Overall Protocol
Authors: Andrew McGillicuddy, Marco Roxas (Poch), Paul Taylor, Michael Zheng

INIT:
The init function creates the .atm and .bank files. If there already exists a file or there is an error creating the initialization files, the program would tell the user. Otherwise, it would generate a 32-byte key and will be written to the .atm and .bank files.

CARD:
.card->
When the bank creates a new user, it makes a .card file. The .card file stores a buffer of 128 bytes. We used system call getrandom fills the buffer with random bytes. The .card file is used to vertify a user.

BANK & ATM:
The ATM and Bank form a networked client-server system. The ATM (client) sends requests to the Bank (server) which carries out the requests and replies with a response. The ATM and Bank communicate using single packets for a request or response. Only exact size data packets accepted.

PACKETS:
5 request types for a packet:
  VERIFY_USERNAME,
  VERIFY_CARD,
  VERIFY_PIN,
  WITHDRAW,
  BALANCE_REQUEST,

1 response type:
  RESPONSE

The same packet format is used for both requests and responses.

There are two layers to the communication protocol: the message packet carrying the request (inner packet) is encapsulated in another packet (outer packet). The outer packet is then encrypted.

Below display the format of the packet:
Format of the Inner Packet:
  Packet Type
  Username
  Status of the Packet (intPayload)
  Section for the pin number (charPayload)

Format of the OuterPacket:
  Header
  Checksum
  Inner Packet
  Footer

ENCRYPTION:
Encryption is symmetric-key performed using an implementation of the AES algorithm with a 256-bit key and 128-bit block size and IV (initial vector), operating in CBC block mode. It is a custom implementation involving the standard aes-256.

Both the key and IV are randomly generated from a cryptographically secure random source.

During startup the ATM and Bank agree on an initial IV by exchanging randomly-generated IVs. It’s a separate protocol used only during startup, with the agreed-upon IV crossing the network in two separately encrypted blocks.

The file formats can be either a raw key or card byte buffer. The data is located at the inner packet, while the integrity data can be found in the outer packet. The files would be encrypted with aes-256 in CBC mode. Only the exact size of the data packet is accepted.

Attacks:
A-01 Unecrypted Data

Description:
When transfering data over the network, an attacker can see the plaintext of the pin and the username. For example, an attacker can open Wireshark to begin sniffing the packets. Since this project is sending data locally, an attacker can use the IP address 127.0.0.1 to sniff the packets. Let say Alice enter "begin-session Alice", you can see the request send over to the server. In the payload, we can see the word "Alice" in plaintext. Then the server sends back a response for the pin. When Alice enters her pin, we can see the pin in plaintext. Now the attacker who is sniffing around knows Alice's pin.

Countermeasure:

In order to encrypt the message sending between the atm, router, and bank, we created the directory called encrypt. THe encrypt directory contains two files that perform the encryption process. The aes-256.c uses s-box to perform the encryption. With encrypted data being sent across the network, an attacker would not be able to see sensitive information.

A-02 Bruteforce on PIN with card

Description:
If an attacker has a user's card, the attacker can brute force the pin. There is no mechanism to prevent the attacker from trying out all the possible pin options. For example, an attacker can create a while loop that repeatedly enters "begin-session Alice" and enter a starting pin at 0000. The attacker can check if the starting pin is correct with an if statement. If the starting pin doesn't match the actual pin, the ATM would print "Not authorized." The while loop would increment the starting pin until the ATM prints out "Authorized."

Countermeasure:
In order to prevent the attacker from bruteforcing the pin, we added the pintime and pinattempt to the the User struct in formats.h. A user has 5 attempts to correctly enter their pin number, otherwise, the user would be locked out for 15 minutes. The implementation is in the file database.c at the db_vertification function.

A-03 Buffer Overflow Attack

Description:
Whenever our program reads a command from stdin or from a file, it is vulnerable to a buffer overflow attack. An attacker can attempt to write data beyond the fixed sized of the buffer. In addition, the attacker could add a malicious shellcode which allows the attacker to execute pieces of code.

Countermeasure:
We used better string functions such "strncpy()" and "memcpy()" over the "strcpy()" functions. String functions such as "strcpy()" relies on a trailing "\0" to end a string. This is vulnerable to buffer overflow attacks. We made sure to switch to safer string functions. For the ATM, we made switches on the remote_vertification(), remote_withdraw(), and remote_balance(). Furthermore, we made switches on the database.c files, aes-256.c, and encrpyt.c.We also used fgets() to read stdin. fgets() always terminates the string with a NULL value on a successful read. This would help prevent attackers performing buffer

overflow attack while our program deciphers the ciphertext.

