Spring 2024 Cryptography and Network Security

Homework 3

Release Date: 2023/05/08

Due Date: 2024/06/3, 23:59

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

Instructions

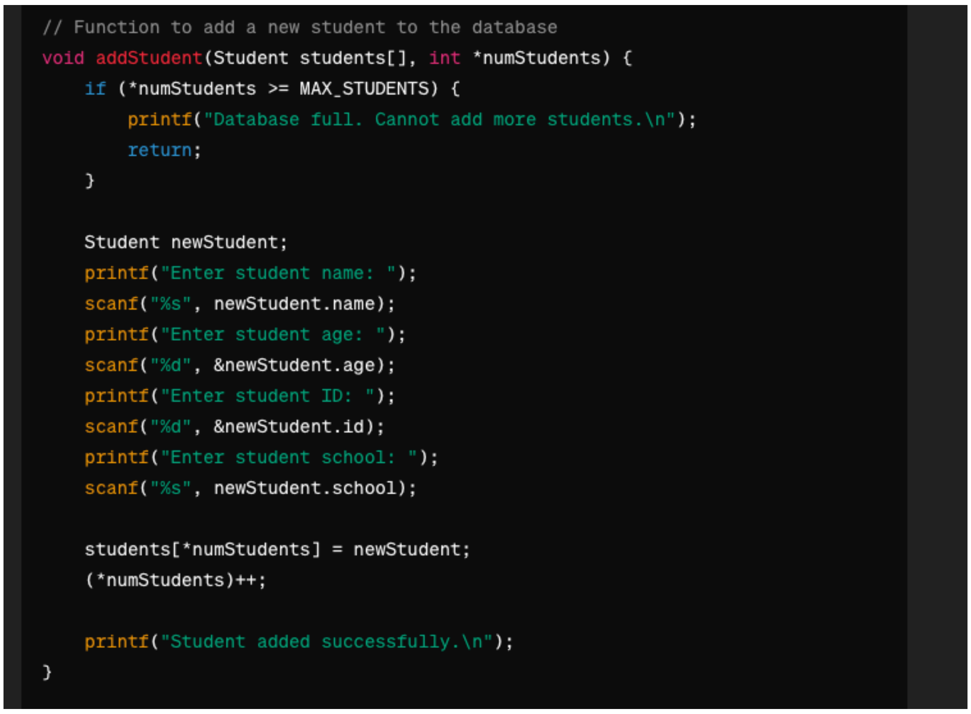
- This homework set is worth 175 points, including 16 bonus points. You do not have to solve all the problems in order to get full points (100 points). Please take your interest into consideration and have fun!
- **Submission Guide:** Please submit all your codes and report to NTU COOL and Gradescope. 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. Welcome To Fuzzing! (35%)

Despite the existence of numerous frameworks and libraries, software implementation flaws continue to emerge in the contemporary era. Many individuals rely on pair programming with LLM, such as utilizing code provided by chatGPT or copilot. Figure 1 presents a brief exercise to illustrate the most prevalent software implementation flaws.

One straightforward way to prevent these implementation flaws is to do a software testing. Nevertheless, manual software testing is labor-intensive and impractical in large-scale projects



```c
// Function to add a new student to the database
void addStudent(Student students[], int *numStudents) {
    if (*numStudents >= MAX_STUDENTS) {
        printf("Database full. Cannot add more students.\n");
        return;
    }

    Student newStudent;
    printf("Enter student name: ");
    scanf("%s", newStudent.name);
    printf("Enter student age: ");
    scanf("%d", &newStudent.age);
    printf("Enter student ID: ");
    scanf("%d", &newStudent.id);
    printf("Enter student school: ");
    scanf("%s", newStudent.school);

    students[*numStudents] = newStudent;
    (*numStudents)++;

    printf("Student added successfully.\n");
}
```

Figure 1: Can you identify which lines are vulnerable?

because it requires manual code inspection and the construction of a payload to demonstrate the existence of a vulnerability (i.e., proof-of-concept (PoC)).

In this context, we introduce fuzzing (also known as fuzz testing), an automated software testing technique that involves providing invalid, unexpected, or random data as inputs to a computer program and monitoring for exceptions such as crashes, failing built-in code assertions, or potential memory leaks.

