ATM Build-It Write Up

Authors: Hrithik Bansal, Gilbert Garczynski, William Shearer, Sebastian Tramontana

Project name: ATM Build-It

Goal: To implement a bank/ATM interface with the computer and network security lessons learned in CMSC414, taught by Michael Marsh, Fall 2020.

The protocol written by the authors consist of 4 programs;

1. Bank
   - Command line interface that holds user accounts
2. ATM
   - Command line interface that a user can utilize to access their bank accounts
3. Router
   - How the bank and atm communicate with each other.
4. Init
   - Establishes a shared key book between the bank and atm

Protocols, Communications and Initializations

1. Card Protocol
   - The contents of the cards are encrypted using AES256. Each card contains only one line of plaintext. This plaintext starts with the username of the user of that card followed by a space and then the 4-number pin of the user. Below is the format of the protocol.

     [username][space][user_pin]

2. Init Protocol
   - The init program creates the two files <init-fname>.bank and <init-fname>.atm. These files are identical. They contain 50,000 lines of randomly generated key-IV

pairs for AES256 encryption and decryption. Each line will start with the 32-number key followed by a space and then the 16-number initialization vector as seen below:

[32-byte key][space][16-byte IV][newline]

3. Message protocol
   - Messages going from the ATM to the Bank will start with a 4-digit ATM ID, which will be followed by a char that indicates the service request of the ATM. That will then be followed by the name of the user that the service is requested for, along with a money amount if it is required for the service. Messages going from the Bank to the ATM will start with a 4-digit Bank ID, which will either be followed by "Authorized" if the requested service was performed, or "Unauthorized" if it wasn't. If the service was a balance request from the ATM, the specified user's balance will be returned. Below are the formats of the protocols. Yellow highlighted text indicates that a message may or may not have this text as it is dependent on the service being requested and/or returned. All messages sent are encrypted using AES256.

     [atm_id][service_request_char][username][money_amount]

     [bank_id][authorized_unauthorized][user_balance]

Protocols, ATM

1. Begin session Protocol
   - The user inputs begin-session ‘username’. The program checks to see if there is a user currently logged in and if there is a user logged in, it prints "A user is already

logged in". If no user is logged in, the program creates a message to send to the bank. If the response from the bank contains "Unauthorized", "No such user" is printed. If the response from the bank contains "Authorized", then the program takes the username and finds the card name associated with that username. If there is no such card for the user, then "Unable to access %s's card" is printed. Otherwise, the data on the card is then decrypted and checks that the card has not been tampered with or lost. If the card corresponding to the given username is not found, then "Not authorized" is printed. Otherwise, the program continues and prompts the user for a PIN. If the input equals the pin associated with the username and card, the program prints "Authorized" and sets atm-username to the username that was inputted. Otherwise, the program prints "Not Authorized".

2. Withdraw Protocol
    - The user inputs withdraw 'amount'. The program checks to see if there is a user currently logged in and if there is no user logged in, it prints "No user is logged in". If there is a user logged in, the program will prepare a message with [atm_id][w][username] and send it to the bank. The bank will then process the request, determine if the amount is available (it is less than or equal to the balance), and send a message back with "Authorized" or "Unauthorized" in the response. The program then parses the returned message and if it contains "Unauthorized", the program prints "Insufficient funds". If the response contains "Authorized", it prints the amount dispensed in the format"$%amt dispensed".
3. Balance Protocol

- The user inputs balance. The program checks to see if there is a user currently logged in and if there is no user logged in, it prints “No user is logged in”. If there is a user logged in, the program will prepare a message with [atm_id][b][username] and send it to the bank. The bank will then process the request, determine the balance, and send a message back with the balance. The program then parses the returned message and prints it in the format “$balance”.

4. End-session Protocol
   - The user inputs end-session. The code first checks to make sure that there is actually someone logged in. If no one is logged in, then the program prints “No user is logged in”. If there is a user logged in, then the ATM sets the user logged in to NULL, attempts to 0, and prints “User logged out”.

Protocols, Bank

1. Create-user protocol
   - The user inputs create-user <user-name> <pin> <balance>. The program checks to make sure that the user-name does not appear in the list of bank users. If it does, "Error: user user-name already exists" is printed. If it is a unique user-name, the user is created in a table. Then, the card is created for the user. If there are any errors during this creation, "Error creating card file for user username" is printed. The data is then encrypted and the card for the user user-name has been created.
2. Deposit
   - The user inputs deposit <user-name> <amt>. If there is no such user, then print "No such user”. Otherwise, find the balance of the user and add amt to this

amount, printing "$amt added to user-name's account". If the new balance is greater than that of the max integer value, then print "Too rich for this program".