A-04 Replay Attacks

Description:
When transmitting data between the bank and the ATM, an attacker can see the packet moves. With Wireshark, an attacker can see the content of the packet and could perform a replay attack.

Countermeasure:
In order to prevent the attacker from conducting replay attacks, we added CBC mode ciphering to the connections in the file encrypt.c when receiving and sending packets. When the ATM sends a packet it encrypts the message with the send_packet() in the encrypt.c file. Then the bank would receive the message and decrypt it in the recieve_packet() in encrypt.c file. The recieve_packet() would return 0 if there are any modifications to the packet. The modified packet would return a 0 and become a failure when it reaches to the bank. Furthermore, the IV changes based on random ciphertext, so any repeat packets decrypt to garbage.

A-05 Chosen Plaintext Attack

Description:
An attacker can create a script where it encrypts random plaintexts in order to find the corresponding ciphertexts. Once an attacker finds the corresponding ciphertexts, the user's pin nubmer could be compromised.

Countermeasure:
To prevent a chosen plaintext attack in compromising the user's pin, we added freshness checking to the IV initialization in our set_iv() function and setup_connection() in the encrypt.c file. The IV initialization process only accepts fresh IVs from the bank, which generates new IVs randomly. Then, the random freshness value is properly encrypted with a new IV. Furthermore, the attacker needs to watch ciphertext from a valid user session. Due to the size of the card, it is too large for an attacker to guess the correct plaintext. Each pin guess is encrypted by the rebooting atm. This would prevent the attacker to find any leaks about the key and any chosen plaintext attacks.

A-06 Dummy Card

Description:
If an attacker knows the pin number, the attacker could used a dummy card to gain access.

Countermeasure:
When a user tries to gain access, the ATM prompts the user for their pin number. Before the user could enter their pin number, our program compares the content of the .card file with memcmp() in our db_verification() function. The db_vertifcation() functions compared the .card file and the one we stored. If the contents don't match the bank would return a REQUEST_FAILED to the client. This would prevent attackers from using a dummy card.

A-07 IV Reset Attack

Description:
An attacker can supply an old ciphertext to either the bank or the atm. Then the CBC mode would set the attackers ciphertext as the IV. Now the bank or the atm is using the old IV, which can lead to replay attacks.

Countermeasure:
To prevent possible relay attacks and resetting the IV, our program only updates the IV from full and valid packets. The old ciphertext under the wrong IV would

be an invalid decryption.

A-08 Packet Modification
While an attacker is observing the packets send between in the network, the attacker could tweak packets in flight to create different results. This could change the commands being sent to the bank. For example, instead of withdrawing $10 from Alice's account, it changes to $100.

Countermeasure:
An attacker can modify the packets being sent through the router. However, in our encrypt.c and aes-256.c, the files produces unpredictable plaintext from any ciphertext change. The packet checksum is across the entire outerpacket. If an attacker modifies any of the packets, the checksum would be invalid. Any invalid packets would get rejected.

A-09 Command Injections
Description:
An attacker could add some invalid inputs to the atm prompt. This could exploit our program.

Countermeasure:
The commands.c file validates any input from the bank and atm. We used regular expression patterns to explicitly define sets of characters that are allowed. There is a function called did_amount_overflow to ensure the input doesn't go past the maximum length. Furthermore, the commands.c file ensures if the command line is null terminated. This would prevent an attacker from entering any malicious inputs.

Current Design Issues:
DI-01
Description:
In our program, the bank does not perform any form of session tracking. A machine that can form properly encrypted packets can trigger withdraws from any users. We need a form of session tracking to provide a summary of every transaction. The summary would include the time and action of the transaction. This would help reveal any presence of an attack.

DI-02
Description:
The connection drops if there is any network issue or foreign data sent. The connection drop causes ATM to be restarted to reconnect. This is an example of denial of service.

DI-03
Description:
Our bank and atm are vulnerable to timing attacks and maybe some possible side-channel attacks. It is too complicated to implement these attacks.

DI-04
Description:
Another issue is that the encryption uses CBC mode, which is not secure against active attackers. This would require extensive work, to keep attackers from setting the IV. The issue was so dangerous, it was removed from later versions of TLS.