SY308: Security Fundamental Principles

Calendar	Policy

Project: ATM Design and Implementation

Overview

In this project, you will design and implement a prototype ATM/Bank system. In the process, you will get a chance to demonstrate all the good security practices that you have learned in this class. To judge how effectively you have achieved this goal, in the next phase of the project, you will get a chance to (try to) attack another team's design!

In the first phase of the project, you will not worry about protecting the communications between the bank and server (although you should have at least an idea about how you could do that by now, using the cryptography we have learned so far). Instead, your goal will be to make sure that your system adheres as closely as you can to the fundamental security principles we have learned and that it has no vulnerabilities in the software or logic. For instance, if you could withdraw more money than a user has in their account that would be bad!

1. You may work in teams of at most three people. Email your instructor indicating who is in your team, and include all names on your write-ups.
2. You will design and implement two programs: an ATM and a bank.
3. You will be provided with starter files for the ATM, the bank, and a router that will route messages between the ATM and the bank. The provided code will allow the ATM and the router to communicate with each other, and the router and the bank to communicate with each other. For Part 1 of the project, the router will be configured to simply pass messages back-and-forth between the ATM and the bank, you will not need to modify it in any way. Looking ahead, **the router will provide a way to carry out passive or active attacks on the "communication channel" between the bank and the ATM.**
4. You will design a protocol allowing a user to withdraw money from the ATM. Requirements include:
    - The ATM card of user XXX will be represented by a file called XXX.card. Just as in real life, this card should have all the information necessary for the bank to locate the user's account (for instance you don't have a username when you login to an ATM, it just knows who you are based on the card). Think something like an account number.
    - The user must input a 4-digit PIN.
    - User balances will be maintained by the bank, not by the ATM.
    - You need not support multiple ATMs connecting to the bank simultaneously.
5. You will then implement your protocol. Most of your work should involve the ATM and bank programs, with no modifications to the router.

(100 pts) Part I - Basic Functionality

Your programs should support the following functionality. (In Part 2 later, you will add communication security aspects to your programs.)

Starter Files

1. Download project.zip. You can use the "unzip" command line tool to extract zip archives.

Program Description

- The bank should be set up with three user accounts, for Alice, Bob, and Carol. Alice's and Bob's balance should be initialized to $100, and Carol's balance should be initialized to $0. In addition to the ATM, bank, and router programs, you will specify the files Alice.card, Bob.card, and Carol.card, and PINs defined for these users.
- Your programs should be run the three programs as follows in the same machine:
    1. Run bank.

        ```
        python3 bank.py
        ```

2. Run router.

   ```
   python3 router.py
   ```

3. Finally run atm.

   ```
   python3 atm.py
   ```

- The code that we provide contains the networking operations you will need. The router, bank, and ATM all run on different ports on localhost (so **you need not even be connected to the Internet for this to work**).

Bank commands

The bank should support the following commands:

- Command **deposit**

  ```
  deposit user amt
  ```

  The above command will add amt to the account of user. After successful completion of this command, this should print amt added to user's account to standard output (print).
- Command **balance**

  ```
  balance user
  ```

  The above command should print the current balance of user to standard output (print). Withdrawals from the bank are not supported.

Sample Executions

Here is an example transcript, with the bank balances initialized as stated above:

```
BANK: balance Alice
$100

BANK: balance Carol
$0

BANK: deposit Carol 2
$2 added to Carol's account

BANK: balance Carol
$2

BANK: ..
```

Three Important Functions in bank.py

There are three important functions in bank.py.

- sendBytes: When you call this function, passing a bytes object, it will send those bytes to the ATM via the router. On the ATM side, this will prompt the handleLocal function to be called where it will receive the bytes sent by the bank.
- handleLocal: This function handles console inputs. Every time a user enters something on the console this function gets called. Currently, it simply takes the input from the console and send it to the router. **Modify this function for your code to work as in the sample execution above.**
- handleRemote: This functions handles messages (in bytes) coming from the ATM (via the router). Currently, it simply prints out the incoming message and sends the same message back to the ATM. **Modify this function so that the bank can handle the ATM messages correctly.**

ATM commands

The ATM should support the following commands:

- Command **begin-session**

  ```
  begin-session
  ```

  The above command represents a **user walking up to the ATM and inserting his or her ATM card**. ATM should then read from **Inserted.card**, and prompt for a PIN. Of course, your ATM should find out the identify of the current user by reading the details stored in the file **Inserted.card**.

