Team JAC Project 4: Build It Design Document

Group Members: Josue Proano, Aayushi Roy, Casey Harris

Note: A graph detailing our protocol can be found on page 10 and at this link.

Bank Protocol

The primary goal of the bank is to create an authentication file, initialize its port and IP, and carry out any valid request sent by the ATM. The CBC decryption takes place in the bank. The bank keeps track of current accounts and balances via a linked list. Cards are cleared in the ATM before the request reaches the bank except in the case of an account creation request. The bank also contains a struct that keeps track of its authfile, key, IV, and users.

1. Bank Create

Creates an auth file upon bank initialization. If the auth file already exists, the bank will exit. It also sets the port and IP and sets defaults if they aren't provided.

2. Create Account

The bank receives a create account request from the ATM. It will proceed to check if the given user already exists in the linked list. If they do exist, no account will be created. Otherwise, an account consisting of the account name and the balance will be added to the linked list. The bank will send a message to the ATM indicating whether or not the account was made, and the ATM will handle the card creation from there.

3. Deposit Amount

The bank receives a request to deposit money from the atm. It will check if the user exists and if they do, the amount will be added to the balance in their account.

4. Withdraw Amount

Operates in a similar fashion to deposit above, but the bank will make an additional check to ensure that the balance will not go below 0 if this transaction is completed. If it will, the transaction is denied. Otherwise, the okay is sent back to the atm.

5. Get Balance

The bank receives a request for the user's balance from the atm. If the user's account exists, the bank prints out the user's balance and sends the okay back to the atm.

ATM Protocol

The primary goal of the ATM is to first parse the input from the command line, then use that parsed input to create requests. For every command except "create new account request", the card file specified by the user is first decrypted. The card file's contents are compared against the account name of the user and if they do not match, the transaction will not move forward. The requests are encrypted with CBC, using a key in the auth file. The ATM also creates a new IV for every new command. The encrypted request and IV are then sent to the bank. The ATM also contains a struct that keeps track of its key and current IV.

1. ATM Create

The ATM saves the authfile that the bank creates in its struct. The ATM creates a key and cipher and sends them to the bank so that it can be ensured that the bank and ATM are using the same auth file. The ATM then makes a new IV and key and sends them to the bank.

2. Create New Account Request

The user interacts with the ATM to indicate that they wish to create a new account (using the -n mode). The ATM sends this request to the bank, and if it gets an okay from the bank (the account doesn't already exist), it will create a card (number 2 below). It will then proceed to tell the user that the account has been created.

3. Create Card

Upon receiving the message from the bank that the new account has been created for the user, the ATM will create a card for that user. The card file contains the encrypted account name of the user. The account name is encrypted using the auth file. The user can provide a desired card name, but if they do not provide one, a default card (account_name.card) will be issued.

4. Withdraw Request

The user indicates an amount that they wish to withdraw (using -w). This command is parsed, encrypted, and sent to the bank. If the bank gives the okay, the ATM will notify the user that they were able to withdraw the amount they indicated.

5. Deposit Request

The user indicates an amount that they wish to deposit (using -d). This command is parsed, encrypted, and sent to the bank. If the bank gives the okay, the ATM will notify the user that they were able to deposit the amount they indicated.

6. Get Balance Request

The user indicates that they would like to get their account balance (using -g). The ATM will send the encrypted request to the bank. It will wait for the okay from the bank, then proceed to print out the user's account balance.

LinkedList Protocol

A basic linked list implementation that allows for users to be added with their account name tied to their balance. Searching for users is also possible and will return their balance upon success.

Parser Protocol

The Parser's role is to use getopt to read in options and arguments from stdin. It will match the provided options to each possible option in a switch statement to determine what further actions to take. Based on the case matched to by the parser, a regex from RegexCheck will be used to confirm the validity of the user's input. If the commands are valid, the arguments are stored in a struct that the parser holds. The struct contains mode, account name, authfile, IP, port, card, and balance. The Parser also handles cases of duplicate parameters and multiple modes.

RegexCheck Protocol

The RegexCheck compiles regexes for valid file names, numbers, and amounts (money) according to project specs. Each individual mode has different criteria in addition to the appropriate regex being applied.

1. Check_Case_A (Account)

Checks the argument that follows the option for a new account against the acceptable file regex and also checks that the length of the account name is between 1 and 122.

2. Check_Case_S (Auth File)

Checks the argument that follows the option for an auth file against the acceptable file name regex and also ensures the characters '.' and '..' are not used.

3. Check_Case_I (IP)

   Checks the argument that follows the option for an IP and checks it against a special regex for valid IP addresses that checks each subsection of the IP (separated by '.').

4. Check_Case_P (Port)

   Checks the argument that follows the option for a port number against the acceptable numbers regex. It also checks to make sure that the number provided is between 1024 and 65535.

5. Check_Case_C (Card File)

   Checks the argument that follows the option for a card file against the acceptable file name regex. It also ensures that the card file name provided is between 1 and 127.

6. Check_Case_N (New Account)

   Checks the argument that follows the option for a new account against the acceptable amount regex. It also checks that the initial balance provided is between 10.00 and 4294967295.99.

7. Check_Case_DW (Deposit/Withdrawal)

   Checks the argument that follows the option for a deposit/withdrawal against the acceptable amount regex. It checks that the amount to deposit/withdraw is between 0 and 4294967295.99.

Crypto Protocol

Encrypts and decrypts in CBC mode. The methods in this class are used to encrypt and decode messages sent to/from by the atm/bank and the contents of the users card files. The key used for these methods is stored in the auth file and a new initialization vector is created with every new request sent between the atm and bank.

