Theron Wolcott
Group 57 (Solo)
CMSC414
Marsh

Overall Protocol

I did not change the underlying network/sockets protocols that were built into the project initially. The ATM still sends messages to and receives messages from the server bank using that system. However, I have tried to make it as secure as possible.

For each message, I concatenate a string similar to those used on the command line: “withdraw <amount> <user>.” In addition, every message from the ATM to the bank includes a sequence number: “withdraw <amount> <user> <sequence>.” Both the bank and the client maintain and increase the sequence number, so it's impossible for any message string to be the same as any other. They are all completely unique. The sequence number becomes important when the message string is encrypted because even two messages for identical transactions encrypted with an identical key are not actually identical.

For encryption, I’m using OpenSSL AES encryption. The init program generates a unique key that is written to both the .card and .atm files. On initialization, the bank and ATM programs read from their files and store the shared key. Then, each message from the ATM to the bank is encrypted using the current key, along with a one-time use IV prepended to the encrypted message. The bank decrypts the message using the prepended IV and the shared key from the .bank file and parses the string into tokens and does the appropriate action with them (if the message is valid). The responses from the bank are not encrypted because they don’t really contain any important content.

Vulnerabilities

Vulnerability 1: Someone with access to our router can simply change the amount that is being withdrawn.
Solution: To prevent somebody messing with withdrawal amounts, I can use AES encryption for integrity and confidentiality. Each message is concatenated as a string and the whole string is sent from the ATM to the bank is encrypted with a symmetric AES key. Meaning, even if an attacker does intercept the message, they can’t modify the withdrawal amount.

Vulnerability 2: Someone with access to the router can change the account that a withdrawal is coming from.
Solution: Our AES encryption prevents attackers from changing the account that a withdrawal is coming from. The reason I am encrypting each message as a whole is so that attackers are

unable to do what I did on our cryptography project and reassemble pieces of different messages. By encrypting each message as a whole, including the sequence number (which I will talk about later), nobody will be able to take a message and replace just the account associated with the withdrawal with a different one.

Vulnerability 3: Someone can convert a balance inquiry into a withdrawal.
Solution: Our AES encryption will prevent an attacker from figuring out the type of transaction that is occurring.

Vulnerability 4: Someone can replay a prior transaction in its entirety. For example, a withdrawal happens over and over again instead of just once.
Solution: To defend against replay attacks, I added a global sequence number to each transaction. This number goes up for every transaction and never repeats, which ensures that nobody can ever repeat a transaction. If I notice an old sequence number whose time has passed, I will reject that transaction.

Vulnerability 5: Someone without the correct pin tries to impersonate a user.
Solution: To prevent someone from trying to impersonate another user, I encrypted each PIN with AES encryption. When a user enters their pin in the ATM, it is encrypted and sent to the bank. The bank decrypts the message using the shared key, verifies the PIN, and sends its response encrypted with the same key. This makes it so that an attacker cannot easily intercept a person's PIN during communication between the bank and ATM.

Note: as of 12/2 my implementation doesn't fully have encryption working, I hoped to have it tonight but I'm going to have to resubmit with it working.

Update: Encryption fully working as of 12/3.