In this problem, we will utilize the well-known grey-box fuzzer American Fuzzy Loop (AFL) as a tool to identify vulnerabilities in a C program. The program is provided in `fuzzing/app.c`.

a) (5%) Briefly explain the common workflows of grey-box fuzzing process in high level (You may take AFL fuzzer as example) and explain the difference between mutation-based and generation-based fuzzers.

b) (30%) There are at least **three different types** of implementation flaws that cause the vulnerable C program might crash or hang. Set up the AFL fuzzer and use it to find the PoC that can trigger these three different types of vulnerability. Furthermore, you must provide an analysis that explains the types of vulnerability triggered by each PoC and identifies the vulnerable areas in source code.
Note1: Please note that in order to solve this problem, you ***MUST*** *use either the AFL or AFLplusplus fuzzer. Failure to do so will result in a score of 0 points. For details of AFL or AFLplusplus, you may refer to* *`https://github.com/google/AFL`* *and* *`https://github.com/AFLplusplus/AFLplusplus`*
Note2: It is not necessary to invest significant time and resources in reviewing the source code in great detail. Instead, you should be able to locate the vulnerable code by using the appropriate tools. AddressSanitizer (ASan) and GNU Debugger (GDB) should be your best friends in this problem.
Note3: It is recommend to run AFL/AFLplusplus in Virtual Environment (either Virtual Machine or Container).

Note4: Please note that submitting multiple PoCs that trigger the same vulnerability will only be ***counted as one****. As example, submitting multiple PoCs that trigger stack buffer overflow will only receive points for finding Stack Buffer Overflow. To receive full points in this problem, please provide three PoCs that trigger three different vulnerabilities, respectively.*
Note5: An example of answer is provided in `fuzzing/example.md`*.*

2. Needham-Schroeder Protocol (20%)

The Needham–Schroeder Symmetric Key Protocol is based on a symmetric encryption algorithm. It forms the basis for the Kerberos protocol. This protocol aims to establish a session key between two parties on a network, typically to protect further communication. You can visit the following link to learning more details about the protocol: `https://en.wikipedia.org/wiki/Needham-Schroeder_protocol`.
In this problem, you have to implement the protocol to communicate with the key distribution center (KDC) and Bob. You can access Bob by `nc cns.csie.org 23472` and access KDC system by `nc cns.csie.org 23471`.
The challenge source is provided in `needham-schroeder-protocol/`.

a) (7%) flag1: Please draw a sequence diagram to explain the protocol (2%). Then, register an account on the KDC and implement the protocol to communicate with Bob. If you implement the protocol and communicate with Bob correctly, you will receive ciphertext containing a flag encrypted with a symmetric secret key (5%).
Hint1: for sequence diagram, you can use any drawing tools, or some markdown language, reference: `https://hackmd.io/@codimd/extra-supported-syntax#Sequence-Diagram`*.*
Hint2: If you are unsure about the details of the protocol, you may be able to check the source code of the KDC server and bob

b) (5%) flag2: You may notice that it is possible to log in to the KDC as admin. However, you don't have the admin password. Fortunately, you have the KDC source code, and the hashed password is in it. Please briefly explain what offline password cracking is (2%) and try logging in as admin to get the flag (3%).
Hint: The password must be inside the following wordlist: `https://github.com/kkrypt0nn/wordlists/blob/main/wordlists/passwords/common_roots.txt`
Warning: Remember to perform ***offline*** *password cracking instead of* ***online****, your request may be blocked if you perform* ***online*** *password cracking and gain 0 points on this problem.*

c) (8%) flag3: Once you have logged into the KDC as an admin, you can view the past sessions of communication between Alice and Bob. Can you use this information to impersonate Alice and communicate with Bob to get the flag (4%)? Besides, briefly explain why the protocol is vulnerable to this attack, where the implementation flaw is in the KDC server, and how to fix it (4%).

3. Accumulator (34% + 6% Bonus)

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

Suppose our set S, and every element in S is a prime number, one can simply use $\prod_{s \in S} s$ as a digest, then to prove that a prime $s \in S$, one only needs to check whether this digest is divisible by s, however, the size of the digest would increase as S grows larger, so we can consider the product under some modulus n, however, it would be easy to generate fake membership proofs by taking the inverse, so we can consider something similar to RSA, in particular, we first generate our RSA parameters $p, q, n := pq$ and a generator g, then we can let the digest be

$$d(S) = g^{\prod_{t \in S} t} \mod n.$$

To prove the membership of $s \in S$, let the proof be

$$\pi = g^{\prod_{t \in S \setminus \{s\}} t},$$

then one can verify this proof by checking whether π^s is the same as the digest.

a) (3 Points) Show that one can create fake membership proof if p and q are known.