Attacks and Implemented Defenses

1. Man in the Middle

   One of the main concerns we had was the potential for man in the middle attacks. From the beginning, we knew that encryption and decryption in our program was a must to protect any information the bank clients provide when using the ATM to communicate with the bank. We opted for a cipher block chaining encryption (CBC) mode to secure the data. For every command created in the atm, a new IV is made. This protects against some of the shortcomings of CBC as it would be extremely difficult to guess the IV, and as it is being randomized every time, it significantly reduces the likelihood of that method working for the attacker.

2. Buffer Overflow

   When we create buffers, we ensure that we allocate enough room but have limits on the size of the buffers. We also only ever use strncpy, never strcpy, to prevent any extra data being copied to the buffer.

3. Format String Vulnerability

   The parsing also protects against format string vulnerabilities as well as other suspicious input that could cause buffer overflows. Any input that does not meet the strict format criteria of the project specs does not make it past the parsing stage, and no information will be made available to the attacker.

4. Authentication and Authorization Attack

Encrypting the user's account name with the authfile contents as the IV inside the card file makes it almost impossible to decrypt that information, as the authfile is not readily available to the attackers.

5. Integer Overflow Vulnerability

The regexes we use for files, numbers, and monetary amounts prohibits the user from making transactions with negative signs. It is very important to make this check considering we use unsigned integers in our implementation.

Potential Future Defenses

1. Tampering with Messages

Given more time, we would have protected against message tampering but did not have a chance to implement it. While our messages are encrypted, an attacker could still modify the ciphertext. To defend against this, we would use message authentication codes (tags) to authenticate all requests to preserve their authenticity and integrity. We would likely use the OpenSSL library to create a tag using our symmetric key and message, send the tag in the encrypted message, and then only process the request if the verification using the key, message, and tag worked.

2. Replay Attack

This is also an attack we did not get a chance to protect against but would have liked to. An attacker who can eavesdrop could see our encrypted messages and resend them. This would allow them to potentially withdraw, deposit, or get an individual's balance multiple times. To prevent this, we could set up a challenge response, where the ATM first sends an initial random message to the Bank to start communication. Then, the Bank would send a challenge message such as a timestamp, and the ATM would send back the actual request concatenated with the timestamp, all encrypted. The Bank would then decrypt and verify that the timestamp matches what was sent earlier, and then proceed with operations based on the request. If the timestamp did not match, that would indicate an attacker and the Bank would not fulfill the request.

3. DOS Attack

This is an attack we would have liked to have protected against but did not have time to do so. It is definitely possible for someone to create a lot of traffic to the application and overwhelm it. The bank would crash and then non-malicious users would be denied access to their accounts. To protect against this, we would limit the number of users that can use the application at the same time, as well as prevent too many requests being sent to the atm or bank at once.

4. XSS/CSRF

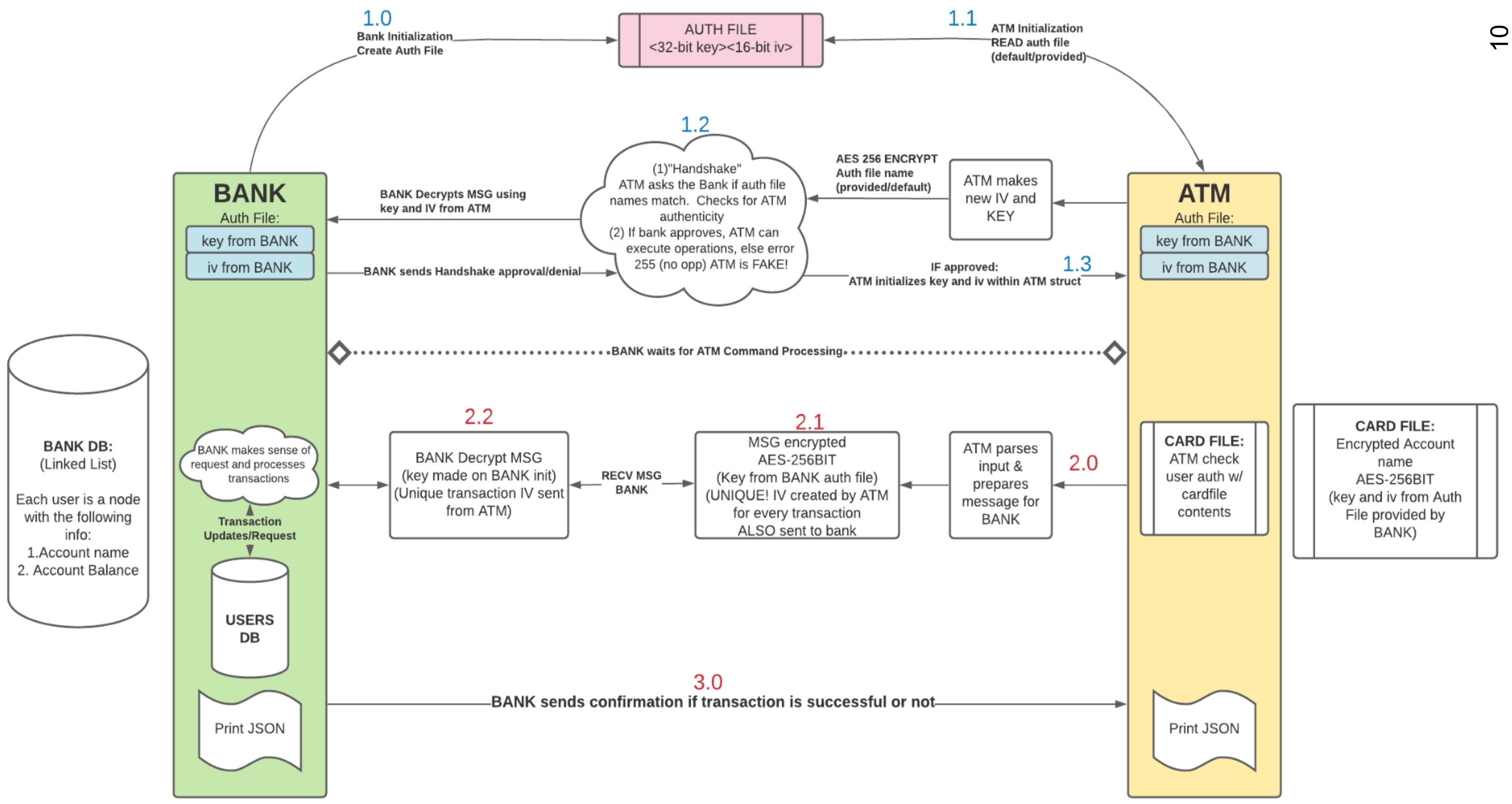
This is definitely an attack that would be worth defending against if the program was running on a site, as this would allow a user to execute javascript in the browser.