3. Balance
    - The user inputs balance <user-name>. The program checks to make sure that the user-name appears in the list of bank users. If the flag for an atm request is true, respond with the appropritare balance for the given user-name. If not and the user exists, print "$balance". If the user does not exist, print"No such user".

Attacks and Countermeasures

1. Brute-force PIN
    - Given the fact that there the PIN for each user is a 4 digit number with each digit having a range of [0,9], one could theoretically brute force the PIN. There are 10000 (10 pow 4) possible combinations. While this is a time consuming action for a human, a program can be written relatively easily to run through all values. Without a lockout time period or PIN reset, this is a simple yet plausible way for an attacker to break in.
    - To combat this, in the atm.h file there is an integer data value counting the number of failed attempts. Additional implementation is used in the atm.c file in the function attempts(ATM *atm). Upon a PIN entered, the value is incremented and if it becomes 3, then there is a lockout of 2 minutes enforced.
2. Intercepting of messages
    - As messages and personal bank info are being sent across the network, it is necessary to encrypt the data as an attacker could potentially sniff and/or intercept the traffic going between the bank and the atm. They could then view the format

of the message and spoof a message to make an unauthorized transaction. They could additionally intercept PIN's, usernames, and other personal information.

- To combat this, we encrypted all messages from the atm to the bank and from the bank to atm.

3. Protection against malicious command line inputs
   - There are a plethora of ways that an attacker could provide malicious input such as a buffer overflow attack. An attacker could craft an input to display the buffers that hold relevant information used by the atm and bank. They could then use information they find here to either decrypt encrypted messages or see usernames and PINs.
   - To defend against this, we implemented a max length for usernames and used strncpy() instead of strcpy(). An attacker could also provide more arguments than are required, thus causing the program to crash. To combat this, we used regex to make sure the commands and their accompanying inputs were valid. We also implemented a counter for the arguments when parsing the inputs. If the counter was either greater than the max arguments or greater than the max arguments for a given command, "Invalid command." is displayed.
4. Spoof Messages Part 1
   - Theoretically, an attacker could perform a brute force, dictionary lookup, or know someone on the inside that could provide them with the key used to encrypt the messages to and from the bank and atm.

- To prevent this, a new key is generated every time a message is encrypted to prevent a passive wiretapper from getting the key used today and using it next week.

5. Spoofed Messages Part 2
    - An attacker could spoof messages for communications between the bank and the atm.
    - To prevent the inclusion of spoofed messages for communications between the bank and the atm, unique IDs are provided in the packets sent between the bank and atm. This is so the bank knows the message came from the atm and the atm knows that the message came from the bank. If either the bank or the atm gets a message without the corresponding ID, it ignores the message.
6. Stealing pins from cards/tampering with or creating new cards
    - These attacks have been grouped into one attack because they have the same counter and the same goal of manipulating pin numbers via the `.card` files to illegally gain access. The first of these attacks would involve a malicious user who has access to a card that is not theirs to open the `.card` file and read the pin number inside and gain access. The second of these attacks would involve a malicious user who either modifies an existing card to have a pin of their choosing or to create an entirely new card for an existing user with a pin of their choosing. Both of these would result in a malicious user knowing the pin of an existing user and using it to gain access to their account.
    - To counter these attacks, we use encryption. Specifically we encrypt the contents of the `.card` files with AES256 using a 32 byte key and 16 byte IV. The contents

of the `.card` files are `<user-name> <pin>` and they are encrypted for the purpose of making reading the cards only possible by the `atm` and `bank` programs. This would not be sufficient if we were to only place the pin of the user in the card. This is because pins can be repeated by users so a malicious user could find out what a user's pin is by creating 10,000 cards with all 10,000 (10 pow 4) possible pins. After doing this a malicious user would be able to find the pin of a user by reading their card and comparing the ciphertext to that of their 10,000 cards. To prevent this we encrypt the username and pin of a user to create a unique card and ciphertext everytime. Furthermore, the username from the card is verified by the atm against the username of the user that is currently signed in to add a check against tampering.

Other Potential Attacks

1. Keylogger
    - Through the use of a keylogger program on the ATM, an attacker may be able to gain PIN's and usernames as they are inputted.
2. DoS attack
    - An attacker could send a plethora of request messages to the bank. Even if the bank deems these messages to be illegitimate, it would still have to check all of these messages. The bank would then become bogged down from these false requests and legitimate requests will be slowed or lost in the network.
3. Hack into the bank database and see plaintext usernames and PINS

- While the documentation says that the database in the bank cannot be viewed, if a bank were to use SQL, a possible SQL injection could take place and thus display the entire username and PIN tables, along with any other information present on the database. A way to prevent this is to use prudent security practices and utilize through testing of the database.