Note that we could also prove the non-membership of a prime $u \notin S$, since by Bézout's theorem, there is a, b such that $au + b \prod_{s \in S} s = 1$, and a, b could be found using the extended GCD algorithm, then let $\pi = (g^a, b)$, then one can verify this by checking if $(g^a)^u \cdot d^b = g$.

b) (3 Points) Show that one can create fake non-membership proof if p and q are known.

Now to make this accumulator work for general elements, one could map the element to a unique prime via a hash function, and the resulting accumulator is called the RSA accumulator.

Next, we are going to introduce bilinear-based accumulators.

Definition 1. Let G_1, G_2, G_3 be groups, a bilinear function $f : G_1 \times G_2 \to G_3$ is a function satisfying

- Linear w.r.t. G_1, i.e., $f(g_1^c, g_2) = f(g_1, g_2)^c$ for all $g_i \in G_i$, $c \in \mathbb{Z}$
- Linear w.r.t. G_2, i.e., $f(g_1, g_2^c) = f(g_1, g_2)^c$ for all $g_i \in G_i$, $c \in \mathbb{Z}$

Essentially, bilinear means that the function is linear with respect to both inputs. In order for the bilinear map to be useful, we require that it is non-degenerating, i.e. the linear maps obtained by fixing either one of the inputs are non-degenerate, in other words, $f(x, y) = 1$ for all $x \in G_1$ implies $y = 1$, and $f(x, y) = 1$ for all $y \in G_2$ implies $x = 1$, and that it is efficiently computable.

Now let g_1 and g_2 be a generator of G_1, G_2, respectively, then let c be a random secret, then we can let the digest be

$$d(S) := g_1^{\prod_{s \in S}(c-s)}.$$

Then to prove the membership of $s \in S$, let the proof be

$$\pi = g_1^{\prod_{t \in S \setminus \{s\}}(c-t)},$$

c) (3 Points) Let $f : G_1 \times G_2 \to G_3$, where G_i are cyclic groups, be a bilinear map, show that one can verify the proof π of $s \in S$ given the digest $d(S)$, g_2, and g_2^c.

d) (3 Points) Show that one can create fake membership proof if c is known.

To prove the non-membership of $u \notin S$, it is a bit more complicated, consider the polynomial $p(x) := \Pi_{s\in S}(x-s)$, let $b = p(u) \neq 0$, then u is a root of $p(x) - b$, hence we can rewrite $p(x)$ as $q(x)(x-u)+b$. Let the proof be

$$\pi = \left(g_1^{q(c)}, b\right)$$

e) (3 Points) Show that one can verify the proof of π of $u \notin S$ given the digest $d(S)$, g_1, g_2, and g_2^c.

f) (3 Points) Show that one can create fake non-membership proof if c is known.

The above construction is called a bilinear-based accumulator, in practice, bilinear accumulators outperform RSA accumulators in terms of speed. However, to use this accumulator, we still need to find the groups G_1, G_2 and G_3, let's see if we can apply this to elliptic curves, given an elliptic curve $E : y^2 = x^3 + ax + b$, let $E(\mathbb{F}_p)$ be the corresponding elliptic curve group over the field $\mathbb{F}_p$, one might be tempted to take $G_1 = G_2 = E(\mathbb{F}_p)$ and look for a non-degenerate pairing function, however, expect for super-singular curves, it is likely to not exist.

To apply the above construction to elliptic curves, we need to look for $G_1 \neq G_2$, let g_1 be a generator which generates a subgroup of prime order q of $E(\mathbb{F}_p)$), without loss of generality, we may assume that our group $E(\mathbb{F}_p)$) is also of order q, otherwise, we can restrict the group in the following discussion to the subgroup generated by g_1 instead. We would want to find cyclic groups $G_2 \neq G_1$ which have the same order, and are points on the same curve, we can just let G_1 be the group generated by g_1, for G_2, we would want to find an element of order q which is not in G_1, but then there are no other groups on the same curve, or is there?