5. SQL Injections

We use getopt for POSIX compliance, especially for rule number 5 and to get each option and argument and use a switch statement to check each individual case for validity of user inputs to the command line. This would defend against SQL injections, but we are not storing the information in a SQL database.

Contributions

We worked on this collectively as our schedule allowed, putting in several hours each day over the past month or so. We used a VS code extension (Visual Studio Live Share) to code simultaneously and troubleshoot bugs and other issues together.

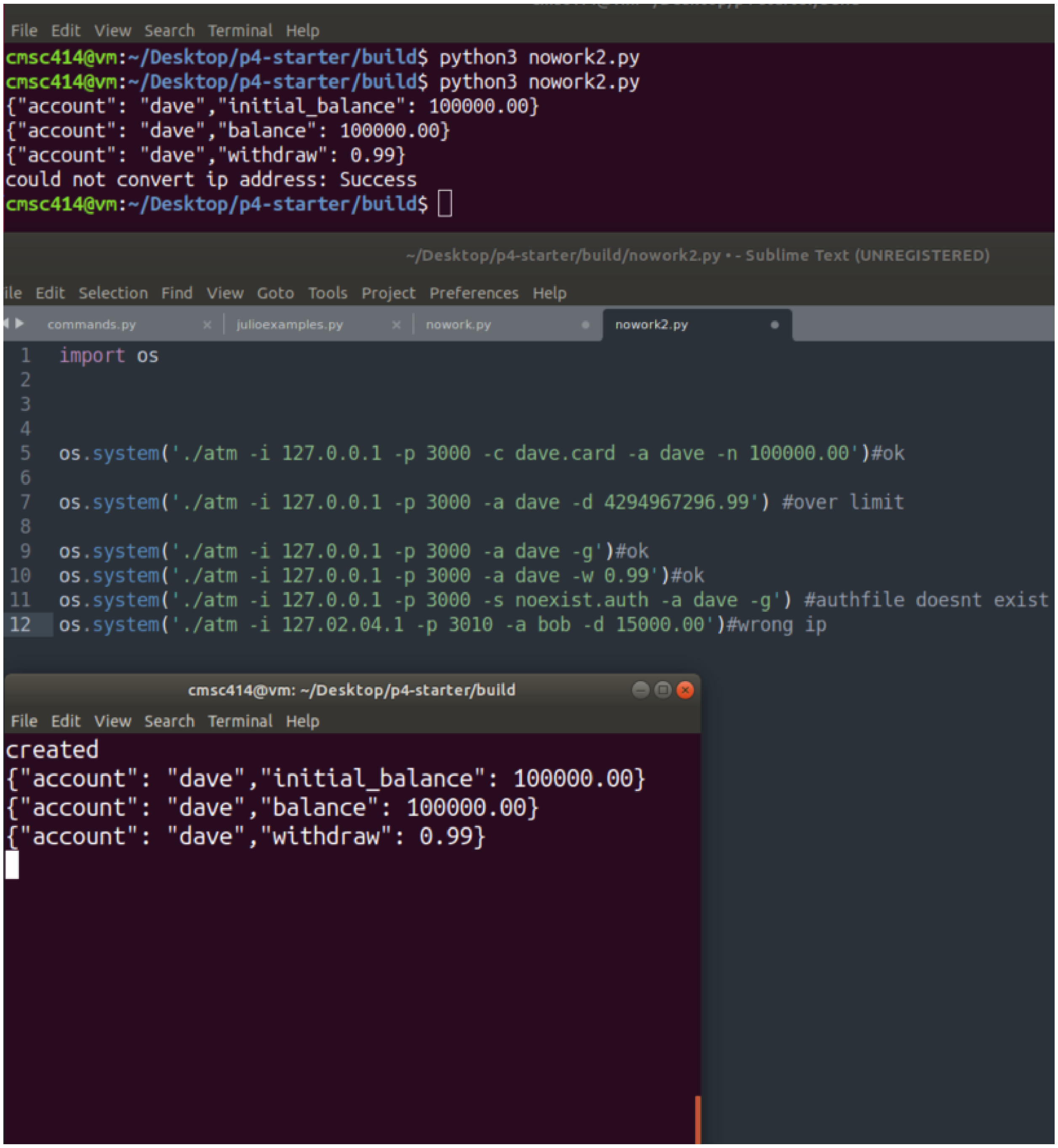

1.0
Bank Initialization
Create Auth File
AUTH FILE
<32-bit key><16-bit iv>
1.1
ATM Initialization
READ auth file
(default/provided)
1.2
(1)"Handshake"
ATM asks the Bank if auth file names match. Checks for ATM authenticity
(2) If bank approves, ATM can execute operations, else error 255 (no opp) ATM is FAKE!
AES 256 ENCRYPT
Auth file name
(provided/default)
ATM makes new IV and KEY
BANK Decrypts MSG using key and IV from ATM
BANK sends Handshake approval/denial
IF approved:
1.3
ATM initializes key and iv within ATM struct
BANK
Auth File:
key from BANK
iv from BANK
ATM
Auth File:
key from BANK
iv from BANK
BANK waits for ATM Command Processing
BANK DB:
(Linked List)
Each user is a node with the following info:
1.Account name
2. Account Balance
BANK makes sense of request and processes transactions
Transaction Updates/Request
USERS DB
2.2
BANK Decrypt MSG
(key made on BANK init)
(Unique transaction IV sent from ATM)
RECV MSG BANK
2.1
MSG encrypted
AES-256BIT
(Key from BANK auth file)
(UNIQUE! IV created by ATM for every transaction
ALSO sent to bank
ATM parses input & prepares message for BANK
2.0
