CPT_S 327 - Homework 3

April 16, 2024

In this homework, please send one pdf file with the following information: (1) Your name, (2) your WSU ID, (3) the write up for the following two questions. More details are specified in each question.

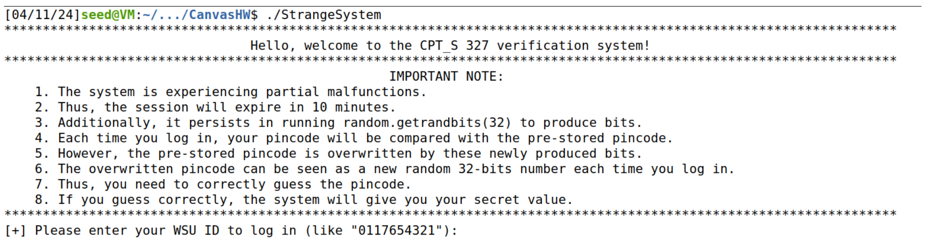
Problem 1 - Implementation Level Attacks

In the area of cryptographic implementation, it is important to notice several subtle issues. If a secure crypto system is implemented in a weak (but still correct) way, the whole system may become vulnerable. Here we demonstrate a weakness, and you are going to use the same concept to attack the code StrangeSystem.

Concept. In many scenarios, we need to implement a comparison function $\mathsf{Compare}(x, y)$, which on input two strings x, y outputs whether they are the same on all bits. For example, for the message authentication code verification, we need to compare the user-input tag τ^* with the supposed-to-be tag τ. This also extends to PIN verification, where use needs to enter some matching PIN code stored in the system.

A straight-forward implementation of $\mathsf{Compare}$ works as follow. Let n be their bit-length. Then we use a for loop. For $i = 1$ to n in the increasing order, compare the i-th bits $x(i)$ and $y(i)$. If $x(i) \neq y(i)$ in any iteration firstly observed, return 0 and terminate the function immediately. Otherwise, continue to compare the next bit.

This implementation is correct but vulnerable! In this problem, you are going to exploit the weakness to break PIN verification. Here we describe a high level attack idae, and you are going to realize that (in code) to break the following task. Consider that a system holds an unknown n-bit string $\mathsf{Pin} \in \{0,1\}^n$, and expects you to enter some matching n-bit string $\mathsf{Pin}^* \in \{0,1\}^n$. It accepts if $\mathsf{Compare}(\mathsf{Pin}, \mathsf{Pin}^*) = 1$. If you can measure the rejecting time, then you can possibly figure out the unknown Pin bit-by-bit. Here is a hint: you can figure out the first bit of Pin by entering 10^{n-1} and 00^{n-1}, i.e., a string starting with 1 followed by $n-1$ 0's and a string starting with 0 followed by $n-1$ 0's. If you



```
[04/11/24]seed@VM:~/.../CanvasHW$ ./StrangeSystem
********************************************************************************************************************
                                Hello, welcome to the CPT_S 327 verification system!
********************************************************************************************************************
                                                  IMPORTANT NOTE:
    1. The system is experiencing partial malfunctions.
    2. Thus, the session will expire in 10 minutes.
    3. Additionally, it persists in running random.getrandbits(32) to produce bits.
    4. Each time you log in, your pincode will be compared with the pre-stored pincode.
    5. However, the pre-stored pincode is overwritten by these newly produced bits.
    6. The overwritten pincode can be seen as a new random 32-bits number each time you log in.
    7. Thus, you need to correctly guess the pincode.
    8. If you guess correctly, the system will give you your secret value.
********************************************************************************************************************
[+] Please enter your WSU ID to log in (like "0117654321"):
```

Figure 1: Run "./StrangeSystem"

can observe the time difference in the rejection time, then you can determine the first bit. After this, you can determine the next bit, and so on so forth.

Task: You are given a code implementing the CPT_S 327 Verification System designed under Linux, which is designed to authenticate users through a PIN code verification process. However, the system has recently started to behave unexpectedly due to some partial malfunctions.

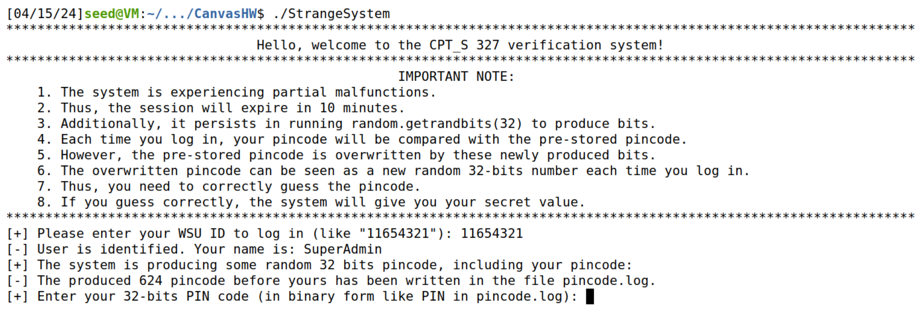
Scenario Description

The Verification System is given in the form of an executable file for Linux. You need to interact with it using "./StrangeSystem" command in your terminal under the Linux environment.