  > For example, if you want to log in as Alice:
  > 1. (Simulationg Alice inserting her card) Copy her file **Alice.card** into **Inserted.card**. The ATM will read Inserted.card and know that Alice is trying to log in.
  > 2. (Simulting Alice entering PIN) ATM will give a prompt asking for a PIN, the correct PIN must be entered.

  If the correct PIN is entered, print "Authorized" and then allow the user to execute the following three commands. Otherwise, print "Unauthorized" and continue listening for further begin-session commands.
- Command **withdraw**

  ```
  withdraw amt
  ```

  The above command should print "$amt dispensed" if the currently logged-in user has sufficient funds in their account. Their account should be debited accordingly. Otherwise, print "insufficient funds". (If no user is currently logged-in, print "no user logged in".)
- Command **balance**

  ```
  balance
  ```

  This command should print the current balance of the currently logged-in user. (If no user is currently logged-in, print "no user logged in".)
- Command **end-session**

  ```
  end-session
  ```

  This command terminates the current session and prints "user logged out". (If no user is currently logged-in, print "no user logged in".) The ATM should then continue listening for further begin-session commands. The ATM should support an unlimited number of withdraw and balance commands per session. Deposits at an ATM are not supported.

Sample Executions

Here is an example transcript assuming the following scenario:

- Alice's balance is $100
- The file Alice.card is present. (The ATM will read this file when a begin-session command is executed.)
- Alice's PIN is 0000.

```
ATM: balance
no user logged in

(... You copy Alice.card to Inserted.card...)

ATM: begin-session
PIN? 0000
authorized

ATM (Alice): balance
$100

ATM (Alice): withdraw 1
$1 dispensed

ATM (Alice): balance
```

```
$99

ATM (Alice): end-session
user logged out

ATM: ...
```

Note: In the above the prompts change as a user logs in:

Three Important Functions in atm.py

The ATM has analogous functions to the Bank. As with bank.py, modify handleLocal and handleRemote.

Note: Security Mindset

Your system should be robust against bad inputs! The other team members will attack your system. In particular, **your program should not crash even if a user gives bad inputs!** There are a few things to think about in support of this idea:

- When using the split() function in python, **always** check the length of the list that gets returned. You cannot assume that the user gave you the right number of arguments. For instance, they could have typed "withdraw" without any dollar amount. Your program should be able to handle this safely. You could even get a situation where they typed nothing at all and there is no command to parse.
- If you use the int() function to cast a string to an integer it will raise an exception if that string does not represent an integer. For instance, "withdraw duck". You can (and should) handle this by surrounding the call to the int() function with a try-except block.
- An illegal command should not crash your program, it should simply tell the user that the command cannot be parsed and ask for them to try again. This is how interactive programs always work. Consider the python interpreter or the terminal itself, if you type something wrong it does not shut down it tells you what was wrong and lets you reinput your command.

Submission

1. Prepare the following files.
    - bank.py, atm.py: Your python files.
    - Alice.card, Bob.card, and Carol.card: the card files.
    - design.doc: the design document
        - It should describe your system. It should provide **sufficient details to understand the mechanisms you put into place, without having to look at the source code.** In particular, it should indicate what messages are sent between the ATM/Bank during login, withdrawal, etc.
        - It should **give the PIN for each user.**
        - You should **identify two components of your design or implementation** that adhere to a fundamental security principle. For instance, you could say (if I had not already given it to you) that the system uses separation of privileges because it requires both an ATM card and a PIN to login.
    - log-bank.txt, log-atm.txt: log files proving your system works correctly.
2. Submit your files. Go to hw submission server, and submit the following files by using the menu in the server:

    ```
    bank.py, atm.py, Alice.card, Bob.card, Carol.card, design.doc, log-bank.txt, log-atm.txt
    ```

    Use the following project name for submission:

    ```
    project01
    ```

3. Deadline: It's due **2359 on 2/20** (Thursday).