CARD FILE:
ATM check user auth w/ cardfile contents
CARD FILE:
Encrypted Account name
AES-256BIT
(key and iv from Auth File provided by BANK)
3.0
BANK sends confirmation if transaction is successful or not
Print JSON
Print JSON

DETAILS:

- All encryption/decryption is AES-256-CBC (crypto.c)
- Bank & ATM sanitize all inputs following project spec (parser.c && regexcheck.c)
- Database handling (linkedlist.c)
- Communication encryption/decryption is the preventative measure against Man-In-Middle Attacks
- Authentication of ATM and Users prevent malicious atms (fakes) and user impersonation, moreover, prevention of data theft.
- Databases, MSG buffers (or buffers in general) have preallocated memory and ensure null termination.
- Parsing of command-line inputs prevent format string vulnerabilites.
- Interger overflow also covered by parsing

Sample Input and Output

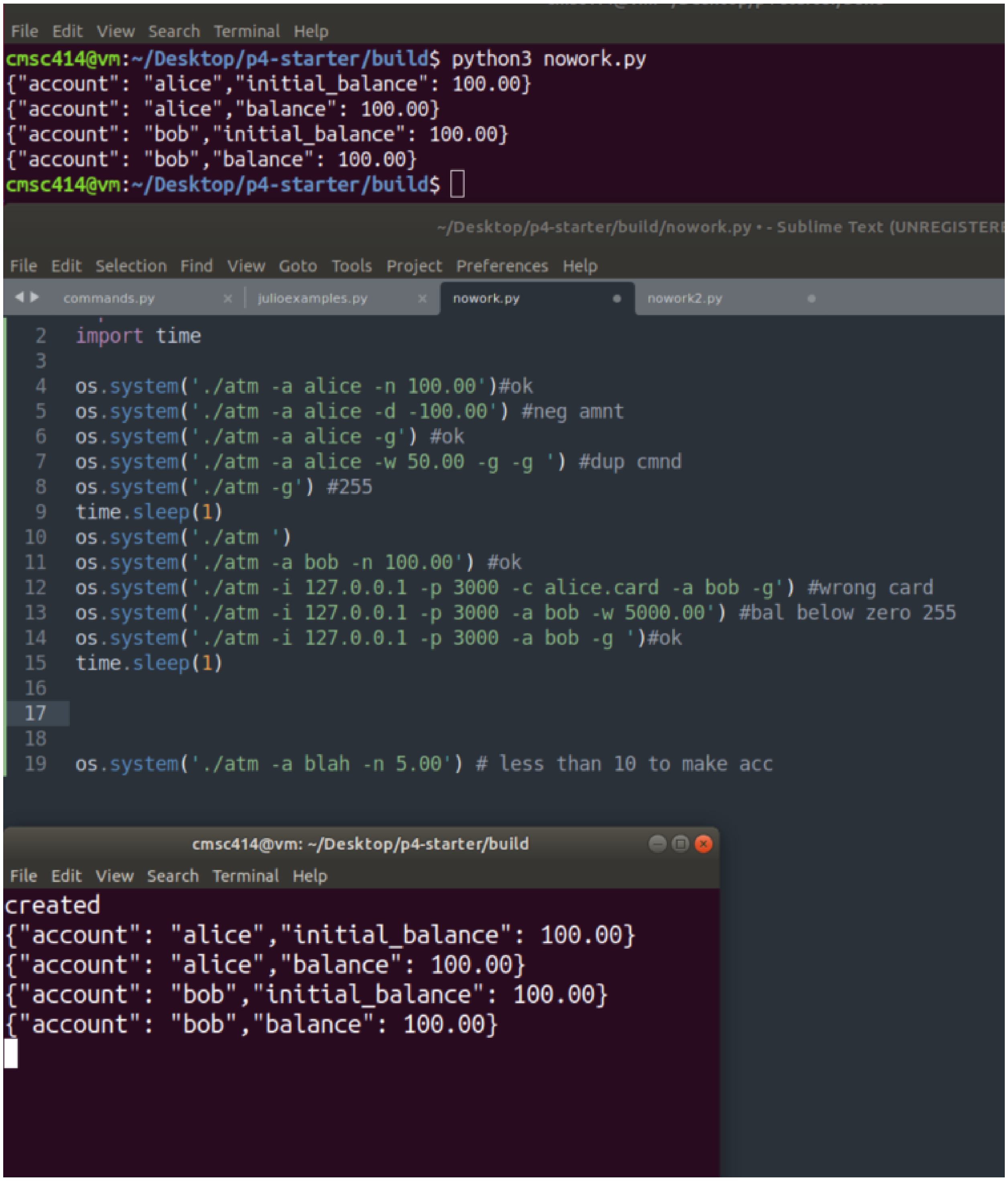
Python scripts with samples included in repo.



```
cmsc414@vm:~/Desktop/p4-starter/build$ python3 nowork2.py
cmsc414@vm:~/Desktop/p4-starter/build$ python3 nowork2.py
{"account": "dave","initial_balance": 100000.00}
{"account": "dave","balance": 100000.00}
{"account": "dave","withdraw": 0.99}
could not convert ip address: Success
cmsc414@vm:~/Desktop/p4-starter/build$
```

```
import os



os.system('./atm -i 127.0.0.1 -p 3000 -c dave.card -a dave -n 100000.00')#ok

os.system('./atm -i 127.0.0.1 -p 3000 -a dave -d 4294967296.99') #over limit

os.system('./atm -i 127.0.0.1 -p 3000 -a dave -g')#ok
os.system('./atm -i 127.0.0.1 -p 3000 -a dave -w 0.99')#ok
os.system('./atm -i 127.0.0.1 -p 3000 -s noexist.auth -a dave -g') #authfile doesnt exist
os.system('./atm -i 127.02.04.1 -p 3010 -a bob -d 15000.00')#wrong ip
```