The system has following features:

1. The system is experiencing partial malfunctions.
2. Thus, the session will expire in around 10 minutes.
3. Additionally, it persists in running random.getrandbits(32) to produce bits.
4. Each time you log in, your pincode will be compared with the pre-stored pincode.
5. However, the pre-stored pincode is overwritten by these newly produced bits.
6. The overwritten pincode can be seen as a new random 32-bits number each time you log in.
7. Thus, you need to correctly guess the pincode.
8. If you guess correctly, the system will give you your secret value.

When you interact with it, it will first ask you to type in your WSU ID, then it will check whether your name is in the CPT_S 327 list. If you pass the



```
[04/15/24]seed@VM:~/.../CanvasHW$ ./StrangeSystem
*******************************************************************************************************************
                                Hello, welcome to the CPT_S 327 verification system!
*******************************************************************************************************************
                                                  IMPORTANT NOTE:
    1. The system is experiencing partial malfunctions.
    2. Thus, the session will expire in 10 minutes.
    3. Additionally, it persists in running random.getrandbits(32) to produce bits.
    4. Each time you log in, your pincode will be compared with the pre-stored pincode.
    5. However, the pre-stored pincode is overwritten by these newly produced bits.
    6. The overwritten pincode can be seen as a new random 32-bits number each time you log in.
    7. Thus, you need to correctly guess the pincode.
    8. If you guess correctly, the system will give you your secret value.
*******************************************************************************************************************
[+] Please enter your WSU ID to log in (like "11654321"): 11654321
[-] User is identified. Your name is: SuperAdmin
[+] The system is producing some random 32 bits pincode, including your pincode:
[-] The produced 624 pincode before yours has been written in the file pincode.log.
[+] Enter your 32-bits PIN code (in binary form like PIN in pincode.log): █
```

Figure 2: Log in as ”SuperAdmin”

ID checking, it will return your name to ensure that you log in as yourself, not others. Here is an example for logging in as ”SuperAdmin”.

Then you need to enter the pincode. As stated above, your pincode has been replaced by some random 32 bits string similar to 624 examples in the file ”pincode.log”. So you may need to guess what the 32 bits string is to log in.

The core code implementing the comparing process looks like:

```python
for i, digit in enumerate(pin_guess):
    if i < len(SECRET_PIN) and digit == SECRET_PIN[i]:
        time.sleep(0.3)
        print("Verifying:...")
        time.sleep(0.3)
        print("Verifying:...")
    else:
        for j in range(i, len(pin_guess)):
            time.sleep(0.1)
            print("Verifying:...")
            time.sleep(0.1)
            print("Verifying:...")
        print("[***] ERROR: Verification failed!")
        print("[-] If you guess correctly, you will get a secret value, which needs to be submitted along with your code in canvas.")
```

Submit Requirement

- Ensure your work adheres to the principles of academic integrity. The analysis, strategy, and testing should be your own work.
- You need to include in your pdf submitted file the description of your idea, analysis and operating steps. Last but not least, ”Talk is cheap. Show me the code.” You can directly attach your code with explanation in the write-up.

- What's more, after you successfully log in, you need to store your name, WSU ID and the secret value in your write-up. For example, as to "SuperAdmin", writing

  Name : SuperAdmin
  WSU ID : 11654321
  secret value : 6b3d...

  in the first line of the file satisfies the requirement.

A Useful Demo

Here is a demo code for interacting with the system. You could use it in your exploration.

```python
from pwn import *

command = '/home/seed/Desktop/share/Final HW/CanvasHW/StrangeSystem'
proc = process(command)

# Log in
proc.sendline(b"11654321")
print (proc.recvuntil("Enter your 32-bits PIN code (in binary form like PIN in pincode.log):"))

# You can use proc.interactive() to communicate with the process
proc.clean()
proc.sendline(b"0"*32)
proc.interactive()

# You can also use proc.recvline(), proc.recvall(), etc. Google and read the official tutorial for pwntools
print(proc.recvall(timeout = 1))

proc.close()  # Close the process when done
```

Some Tips

We offers you some tips to come up a good idea:

- Start by interacting with the system to familiarize yourself with its responses to various inputs. Spring into action, and you may come up with good ideas during the process.

