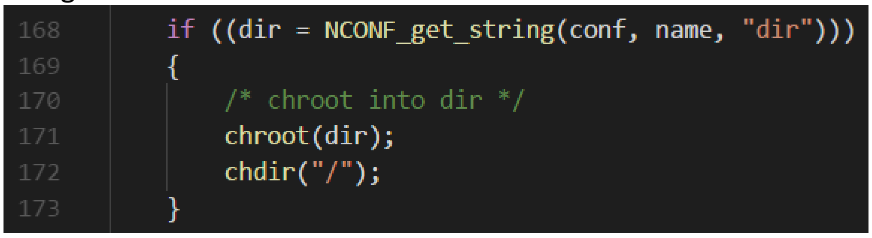
B) Web server setup using Unix principals and permissions

Exercise 2:

To insert the call to `chroot`, we change root directory to `dir`, then we change working directory to "/" in `zookld.c` lines 171 and 172 as shown in Figure 1. This jails the process being created.



```
168     if ((dir = NCONF_get_string(conf, name, "dir")))
169     {
170         /* chroot into dir */
171         chroot(dir);
172         chdir("/");
173     }
```

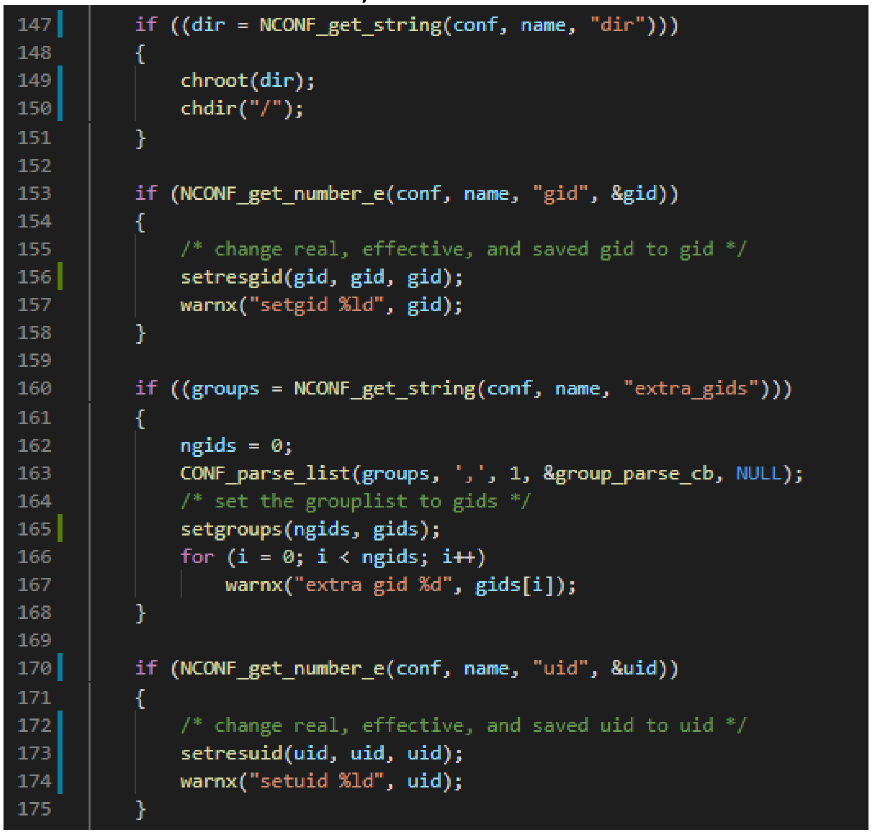
Figure 1: Insert chroot in zookld.c

Exercise 3:

First, we needed to modify the function `launch_svc` in `zookld.c` so that it sets the user and group IDs and the supplementary group list specified in `zook.conf` by using the system calls `setresuid`, `setresgid`, and `setgroups`.

The correct order in which the system calls should be called is `setresgid`, `setgroups` and `setresuid`. The `gid` is set before the `uid` as we want to set the `uids` of all those in the same group. The `uid` must be changed last as only superusers can set the list of supplementary group IDs using `setgroups`.

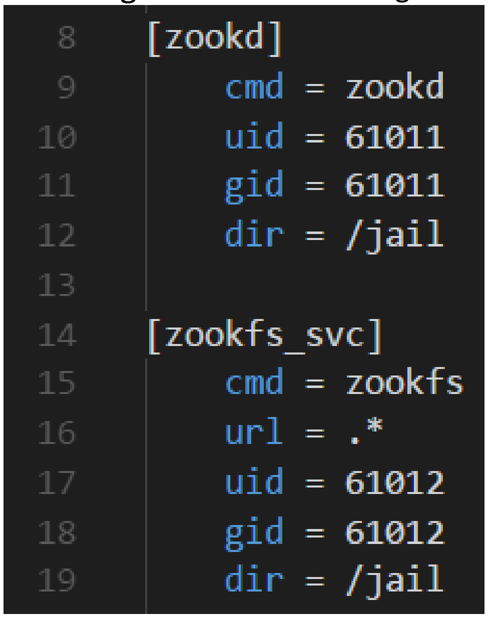
This can be seen in Figure 2 in lines 149, 156, 165 and 173 respectively, where the system calls have been ordered correctly.



```
147     if ((dir = NCONF_get_string(conf, name, "dir")))
148     {
149         chroot(dir);
150         chdir("/");
151     }
152 
153     if (NCONF_get_number_e(conf, name, "gid", &gid))
154     {
155         /* change real, effective, and saved gid to gid */
156         setresgid(gid, gid, gid);
157         warnx("setgid %ld", gid);
158     }
159 
160     if ((groups = NCONF_get_string(conf, name, "extra_gids")))
161     {
162         ngids = 0;
163         CONF_parse_list(groups, ',', 1, &group_parse_cb, NULL);
164         /* set the grouplist to gids */
165         setgroups(ngids, gids);
166         for (i = 0; i < ngids; i++)
167             warnx("extra gid %d", gids[i]);
168     }
169 
170     if (NCONF_get_number_e(conf, name, "uid", &uid))
171     {
172         /* change real, effective, and saved uid to uid */
173         setresuid(uid, uid, uid);
174         warnx("setuid %ld", uid);
175     }
```