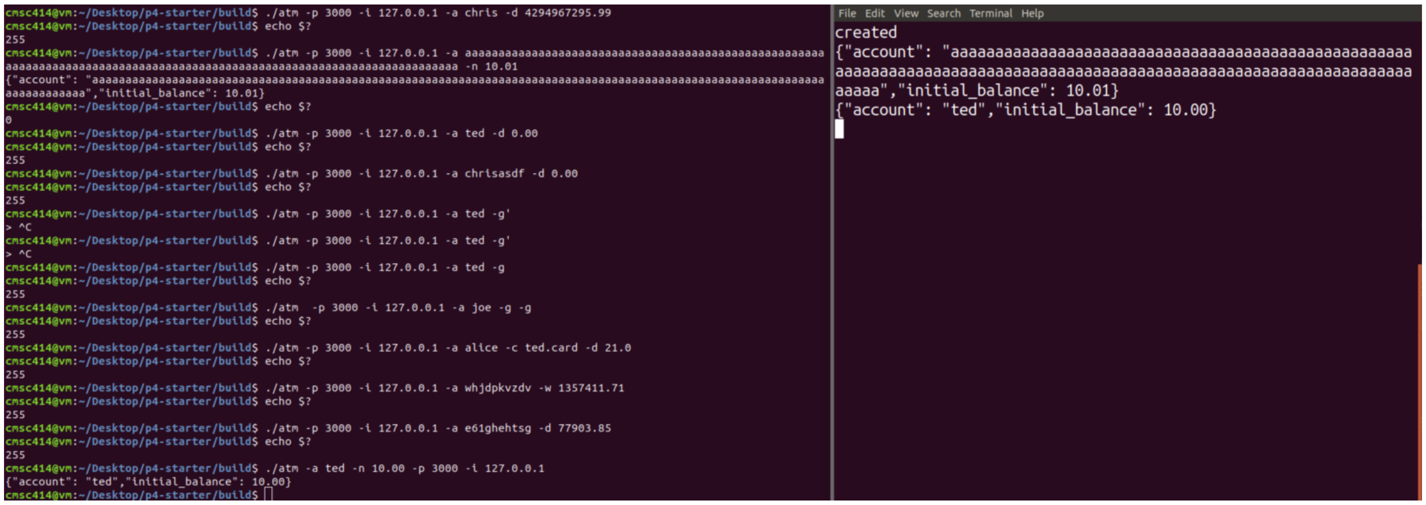
```
created
{"account": "dave","initial_balance": 100000.00}
{"account": "dave","balance": 100000.00}
{"account": "dave","withdraw": 0.99}
```


File Edit View Search Terminal Help
~/Desktop/p4-starter/build/nowork.py • - Sublime Text (UNREGISTERE
File Edit Selection Find View Goto Tools Project Preferences Help
commands.py
julioexamples.py
nowork.py
nowork2.py
cmsc414@vm: ~/Desktop/p4-starter/build
File Edit View Search Terminal Help

```
cmsc414@vm:~/Desktop/p4-starter/build$ python3 nowork.py
{"account": "alice","initial_balance": 100.00}
{"account": "alice","balance": 100.00}
{"account": "bob","initial_balance": 100.00}
{"account": "bob","balance": 100.00}
cmsc414@vm:~/Desktop/p4-starter/build$
```

