Cryptography Final Project

April 19 2024

Angelica Loshak

Black Hat

Overview

Analyzing a program from a black hat perspective is crucial for preventing malicious external attacks. Many companies hire black hat professionals to identify vulnerabilities in their code before its public release. This is especially important for large-scale programs handling sensitive customer data like banking credentials.

Our testing process involved a line-by-line examination of every cryptographic algorithm, ensuring they produced expected outputs and met recommended security standards. We also broke parts of the user interface using common attacks such as buffer overflows and invalid user inputs. Any unexpected program behavior was exploited further. We also analyze the developer's choices and design flaws in the use of Python.

Lastly, we discovered an oversight where the GitHub repository was left exposed, allowing access to git logs and comments openly shared by the developers. The logs revealed personal data such as developers' GitHub accounts, full names, and personal email addresses, which could lead to phishing scams and identity theft beyond the scope of the Bank project. It's important to separate any personal data used during testing from production code.

Cryptographic Algorithms Analysis

We did an in-depth analysis of each cryptographic algorithm utilized by the bank. Most of these algorithms were well-implemented. We first did a run of the SHA-1 and HMAC algorithms and referenced their output with an online algorithm. The matching outputs indicated correct implementation and no vulnerability.

Additionally, we verified that all key sizes aligned with the expected standards. The RSA algorithm, utilized for digital signatures, demonstrated proper implementation of both encryption and decryption processes. Similarly, the Diffie-Hellman key exchange mechanism appeared to be working correctly.

In the AES code, we found a potential security concern, which was the reuse of the initialization vector (IV) across multiple sessions with the same key. This makes the system susceptible to a chosen-plaintext attack due to the predictability of the IV. Apart from this, we did not find any other flaws within the algorithms themselves.

Attacks

User Interface

The lack of a login page in the ATM's interface immediately caught our attention. Upon startup, the ATM directly displays the options menu, allowing unrestricted access to critical functions such as deposits, withdrawals, and balance checks without any form of authentication.

The lack of authentication is a significant security risk because it allows anyone to perform transactions on the account.

We would recommend implementing a secure authentication mechanism requiring valid credentials including a bank card and a pin before granting access to banking operations. This would ensure that only authorized individuals can utilize the ATM's features, enhancing overall security.

Even in scenarios where the ATM is under the ownership of the client or used on a trusted device like a smartphone, maintaining a logged-out state when idle remains crucial. This prevents unauthorized access in case the device is lost or stolen by malicious actors.

Input Validation

There are severe vulnerabilities in the bank server due to the absence of input validation. The server crashes when processing deposit or withdrawal requests containing non-integer input, such as strings of characters.

Input validation checks on the client side were omitted because clients can potentially create their own ATM devices and establish connections with the bank, thus bypassing client-side validation. This is why the bank server should be checking and validating against malicious inputs. This was overlooked by the developers, and there are several inputs which crash the bank server.

The absence of proper input validation opens up the server to exploitation to denial of service (DoS) attacks. An adversary can cause the bank server to crash and make it inaccessible to the clients.

Replay Attack

The bank is vulnerable to a replay attack because the messages have no method of serialization. A replay attack can be done by intercepting a message and then forwarding it to the bank. This can be done multiple times to withdraw money for example. The bank would have no way to identify the attack because the message would still be encrypted and have the correct digital signature.

The way that the developers need to fix this is by adding a sequence number to the end of every message. That way a message can only be sent once in the correct sequence order.

Random Library

The python random library is used by the program to create random bits for the keys. However, the random library is known to be vulnerable because it is predictable and uses the Mersenne Twister generator. According to the random Python docs, “Warning: The pseudo-random generators of this module should not be used for security purposes.” (https://docs.python.org/3/library/random.html). The developers need to find a more secure method of generating random bits or else they will be susceptible to Python Random Generator Number (PRGN) exploits.

Python3.11

For cryptographic purposes, Python is not the most recommended choice. The project didn't specify the necessity of Python version 3.11, causing compatibility issues when using Python 3.10 due to type discrepancies. Python's nature as an interpreted language means that type errors are often detected at runtime, not during compilation. Somehow, Python 3.11 manages to execute despite these type errors, however using Python 3.10 will break the code.

Exposed GitHub

The git metadata was not deleted before submission and was available to investigation. We can see the commit logs that each developer made, along with timestamps that they were working on the project. We also saw personal information exposed, such as the full names, personal email addresses, and github repos. It is important for developers to protect their privacy when working on projects which will be exposed to the public.