Figure 2: System calls in zookld.c

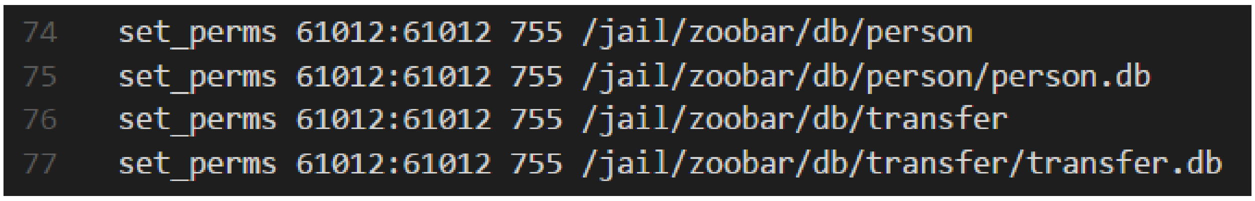
Secondly, we need to change the `zook.conf` `uid` and `gid` for `zookd` and `zookfs` so that they run as something other than root, i.e. with a `uid` and `gid` that is not 0. The modified `uid` and `gid` can be seen in Figure 3 below.



```
[zookd]
    cmd = zookd
    uid = 61011
    gid = 61011
    dir = /jail

[zookfs_svc]
    cmd = zookfs
    url = .*
    uid = 61012
    gid = 61012
    dir = /jail
```

Figure 3: Modifying uid & gid in zook.conf

Lastly, we had to modify the `chroot-setup.sh` to ensure that the files on disk, such as the database, can be read only by the processes that should be able to read them. Instead of using the built-in `chmod` and `chown` commands, we used the provided `set_perms` function. The octal notation is used to determine who in the user, group and others can read, write and execute. This was a very helpful resource. We only want the user to be able to write while the group and others should only be able to read and execute. As such, the access permissions were set as shown in Figure 4.



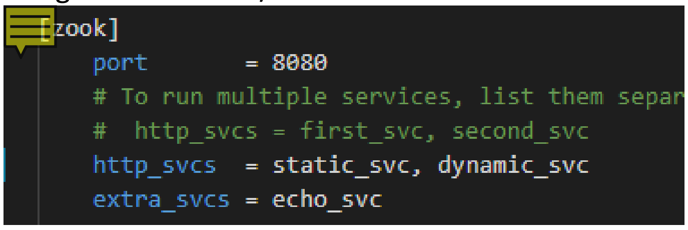
```
set_perms 61012:61012 755 /jail/zoobar/db/person
set_perms 61012:61012 755 /jail/zoobar/db/person/person.db
set_perms 61012:61012 755 /jail/zoobar/db/transfer
set_perms 61012:61012 755 /jail/zoobar/db/transfer/transfer.db
```

Figure 4: Modified permissions in chroot-setup.sh

After making modifications to the `chroot-setup.sh` file, we had to run `sudo make setup` again before running `sudo make check` to verify that we successfully completed the activity.

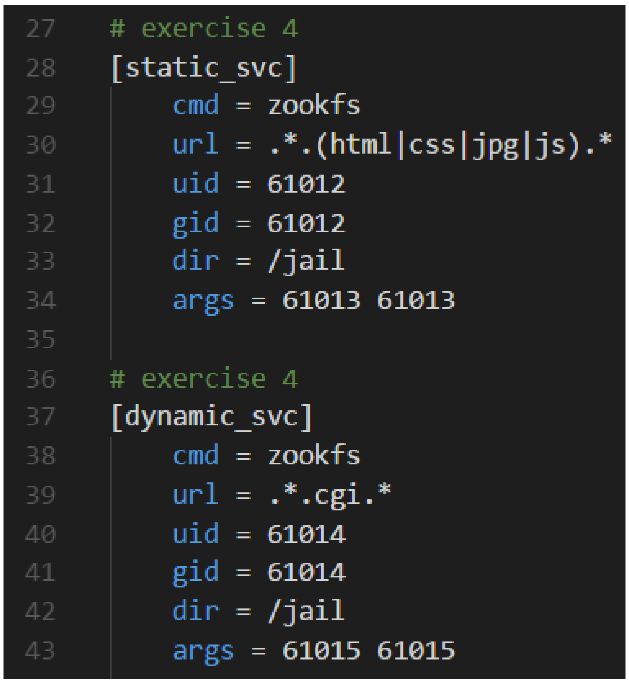
Exercise 4:

We were able to privilege-separate the `zookfs_svc` service in `zook.conf` that handles both static files and dynamic scripts using `static_svc` and `dynamic_svc` with different user and group IDs as shown in Figure 5 and 6. The appropriate URL filters were applied accordingly as well as the required arguments for the static and dynamic services to ensure that the correct things are executed/not executed.



```
[zook]
    port       = 8080
    # To run multiple services, list them separ
    #  http_svcs = first_svc, second_svc
    http_svcs  = static_svc, dynamic_svc
    extra_svcs = echo_svc
```

