Brandon Daniel & Khanh Nguyen

CMSC414

Overall Protocol

In our system, our protocol involves using our own encryption/decryption methods utilizing private keys that are stored within the code of the files atm.c and bank.c. In order to use our atm system, the user must compile all of the files using the make command. The make command will make init, which calls init.py, creating a .bank file and a .atm file. Upon using the bank to create a user, a card is generated. A file <username>.card is created. The only information on this card is a pin that has been encrypted by performing bit-shifts. Since both bank and atm have take in user input, our protocol sanitizes and handles unwanted input through regular expressions. When atm is being accessed, messages are sent between atm and bank. Our protocol encrypts these messages using 2 private keys for sending and receiving operations. Each character of the messages sent between the system are xor'd with our private key to ensure confidentiality in our sent messages, along with this.

Security attacks considered and implemented solutions

1. One of the inherent security attacks that can be performed on a system that takes in input from users is a buffer overflow. Our solution to this issue is to make use of functions such as strncopy, fgets. With these functions, we can limit the amount of characters copied when we are using the input provided by the user. By using strncopy, we only copy the exact amount of characters that are needed. We also explicitly insert a null byte at the end of each character array and safely copy to ensure the amount of characters there are in any given character array are exactly the number of characters that are needed.
2. Another security attack would be the attacker examining the atm card files and discovering the confidential pin. Since we use cards to track users and permit access to the atm, this can be an issue. Pins cannot be in plaintext on the cards but there also has to be a way for our system to use the pin on the cards to authorize the user's sessions. In order to protect the confidentiality of our cards, we encrypt the pins so that even when the attacker has a victim's card, upon examination, they would not be able to determine the real pin. We implemented this by performing our own encryption on the pins before loading them onto the card. We preform an xor operation on the card in order to give a false value. While the attacker cannot obtain the real pin information on the card, our system can decrypt the cards and verify entered pins.
3. Another security attack would be the spoofing of cards, with access to cards, attackers are also able to create fake cards with pins to imitate a user using their real card to access the atm. By encrypting each card a secret xor value, attackers are unable to just create a fake cards with a pin since they do not know our how we are shifting.
4. Another valid security attack would be invalid inputs to crash our atm or bank. While attacks such as these do not violate the privacy of our users, being able to crash our system can still be considered an attack. To prevent invalid inputs with the intent to crash our system, we utilize regular expressions to handle our inputs. With the use of regex, we are able to sanitize every input that is sent to our systems and prevent inputs such as "withdraw 0/apple" or "balance %s%s%d" that could very easily crash our systems due to the way they are implemented.

5. Since our attackers can modify packets sent through our router from or to the bank and atm system, one of the security attacks that they could perform are sending intercepting messages sent as well as sending messages from the routers to the atm or bank themselves. Since these messages sent between systems are the commands for actions, having these messages exposed can be a huge security threat. To deal with this threat, in both of our systems, we have implemented an encryption on the plaintext commands. Each plaintext command goes through an xor encryption using our private key and when the messages are received they get decrypted. Bank and ATM both have a different key for encrypting and decrypting. As a result, the attacker is unable to interpret messages intercepted and messages sent from the attacker will not be read due to malformed input as a result of bad decryption.