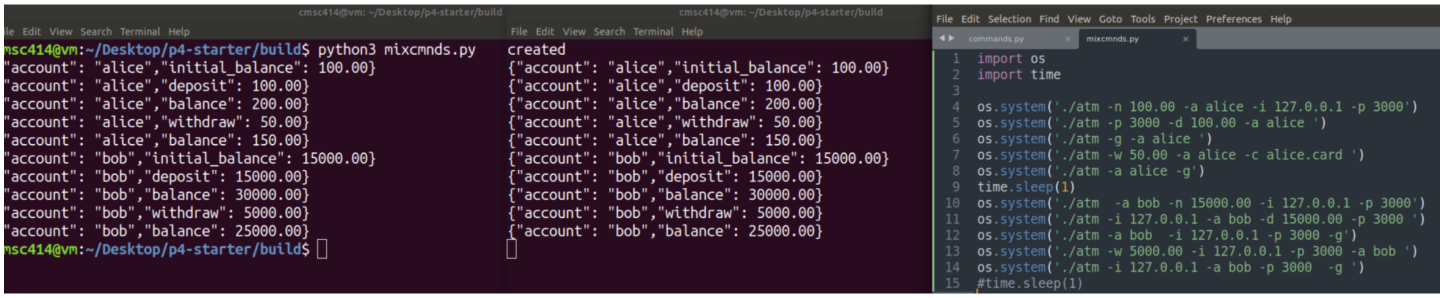
```python
import time

os.system('./atm -a alice -n 100.00')#ok
os.system('./atm -a alice -d -100.00') #neg amnt
os.system('./atm -a alice -g') #ok
os.system('./atm -a alice -w 50.00 -g -g ') #dup cmnd
os.system('./atm -g') #255
time.sleep(1)
os.system('./atm ')
os.system('./atm -a bob -n 100.00') #ok
os.system('./atm -i 127.0.0.1 -p 3000 -c alice.card -a bob -g') #wrong card
os.system('./atm -i 127.0.0.1 -p 3000 -a bob -w 5000.00') #bal below zero 255
os.system('./atm -i 127.0.0.1 -p 3000 -a bob -g ')#ok
time.sleep(1)



os.system('./atm -a blah -n 5.00') # less than 10 to make acc
```

```
created
{"account": "alice","initial_balance": 100.00}
{"account": "alice","balance": 100.00}
{"account": "bob","initial_balance": 100.00}
{"account": "bob","balance": 100.00}
```



```
cmsc414@vm:~/Desktop/p4-starter/build$ ./atm -p 3000 -i 127.0.0.1 -a chris -d 4294967295.99
cmsc414@vm:~/Desktop/p4-starter/build$ echo $?
255
cmsc414@vm:~/Desktop/p4-starter/build$ ./atm -p 3000 -i 127.0.0.1 -a aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa -n 10.01
{"account": "aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaa","initial_balance": 10.01}
cmsc414@vm:~/Desktop/p4-starter/build$ echo $?
0
cmsc414@vm:~/Desktop/p4-starter/build$ ./atm -p 3000 -i 127.0.0.1 -a ted -d 0.00
cmsc414@vm:~/Desktop/p4-starter/build$ echo $?
255
cmsc414@vm:~/Desktop/p4-starter/build$ ./atm -p 3000 -i 127.0.0.1 -a chrisasdf -d 0.00
cmsc414@vm:~/Desktop/p4-starter/build$ echo $?
255
cmsc414@vm:~/Desktop/p4-starter/build$ ./atm -p 3000 -i 127.0.0.1 -a ted -g'
> ^C
cmsc414@vm:~/Desktop/p4-starter/build$ ./atm -p 3000 -i 127.0.0.1 -a ted -g'
> ^C
cmsc414@vm:~/Desktop/p4-starter/build$ ./atm -p 3000 -i 127.0.0.1 -a ted -g
cmsc414@vm:~/Desktop/p4-starter/build$ echo $?
255
cmsc414@vm:~/Desktop/p4-starter/build$ ./atm  -p 3000 -i 127.0.0.1 -a joe -g -g
cmsc414@vm:~/Desktop/p4-starter/build$ echo $?
255
cmsc414@vm:~/Desktop/p4-starter/build$ ./atm -p 3000 -i 127.0.0.1 -a alice -c ted.card -d 21.0
cmsc414@vm:~/Desktop/p4-starter/build$ echo $?
255
cmsc414@vm:~/Desktop/p4-starter/build$ ./atm -p 3000 -i 127.0.0.1 -a whjdpkvzdv -w 1357411.71
cmsc414@vm:~/Desktop/p4-starter/build$ echo $?
255
cmsc414@vm:~/Desktop/p4-starter/build$ ./atm -p 3000 -i 127.0.0.1 -a e61ghehtsg -d 77903.85
cmsc414@vm:~/Desktop/p4-starter/build$ echo $?
255
cmsc414@vm:~/Desktop/p4-starter/build$ ./atm -a ted -n 10.00 -p 3000 -i 127.0.0.1
{"account": "ted","initial_balance": 10.00}
cmsc414@vm:~/Desktop/p4-starter/build$ 
```