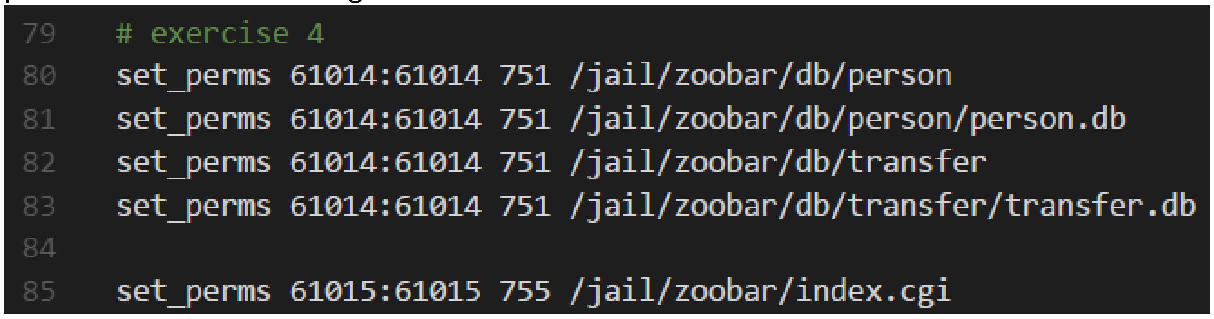
Figure 5: Privilege-separate the zookfs_svc service in zook.conf



```
# exercise 4
[static_svc]
    cmd = zookfs
    url = .*.(html|css|jpg|js).*
    uid = 61012
    gid = 61012
    dir = /jail
    args = 61013 61013

# exercise 4
[dynamic_svc]
    cmd = zookfs
    url = .*.cgi.*
    uid = 61014
    gid = 61014
    dir = /jail
    args = 61015 61015
```

Figure 6: Privilege-separate the zookfs_svc service in zook.conf

We used the provided `set_perms` function in the `chroot-setup.sh` file, setting the access permissions as shown in Figure 6.



```
# exercise 4
set_perms 61014:61014 751 /jail/zoobar/db/person
set_perms 61014:61014 751 /jail/zoobar/db/person/person.db
set_perms 61014:61014 751 /jail/zoobar/db/transfer
set_perms 61014:61014 751 /jail/zoobar/db/transfer/transfer.db

set_perms 61015:61015 755 /jail/zoobar/index.cgi
```

Figure 6: Modified permissions in chroot-setup.sh

After making modifications to the `chroot-setup.sh` file, we had to run `sudo make setup` again before running `sudo make check` to verify that we successfully completed the activity.

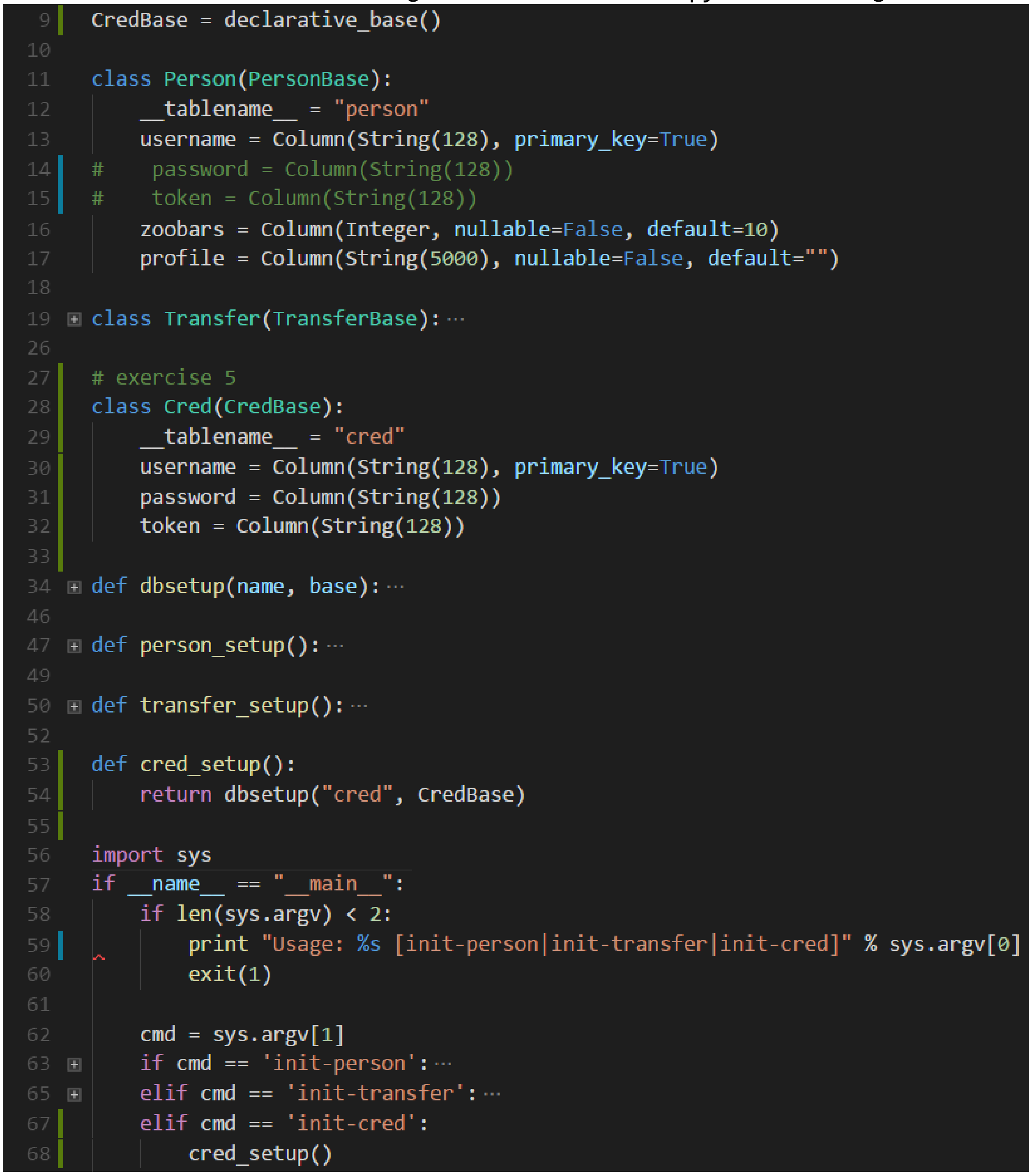
D) Privilege-separating the login service in Zoobar

Exercise 5:

Privilege-separating the login service in Zoobar was extremely tedious. 7 individual files needed to be modified:

1. `zoodb.py`

A new `Cred` database had to be created and passwords or tokens were to be removed from the old `Person` database. These changes were made in `zoodb.py` as shown in Figure 7.