To find G_2, we need to dive deeper. Recall that for a general polynomial in $\mathbb{R}$, it is not necessary to find all the roots of the polynomial in $\mathbb{R}$, since some roots might be in $\mathbb{C}$, we call $\mathbb{C}$ the algebraic closure of $\mathbb{R}$, where the closure contains any roots of any polynomial over $\mathbb{R}$. To find more points, we can apply the same trick to $\mathbb{F}_p$, and consider the curve over $E(\bar{\mathbb{F}_p})$ instead. Now let the sets of elements x of a group G with ℓx being the identity function be the set of ℓ-torsion elements of G. For example, in $E(\mathbb{F}_p)$, every element is a q-torsion element. And it turns out the q-torsion elements of $E(\bar{\mathbb{F}_p})$ are contained in $E(\mathbb{F}_{p^k})$ for some k, the smallest k is called the embedding degree of the curve, and there are exactly q^2 torsion elements, we won't go through the details for how to find such elements, but once we obtain an q-torsion elements g_2 of $E(\mathbb{F}_{p^k})$ not in $E(\mathbb{F}_p)$, then we can take G_2 to be the group generated by g_2, and consider the Weil's pairing.

g) (4×4 points) Implement generation and verification for membership/non-membership proofs of the above construction, connect to `cns:csie.org:9721` to get the parameters of the curve, there are four flags in total, one for each method implemented.

This construction is not just useful for building accumulators, but is a whole field called pairing-based cryptography, in the next exercise, you will see an application of this construction.

Given an elliptic curve over $\mathbb{F}_p$, let E be the corresponding elliptic curve group and n be its order, Weil's paring is a bilinear map $f : E(\mathbb{F}_p) \times E(\mathbb{F}_{p^k}) \to \mathbb{F}_{p^k}^\times$ where k is the smallest integer satisfying $n | p^k - 1$, k is also called the embedding degree of the curve.

h) (3 points) Show that if a curve has a small embedding degree, we can use Weil's pairing to solve a discrete log problem faster than methods on the elliptic curve.

i) (3 points) Compute the embedding degree of the following curves:

- (secp256k1) $y^2 = x^3 + 7 \mod (2^{256} - 2^{32} - 2^9 - 2^8 - 2^7 - 2^6 - 2^4 - 1)$

- (Curve25519) $y^2 = x^3 + 486662x^2 + x \mod (2^{255} - 19)$

Are they vulnerable to such attacks?

4. DDoS (40%)

In this problem, you are asked to analyze the packets under `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

   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?

**The packets and the problem design are sponsored by NTU Computer and Information Networking Center*

5. Private Information Retrieval (15% + 5% Bonus)

Here are some resources that may be useful for this problem:

- Sage for matrix operation over finite fields.
- An introduction to private information retrieval by Dima Kogan.
- Lecture notes by Daniele Micciancio and by Vinod Vaikuntanathan on the learning with error problem and homomorphic encryption.

To solve this problem, please connect to `cns.csie.org 44444`

Is it possible to search for a piece of information without revealing what you are looking for? That is the goal of Private Information Retrieval (PIR). In this problem, you have to implement a minimal PIR protocol, and act as a server to provide information to clients.

In this protocol, there are two non-colluding servers S_0 and S_1, each holding a copy of the database $X \in \mathbb{Z}_2^{m\times m}$. That is, X is a bit matrix of size m times m. We use X_j to denote the j-th column of X.

When a client wants to query some position $(i, j) \in [m] \times [m]$ of the database X, they choose a uniformly random query vector $\mu_0 \in_R \mathbb{Z}_2^m$, and compute the complement query vector

$$\mu_1 = \hat{j} + \mu_0,$$

where $\hat{j} \in \mathbb{Z}_2^m$ is the unit vector with one in the j-th entry and zero everywhere else.

When server S_b receives the query vector μ_b, it replies with

$$r_b = X\mu_b \in \mathbb{Z}_2^m.$$

Finally, the client can reconstruct the desired position by taking the i-th entry of the vector

$$r_0 + r_1 = X\mu_0 + X\mu_1 = X(\mu_0 + \mu_1) = X\hat{j} = X_j,$$

which is X_{ij}.

a) (5%) flag1: Please implement the client side code to query the servers without leaking your query.

b) (5%) flag2: To each server, since it receives a uniformly random vector $\mu_b \in_R \mathbb{Z}_2^m$ from the client, it cannot learn the query of the client. However, since you operate both servers, you can find out the query of the client.