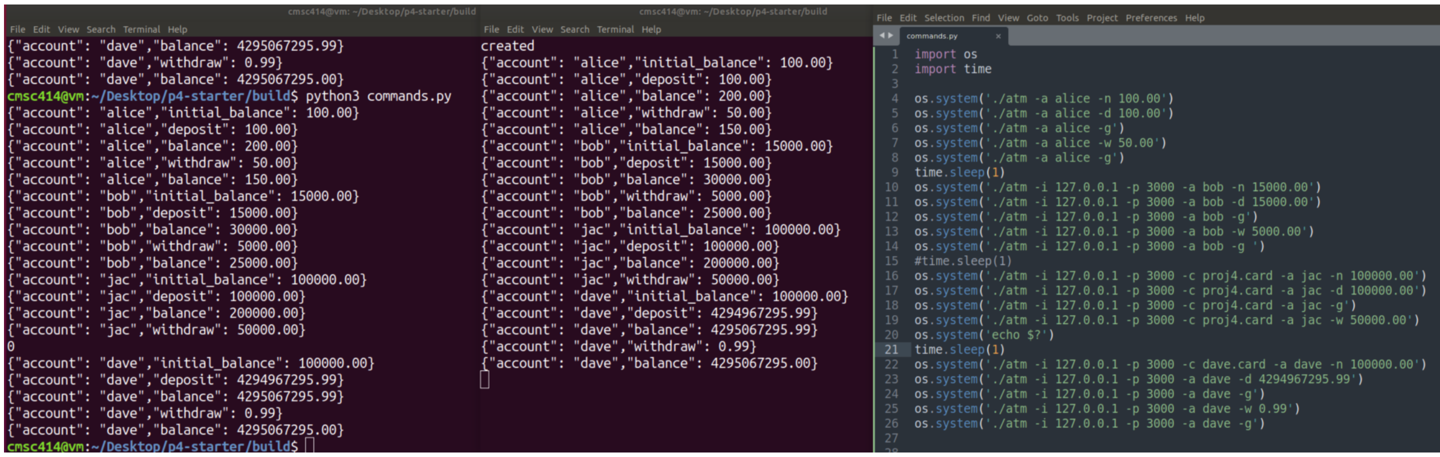
```
created
{"account": "aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaa","initial_balance": 10.01}
{"account": "ted","initial_balance": 10.00}
```



```
msc414@vm:~/Desktop/p4-starter/build$ python3 mixcmnds.py
"account": "alice","initial_balance": 100.00}
"account": "alice","deposit": 100.00}
"account": "alice","balance": 200.00}
"account": "alice","withdraw": 50.00}
"account": "alice","balance": 150.00}
"account": "bob","initial_balance": 15000.00}
"account": "bob","deposit": 15000.00}
"account": "bob","balance": 30000.00}
"account": "bob","withdraw": 5000.00}
"account": "bob","balance": 25000.00}
msc414@vm:~/Desktop/p4-starter/build$ 
```

```
created
{"account": "alice","initial_balance": 100.00}
{"account": "alice","deposit": 100.00}
{"account": "alice","balance": 200.00}
{"account": "alice","withdraw": 50.00}
{"account": "alice","balance": 150.00}
{"account": "bob","initial_balance": 15000.00}
{"account": "bob","deposit": 15000.00}
{"account": "bob","balance": 30000.00}
{"account": "bob","withdraw": 5000.00}
{"account": "bob","balance": 25000.00}
```

```python
import os
import time

os.system('./atm -n 100.00 -a alice -i 127.0.0.1 -p 3000')
os.system('./atm -p 3000 -d 100.00 -a alice ')
os.system('./atm -g -a alice ')
os.system('./atm -w 50.00 -a alice -c alice.card ')
os.system('./atm -a alice -g')
time.sleep(1)
os.system('./atm  -a bob -n 15000.00 -i 127.0.0.1 -p 3000')
os.system('./atm -i 127.0.0.1 -a bob -d 15000.00 -p 3000 ')
os.system('./atm -a bob  -i 127.0.0.1 -p 3000 -g')
os.system('./atm -w 5000.00 -i 127.0.0.1 -p 3000 -a bob ')
os.system('./atm -i 127.0.0.1 -a bob -p 3000  -g ')
#time.sleep(1)
```

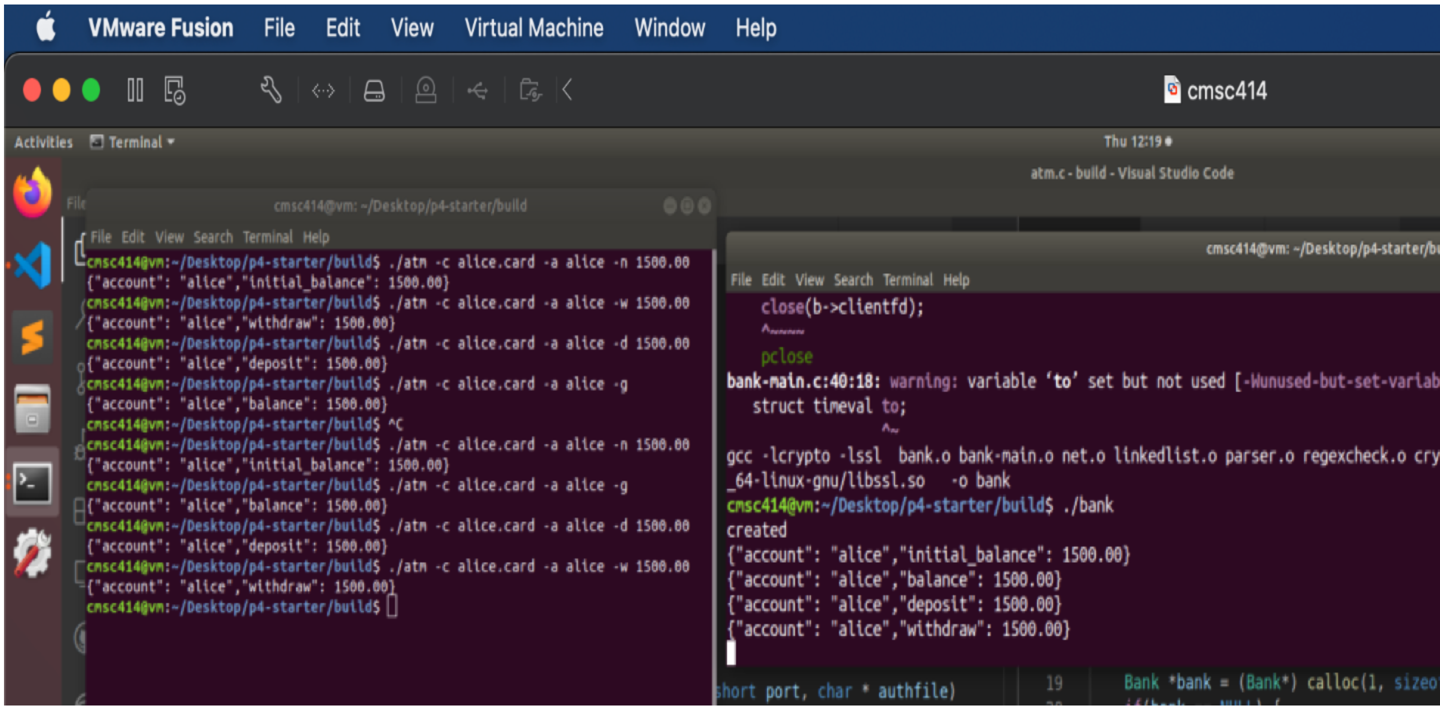


```
{"account": "dave","balance": 4295067295.99}
{"account": "dave","withdraw": 0.99}
{"account": "dave","balance": 4295067295.00}
cmsc414@vm:~/Desktop/p4-starter/build$ python3 commands.py
{"account": "alice","initial_balance": 100.00}
{"account": "alice","deposit": 100.00}
{"account": "alice","balance": 200.00}
{"account": "alice","withdraw": 50.00}
{"account": "alice","balance": 150.00}
{"account": "bob","initial_balance": 15000.00}
{"account": "bob","deposit": 15000.00}
{"account": "bob","balance": 30000.00}
{"account": "bob","withdraw": 5000.00}
{"account": "bob","balance": 25000.00}
{"account": "jac","initial_balance": 100000.00}
{"account": "jac","deposit": 100000.00}
{"account": "jac","balance": 200000.00}
{"account": "jac","withdraw": 50000.00}
0
{"account": "dave","initial_balance": 100000.00}
{"account": "dave","deposit": 4294967295.99}
{"account": "dave","balance": 4295067295.99}
{"account": "dave","withdraw": 0.99}
{"account": "dave","balance": 4295067295.00}
cmsc414@vm:~/Desktop/p4-starter/build$
```

```
created
{"account": "alice","initial_balance": 100.00}
{"account": "alice","deposit": 100.00}
{"account": "alice","balance": 200.00}
{"account": "alice","withdraw": 50.00}
{"account": "alice","balance": 150.00}
{"account": "bob","initial_balance": 15000.00}
{"account": "bob","deposit": 15000.00}
{"account": "bob","balance": 30000.00}
{"account": "bob","withdraw": 5000.00}
{"account": "bob","balance": 25000.00}
{"account": "jac","initial_balance": 100000.00}
{"account": "jac","deposit": 100000.00}
{"account": "jac","balance": 200000.00}
{"account": "jac","withdraw": 50000.00}
{"account": "dave","initial_balance": 100000.00}
{"account": "dave","deposit": 4294967295.99}
{"account": "dave","balance": 4295067295.99}
{"account": "dave","withdraw": 0.99}
{"account": "dave","balance": 4295067295.00}
```