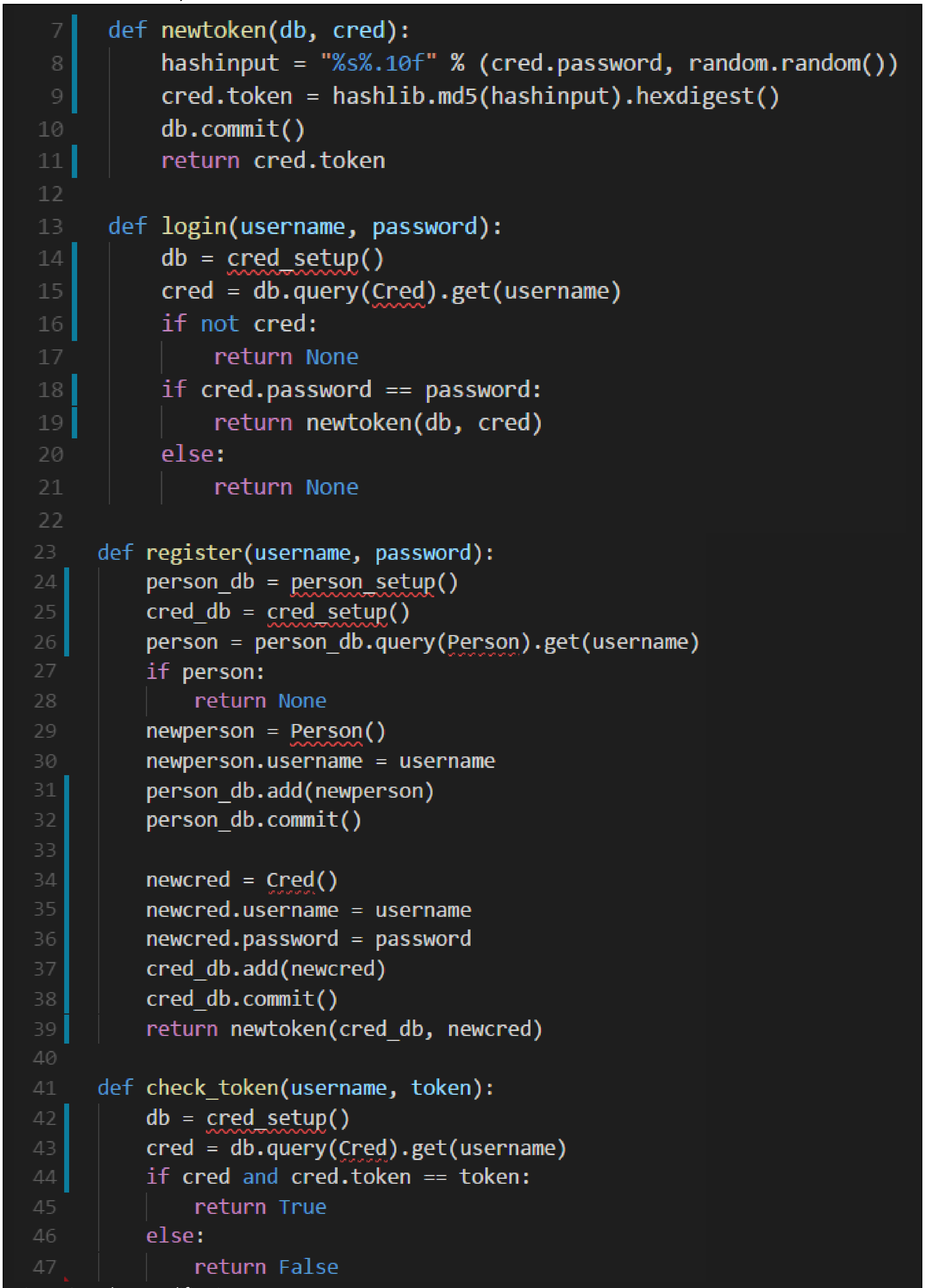


```python
CredBase = declarative_base()

class Person(PersonBase):
    __tablename__ = "person"
    username = Column(String(128), primary_key=True)
#    password = Column(String(128))
#    token = Column(String(128))
    zoobars = Column(Integer, nullable=False, default=10)
    profile = Column(String(5000), nullable=False, default="")

class Transfer(TransferBase): ···

# exercise 5
class Cred(CredBase):
    __tablename__ = "cred"
    username = Column(String(128), primary_key=True)
    password = Column(String(128))
    token = Column(String(128))

def dbsetup(name, base): ···

def person_setup(): ···

def transfer_setup(): ···

def cred_setup():
    return dbsetup("cred", CredBase)

import sys
if __name__ == "__main__":
    if len(sys.argv) < 2:
        print "Usage: %s [init-person|init-transfer|init-cred]" % sys.argv[0]
        exit(1)

    cmd = sys.argv[1]
    if cmd == 'init-person': ···
    elif cmd == 'init-transfer': ···
    elif cmd == 'init-cred':
        cred_setup()
```

Figure 7: zoodb.py modifications

2. `auth.py`

`auth.py` was modified to have the appropriate information returned and stored in the correct databases, `Cred` or `Person`.