- Pay close attention to the verification process.
- Based on your observations, formulate a hypothesis on how the system might be vulnerable to an attack.
- All Roads Lead to Rome. We recommend you to revise and use some knowledge we introduced in this course, but we accept any other approach as long as it is reasonable to pass the verification.

Problem 2 - Solving Discrete Log

In our lecture, we showed the Diffie-Hellman key agreement protocol. The idea is simple: given a cyclic group G with order q and generator g, i.e., $G = \{1, g, g^2, \ldots, g^{q-1}\}$, Alice sends g^a for some secret $a \leftarrow \mathbb{Z}_q$ and Bob sends g^b for some secret $b \leftarrow \mathbb{Z}_q$. Their final agreed key is g^{ab}, which is computable given one's secret information and the message from the other party.

The security relies on the fact that an eavesdropping adversary (Eve) who gets g^a and g^b cannot figure out their agreed key. This problem (of several variants) is known as the discrete log (DL), computational Diffie-Hellman (CDH), and decisional Diffie-Hellman (DDH).

In our lecture, we emphasized that these variants of problem need to use large order groups, i.e., q must be very large. In this task, you are going to attack the protocol when q is small. Thus, you see the necessary condition of q.

Task. Recall that in the class lecture, we use a subgroup of prime order q in $\mathbb{Z}_P^*$. That is, we find a generator g and the set $\{1, g^1, g^2, \ldots, g^{q-1} \mod P\}$ would form a group of order q.

In this task, we particularly use $q = 509$, $P = 1019$, and generator $g = 3$. You can check that $\{3^i \mod P\}_{i \in \{0,1,\ldots,q-1\}}$ really forms a cyclic group. Notice that each group exponentiation is computed under modulo P.

Now you can look up the excel file "challenges.xlsx" and find your challenges according to the WSU ID. You will find g^a and g^b for some unknown randomly chosen a, b over $\mathbb{Z}_q$. Your job is to figure out g^{ab}. In your write-up, please include your method/code, and $(WSUID, g^a, g^b, g^{ab})$.

Note: in python, the following code can calculate the $g^a \mod P$:
$g_a = pow(g, a, P)$

Problem 3 - Basic Concept of ENC and MAC

In our semester, we have emphasized couple times that private-key encryption and message authentication codes are different. In this task, you are going to show the following:

- There exists a private-key encryption whose ciphertexts do not provide data authentication.

  Particularly, you need to describe (1) a secure private-key encryption scheme $\mathsf{KeyGen}(), \mathsf{Enc}(), \mathsf{Dec}()$. (2) An adversary's strategy in the MAC game: he can make one challenge of message m, and get $\mathsf{Enc}(m)$. Then he can produce another message $m' \neq m$ with a valid encryption $\mathsf{Enc}(m')$.

- There exists a message authentication code such that its tag does not provide secrecy.

  Particularly, you need to describe (1) a secure message authentication code $\mathsf{KeyGen}(), \mathsf{Tag}(), \mathsf{Verify}()$. (2) An adversary's strategy in the semantic security game of encryption: he provides m_0, m_1 and gets $C = \mathsf{Tag}(m_b)$ where b is a uniform bit from $\{0,1\}$. From the ciphertext C, adversary is going to send another bit b' and wins if $b = b'$. Your adversary should be able to win with probability better than 0.5, significantly.

Problem 4 - Basic Concept of PRF and MAC

In our lecture, we showed how to use pseudorandom function (PRF) to construct message authentication code. The construction is simple, $\mathsf{Tag}(\mathsf{sk}, m)$ just outputs $F(\mathsf{sk}, m)$. The security intuitively follows from PRF, as an adversary cannot figure out $F(\mathsf{sk}, m^*)$ for any m^* not queried by him. Note: here "cannot figure out" implicitly requires some condition. You will notice that in the following tasks.

Below there are two task:

- We want to answer a high-level question: is this (MAC) construction secure as long as F is any secure PRF? If yes, give a proof. If No, give a counter-example.

- Let $F(\cdot,\cdot) : \{0,1\}^{128} \times \{0,1\}^{128} \to \{0,1\}^{128}$ be a secure PRF given to you. Then construct another secure PRF $F'(\cdot,\cdot) : \{0,1\}^{128} \times \{0,1\}^{128} \to \{0,1\}$. At a high level, we want you to construct a PRF that outputs only one bit, from a given PRF that outputs 128 bits.