commands.py

```python
import os
import time

os.system('./atm -a alice -n 100.00')
os.system('./atm -a alice -d 100.00')
os.system('./atm -a alice -g')
os.system('./atm -a alice -w 50.00')
os.system('./atm -a alice -g')
time.sleep(1)
os.system('./atm -i 127.0.0.1 -p 3000 -a bob -n 15000.00')
os.system('./atm -i 127.0.0.1 -p 3000 -a bob -d 15000.00')
os.system('./atm -i 127.0.0.1 -p 3000 -a bob -g')
os.system('./atm -i 127.0.0.1 -p 3000 -a bob -w 5000.00')
os.system('./atm -i 127.0.0.1 -p 3000 -a bob -g ')
#time.sleep(1)
os.system('./atm -i 127.0.0.1 -p 3000 -c proj4.card -a jac -n 100000.00')
os.system('./atm -i 127.0.0.1 -p 3000 -c proj4.card -a jac -d 100000.00')
os.system('./atm -i 127.0.0.1 -p 3000 -c proj4.card -a jac -g')
os.system('./atm -i 127.0.0.1 -p 3000 -c proj4.card -a jac -w 50000.00')
os.system('echo $?')
time.sleep(1)
os.system('./atm -i 127.0.0.1 -p 3000 -c dave.card -a dave -n 100000.00')
os.system('./atm -i 127.0.0.1 -p 3000 -a dave -d 4294967295.99')
os.system('./atm -i 127.0.0.1 -p 3000 -a dave -g')
os.system('./atm -i 127.0.0.1 -p 3000 -a dave -w 0.99')
os.system('./atm -i 127.0.0.1 -p 3000 -a dave -g')
```



```
cmsc414@vm:~/Desktop/p4-starter/build$ ./atm -c alice.card -a alice -n 1500.00
{"account": "alice","initial_balance": 1500.00}
cmsc414@vm:~/Desktop/p4-starter/build$ ./atm -c alice.card -a alice -w 1500.00
{"account": "alice","withdraw": 1500.00}
cmsc414@vm:~/Desktop/p4-starter/build$ ./atm -c alice.card -a alice -d 1500.00
{"account": "alice","deposit": 1500.00}
cmsc414@vm:~/Desktop/p4-starter/build$ ./atm -c alice.card -a alice -g
{"account": "alice","balance": 1500.00}
cmsc414@vm:~/Desktop/p4-starter/build$ ^C
cmsc414@vm:~/Desktop/p4-starter/build$ ./atm -c alice.card -a alice -n 1500.00
{"account": "alice","initial_balance": 1500.00}
cmsc414@vm:~/Desktop/p4-starter/build$ ./atm -c alice.card -a alice -g
{"account": "alice","balance": 1500.00}
cmsc414@vm:~/Desktop/p4-starter/build$ ./atm -c alice.card -a alice -d 1500.00
{"account": "alice","deposit": 1500.00}
cmsc414@vm:~/Desktop/p4-starter/build$ ./atm -c alice.card -a alice -w 1500.00
{"account": "alice","withdraw": 1500.00}
cmsc414@vm:~/Desktop/p4-starter/build$
```

```
    close(b->clientfd);
    ^~~~~~
    pclose
bank-main.c:40:18: warning: variable 'to' set but not used [-Wunused-but-set-variab
  struct timeval to;
                 ^~

gcc -lcrypto -lssl  bank.o bank-main.o net.o linkedlist.o parser.o regexcheck.o cry
_64-linux-gnu/libssl.so   -o bank
cmsc414@vm:~/Desktop/p4-starter/build$ ./bank
created
{"account": "alice","initial_balance": 1500.00}
{"account": "alice","balance": 1500.00}
{"account": "alice","deposit": 1500.00}
{"account": "alice","withdraw": 1500.00}
```