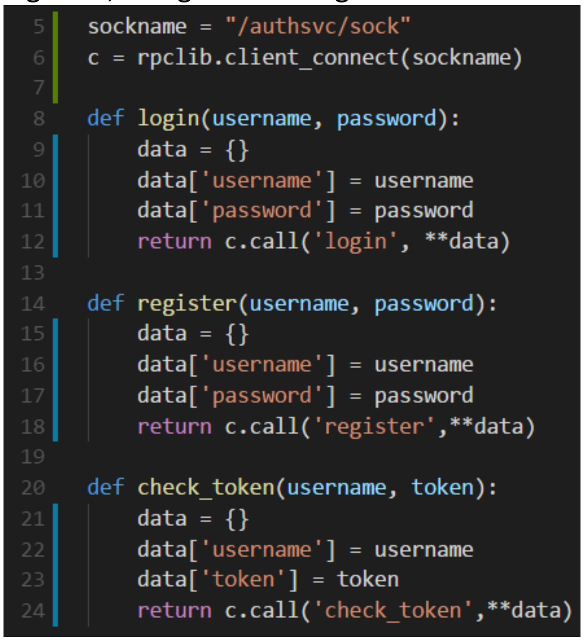


```python
def newtoken(db, cred):
    hashinput = "%s%.10f" % (cred.password, random.random())
    cred.token = hashlib.md5(hashinput).hexdigest()
    db.commit()
    return cred.token

def login(username, password):
    db = cred_setup()
    cred = db.query(Cred).get(username)
    if not cred:
        return None
    if cred.password == password:
        return newtoken(db, cred)
    else:
        return None

def register(username, password):
    person_db = person_setup()
    cred_db = cred_setup()
    person = person_db.query(Person).get(username)
    if person:
        return None
    newperson = Person()
    newperson.username = username
    person_db.add(newperson)
    person_db.commit()

    newcred = Cred()
    newcred.username = username
    newcred.password = password
    cred_db.add(newcred)
    cred_db.commit()
    return newtoken(cred_db, newcred)

def check_token(username, token):
    db = cred_setup()
    cred = db.query(Cred).get(username)
    if cred and cred.token == token:
        return True
    else:
        return False
```

Figure 8: auth.py modifications

3. `auth_client.py`

The initial RPC stubs for the client in `zoobar/auth_client.py` was completed as shown in Figure 9, using the existing functions in `auth.py`.

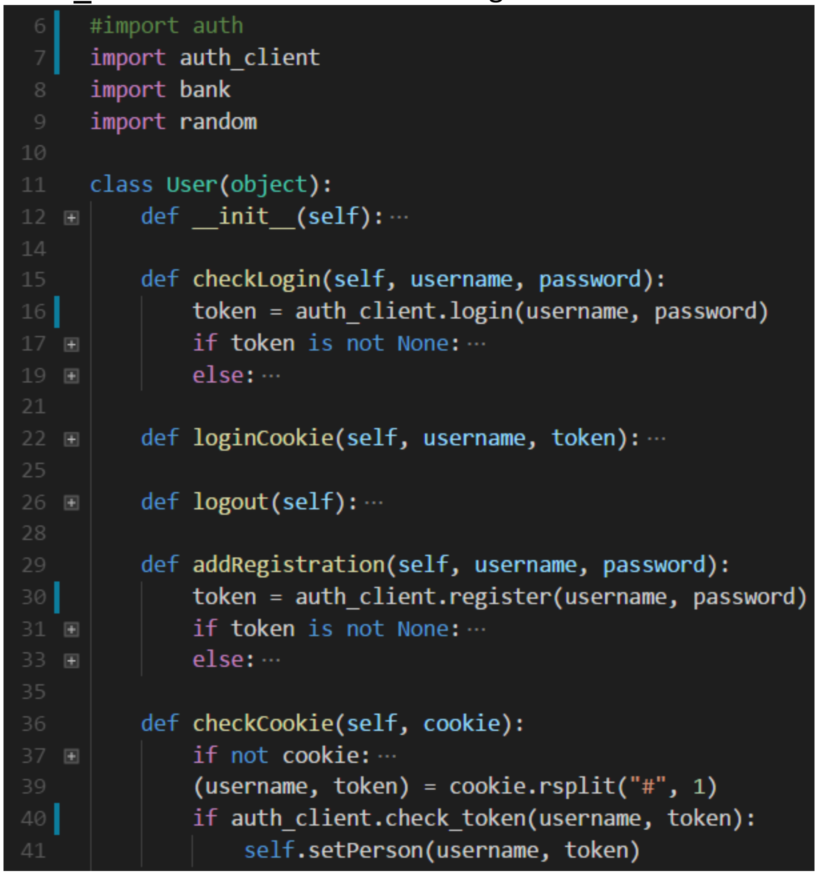


```python
sockname = "/authsvc/sock"
c = rpclib.client_connect(sockname)

def login(username, password):
    data = {}
    data['username'] = username
    data['password'] = password
    return c.call('login', **data)

def register(username, password):
    data = {}
    data['username'] = username
    data['password'] = password
    return c.call('register',**data)

def check_token(username, token):
    data = {}
    data['username'] = username
    data['token'] = token
    return c.call('check_token',**data)
```

Figure 9: auth_client.py modifications

4. `login.py`

The login code in `login.py` was modified to invoke our new auth service instead of calling `auth.py` directly by importing `auth_client` and replacing all instances of `auth` with `auth_client` instead as shown in Figure 10.