From problem (b), we learned that colluding servers can learn the queries made by the clients. Therefore, we want a PIR protocol that preserves privacy that does not require non-colluding servers.

To achieve this, we use a homomorphic encryption (HE) scheme to encrypt the query μ to a ciphertext $\overline{\mu}$ before sending to the server. HE allows the server to compute the ciphertext $\overline{X\mu}$ corresponding to the response $X\mu$ just from the ciphertext $\overline{\mu}$.

We use the following HE scheme (See Table 1):

Here, $\tilde{\mu} := \left\lceil \frac{q}{2} \right\rceil \mu \in \left\{0, \left\lceil \frac{q}{2} \right\rceil\right\}^m$, and for $x \in \mathbb{Z}_q$, $\|x\|$ is defined to be the minimal absolute value of the integers in its equivalence class. For example, let $q = 7$. Then

$$\|5\| = \min_{k\in\mathbb{Z}} |5 + 7k| = |-2| = 2.$$

Gen(n):	Enc$_s(\mu \in \{0,1\}^m)$:	Dec$_s(A, c)$:
$s \in_R \mathbb{Z}_q^n$	$A \in_R \mathbb{Z}_q^{m \times n}$	$\mu' := c - As$
Output s	For $i \in [m]$, $e_i \leftarrow \chi$	For the i-th entry,
	$r := As + e$	output $\begin{cases} 1 & \|\mu'_i\| \geq \frac{q}{4} \\ 0 & \|\mu'_i\| < \frac{q}{4} \end{cases}$
	Output $(A, r + \tilde{\mu})$	

Table 1: Homomorphic encryption with parameters n, m, q

(You do not have to worry about what χ is since you only have to implement the server side code. But in case you are curious, χ is a *small* distribution in the sense that $e \in \mathbb{Z}_q^m$ sampled from χ^m has $\|e_i\| < \frac{q}{4m}$ for each $i \in [m]$ with high probability. We need χ to be small so that we can decrypt the ciphertexts correctly.)

If $\|e_i\|$ in each bit in the ciphertext stays below $\frac{q}{4}$, we can correctly decrypt the ciphertext. (That is why we require the initial e to have $\|e_i\| < \frac{q}{4m}$. Each entry in $X\mu$ is the sum of m elements, and the resulting error term is at most m times the original error.)

c) (5%) flag3: Please reply the queries of the client encrypted using the above HE scheme.

   Hint: Suppose $(A, c) = \texttt{Enc}_s(\mu)$, and $X \in \mathbb{Z}_2^{m \times m}$. You should find a formula to compute $(A', c') = \texttt{Enc}_s(X\mu)$ only from the ciphertext (A, c) and X.

d) (Bonus 5%) flag4: Can you break the HE scheme to learn the query made by the client?

   Hint: I heard Learning with error (LWE) can be reduced to Shortest Vector Problem (SVP), which is a difficult problem, so I don't know how to solve it.

6. Randomness Casino (15% + 5% Bonus)

Hello classmates, I just opened an online casino. Do you guys want to have some fun?

1. (5%) You are welcome to play a fun game with us! Can you earn enough money to buy the flag?
   You can access the online casino via `nc cns.csie.org 6000`, and the source code for this challenge is provided in `casino/case1.py`.

2. (10%) Someone caught you cheating! Good students won't do that. I decide to start a VIP service to help VIP players keep their numbers secret. I bet this time you will have nothing to do but be honest.
   You can access the online casino via `nc cns.csie.org 6001`, and the source code for this challenge is provided in `casino/case2.py`.
   Hint1: Do you know what MT19937 is and why we should avoid using it in a cryptographic system?

3. (Bonus 5%) Your constant cheating behaviors are unacceptable!! You are now forbidden to play the game. Maybe you should observe how normal players behave and learn to be honest!!
   Hint1: analyze `casino/case3.py` and `casino/output.txt` and try to find the flag.
   Hint2: The seeding of Python's built-in random seems different from the original MT19937.