest will also become larger. To solve this problem, we can use a simple approach - making all of our operations under a modular, which will keep our digest within a certain range.

3. However, if we simply take the modulus of a number, attackers can simply create fake proofs. To address this issue, we move our digest to the power of a generator of an RSA Group. Let g be a generator of a RSA group, S be the set of primes we want to prove, we have $d = g^{\prod_{s \in S} s}$ as our digest. In this way, the value of the digest will be automatically modulated by the order of the RSA Group.

4. To make a membership proof for m in S, we can calculate the product of all primes except for m as the proof. That is $p = g^{\prod_{s \in S/\{m\}} s}$. For verifying, the verifier just need to check that
$$p^m = g^{(\prod_{s \in S/\{m\}} s) \cdot m} = g^{\prod_{s \in S} s} = d$$
is the digest provided.

5. To make a non-membership proof for m not in S, we can find a, b satisfying $a \cdot m + b \cdot (\prod_{s \in S} s) = 1$ with the extended GCD. And then, we can use (g^a, b) as our proof. For verifying, the verifier just needs to check that
$$(g^a)^m \cdot d^b = g^{a \cdot m} \cdot g^{b \cdot (\prod_{s \in S} s)} = g^{a \cdot m + b \cdot (\prod_{s \in S} s)} = g$$
is the generator.

6. Finally, we only need to map the content into a unique prime by Hash function. By doing so, we can complete our Accumulator.

This question has two parts. In problem (a), you will have to implement an Accumulator based on RSA Group. In problem (b), (c), and (d), you will have to make fake proofs to solve the challenges.

a) (14%) In this problem, you are going to implement the class Accumulator in `hw3/accumulator/accumulator.py` (fill in the TODO parts in the code provided). Briefly explain your implementation and provide the output of your script.

b) (8%) In this problem, you are going to make a fake membership proof to cheat the server. You can access the server by `nc cns.csie.org 4001`. The source code is provided in `hw3/accumulator/challenge1.py`.

c) (8%) In this problem, you are going to make a fake non-membership proof to cheat the server. You can access the server by `nc cns.csie.org` 4001. The source code is provided in `hw3/accumulator/challenge1.py`.

d) (Bonus 10%) While the Accumulator can achieve constant-time verification, it requires the elements in the member set to be primes, which needs additional time for the hash-to-prime operation. To address this issue, another approach is to use a Divisor-Intractable Hash function. This kind of functions ensures that although the elements may not be coprime, each element is not a factor of the product of any other arbitrary elements. You are going to make a fake membership proof to cheat the server with not-well-implemented Devide-Intractable Hash function. You can access the server by `nc cns.csie.org 4002`. The source code is provided in `hw3/accumulator/challenge2.py`.

If you are interested in learning more about Accumulator, you can read the following papers:

1. *Helger Lipmaa. Secure accumulators from euclidean rings without trusted setup. In Feng Bao, Pierangela Samarati, and Jianying Zhou, editors, ACNS 12, volume 7341 of LNCS, pages 224–240. Springer, Heidelberg, June 2012.*

2. *Boneh, D., Bünz, B., Fisch, B. (2019). Batching Techniques for Accumulators with Applications to IOPs and Stateless Blockchains. In: Boldyreva, A., Micciancio, D. (eds) Advances in Cryptology – CRYPTO 2019. CRYPTO 2019. Lecture Notes in Computer Science(), vol 11692. Springer, Cham.*