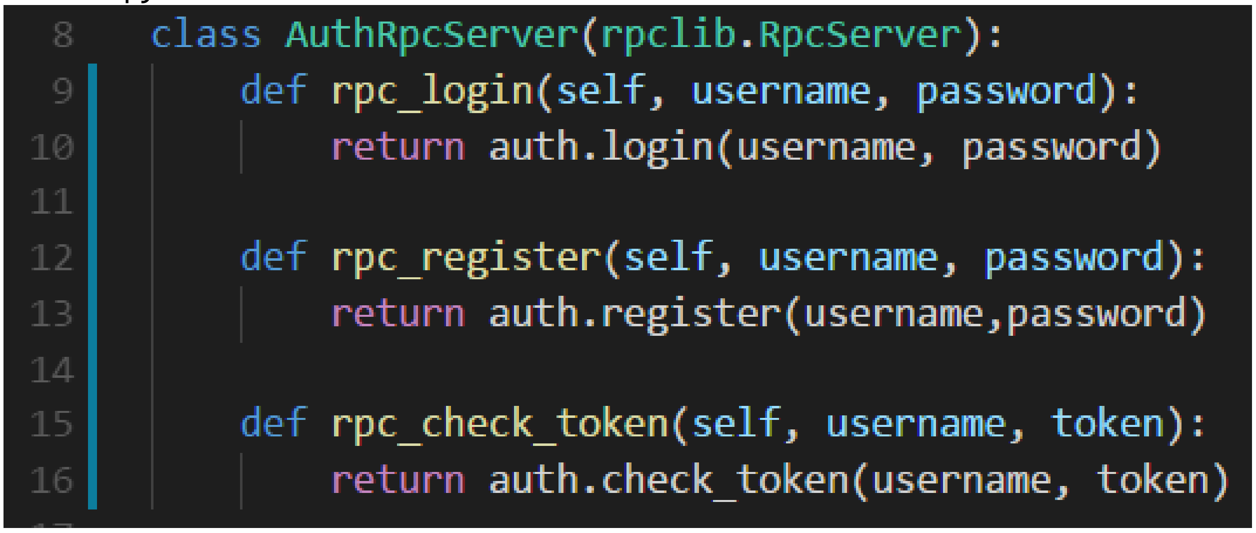


```python
#import auth
import auth_client
import bank
import random

class User(object):
    def __init__(self):···

    def checkLogin(self, username, password):
        token = auth_client.login(username, password)
        if token is not None:···
        else:···

    def loginCookie(self, username, token):···

    def logout(self):···

    def addRegistration(self, username, password):
        token = auth_client.register(username, password)
        if token is not None:···
        else:···

    def checkCookie(self, cookie):
        if not cookie:···
        (username, token) = cookie.rsplit("#", 1)
        if auth_client.check_token(username, token):
            self.setPerson(username, token)
```

Figure 10: login.py modifications

5. `auth-server.py`

The new `auth_svc` service for user authentication was created by modifying the initial file `zoobar/auth_server.py` as shown in Figure 11. Once again, using the existing functions in `auth.py`.

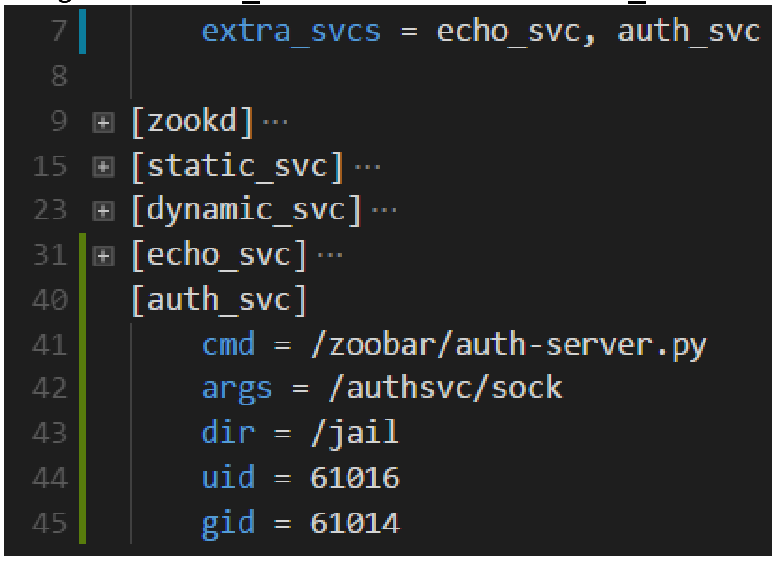


```python
class AuthRpcServer(rpclib.RpcServer):
    def rpc_login(self, username, password):
        return auth.login(username, password)

    def rpc_register(self, username, password):
        return auth.register(username,password)

    def rpc_check_token(self, username, token):
        return auth.check_token(username, token)
```

Figure 11: auth-server.py modifications

6. `zook.conf`

In order to start the `auth_server` appropriately under a different UID but the same GID as `dynamic_svc` since `dynamic_svc` uses the auth server, `zook.conf` was modified as shown in Figure 12. `auth_svc` was added as an `extra_svc` service to be run as shown in line 7.

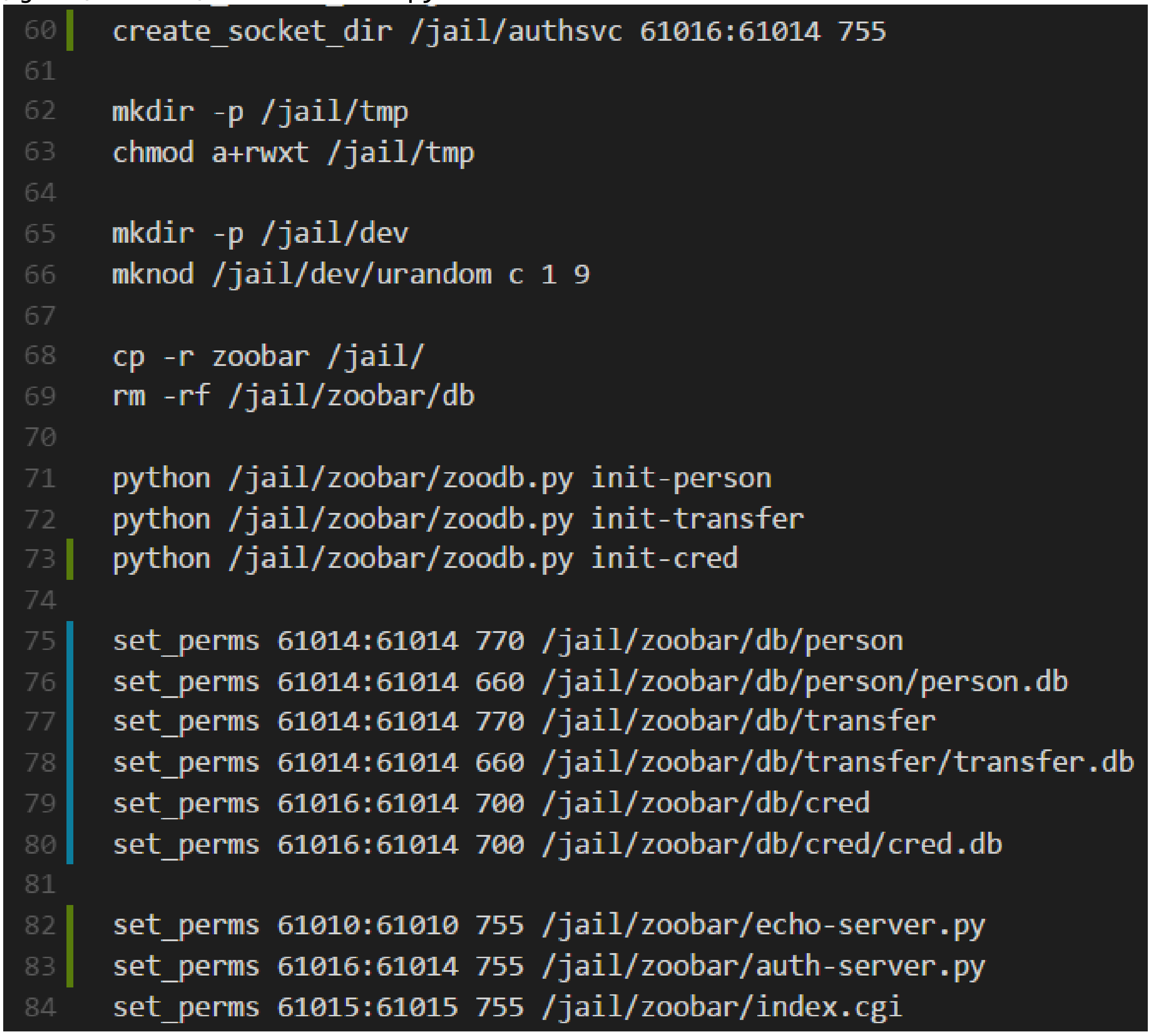


```
 7      extra_svcs = echo_svc, auth_svc
 8
 9  ⊞ [zookd] ···
15  ⊞ [static_svc] ···
23  ⊞ [dynamic_svc] ···
31  ⊞ [echo_svc] ···
40    [auth_svc]
41      cmd = /zoobar/auth-server.py
42      args = /authsvc/sock
43      dir = /jail
44      uid = 61016
45      gid = 61014
```

Figure 12: zook.conf modifications

7. `chroot-setup.sh`

The `chroot-setup.sh` file was modified as shown in Figure 13. Line 60 was added to create the socket for the new `auth_svc`. Lines 73 to 83 were added to set the permissions on the `Cred` database appropriately, applying the execute permissions on the binary `/jail/zoobar/auth-server.py` as well.



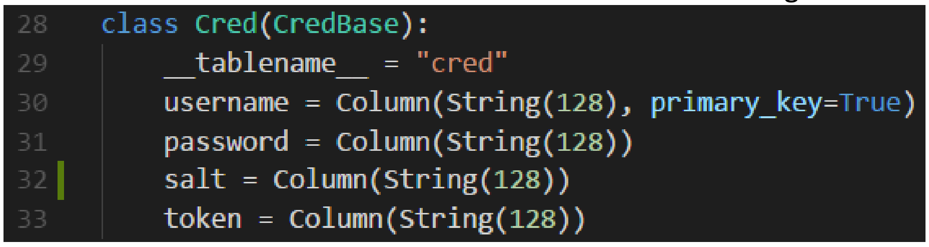
```
60  create_socket_dir /jail/authsvc 61016:61014 755
61
62  mkdir -p /jail/tmp
63  chmod a+rwxt /jail/tmp
64
65  mkdir -p /jail/dev
66  mknod /jail/dev/urandom c 1 9
67
68  cp -r zoobar /jail/
69  rm -rf /jail/zoobar/db
70
71  python /jail/zoobar/zoodb.py init-person
72  python /jail/zoobar/zoodb.py init-transfer
73  python /jail/zoobar/zoodb.py init-cred
74
75  set_perms 61014:61014 770 /jail/zoobar/db/person
76  set_perms 61014:61014 660 /jail/zoobar/db/person/person.db
77  set_perms 61014:61014 770 /jail/zoobar/db/transfer
78  set_perms 61014:61014 660 /jail/zoobar/db/transfer/transfer.db
79  set_perms 61016:61014 700 /jail/zoobar/db/cred
80  set_perms 61016:61014 700 /jail/zoobar/db/cred/cred.db
81
82  set_perms 61010:61010 755 /jail/zoobar/echo-server.py
83  set_perms 61016:61014 755 /jail/zoobar/auth-server.py
84  set_perms 61015:61015 755 /jail/zoobar/index.cgi
```

Figure 13: chroot-setup.h modifications

After making the necessary modifications to each of these 7 files, we had to run `sudo make setup` again before running `sudo make check` to verify that we successfully completed the activity, successfully privilege-separating the login service in Zoobar.

Exercise 6:

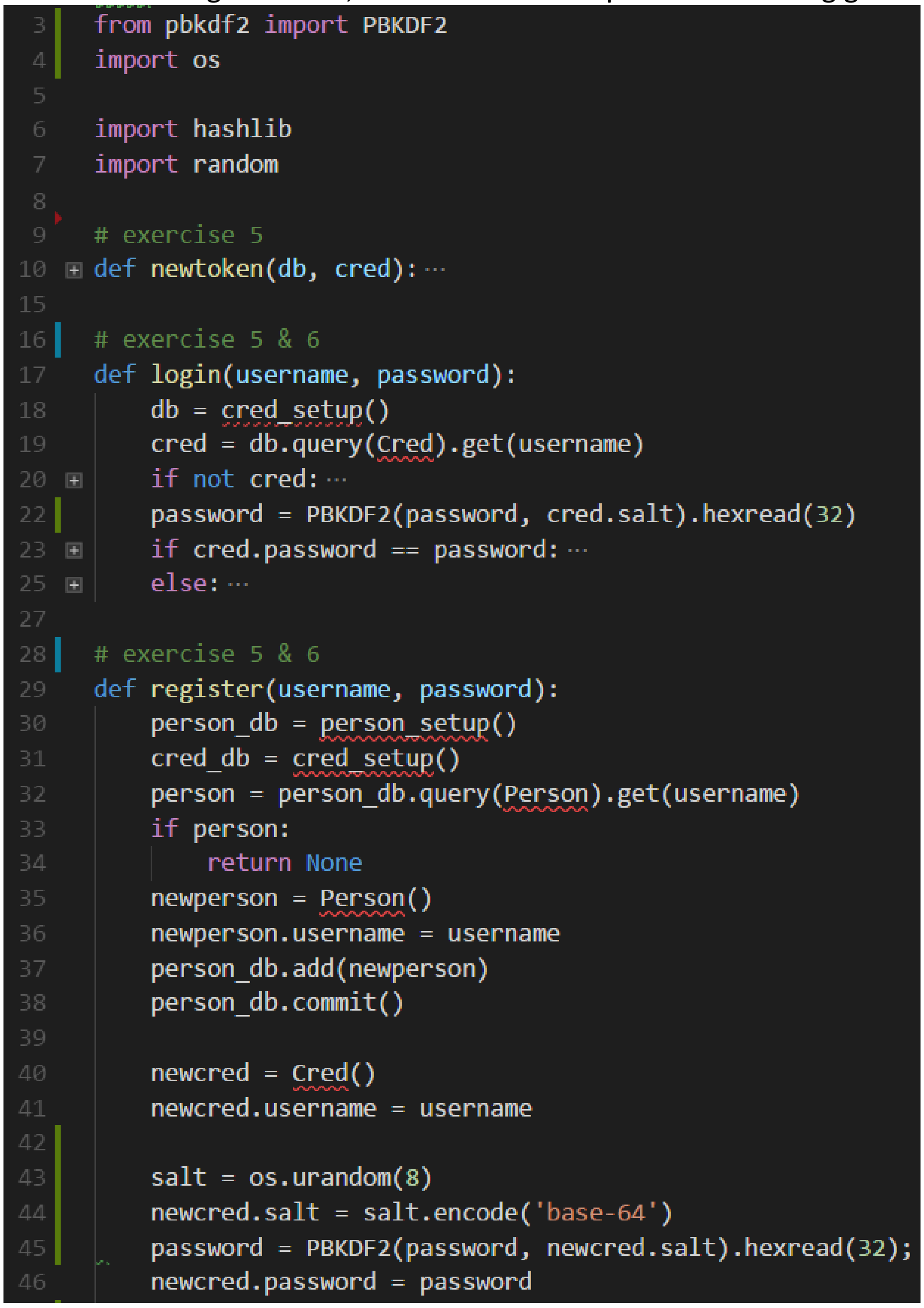
The Cred table is extended with a salt column as shown in Figure 14.



```python
class Cred(CredBase):
    __tablename__ = "cred"
    username = Column(String(128), primary_key=True)
    password = Column(String(128))
    salt = Column(String(128))
    token = Column(String(128))
```

Figure 14: Cred table extension in zoodb.py

Figure 15 shows how password hashing and salting is implemented. The password is hashed as shown in line 22 and 45 where the former occurs during login and the latter occurs during user registration. A 64-bit salt is generated in line 43. The salt is stored after undergoing base64 encoding in line 44, before the hashed password is being generated in line 45.



```python
 3  from pbkdf2 import PBKDF2
 4  import os
 5
 6  import hashlib
 7  import random
 8
 9  # exercise 5
10  def newtoken(db, cred):···
15
16  # exercise 5 & 6
17  def login(username, password):
18      db = cred_setup()
19      cred = db.query(Cred).get(username)
20      if not cred:···
22      password = PBKDF2(password, cred.salt).hexread(32)
23      if cred.password == password:···
25      else:···
27
28  # exercise 5 & 6
29  def register(username, password):
30      person_db = person_setup()
31      cred_db = cred_setup()
32      person = person_db.query(Person).get(username)
33      if person:
34          return None
35      newperson = Person()
36      newperson.username = username
37      person_db.add(newperson)
38      person_db.commit()
39
40      newcred = Cred()
41      newcred.username = username
42
43      salt = os.urandom(8)
44      newcred.salt = salt.encode('base-64')
45      password = PBKDF2(password, newcred.salt).hexread(32);
46      newcred.password = password
```

Figure 15: Modified auth.py

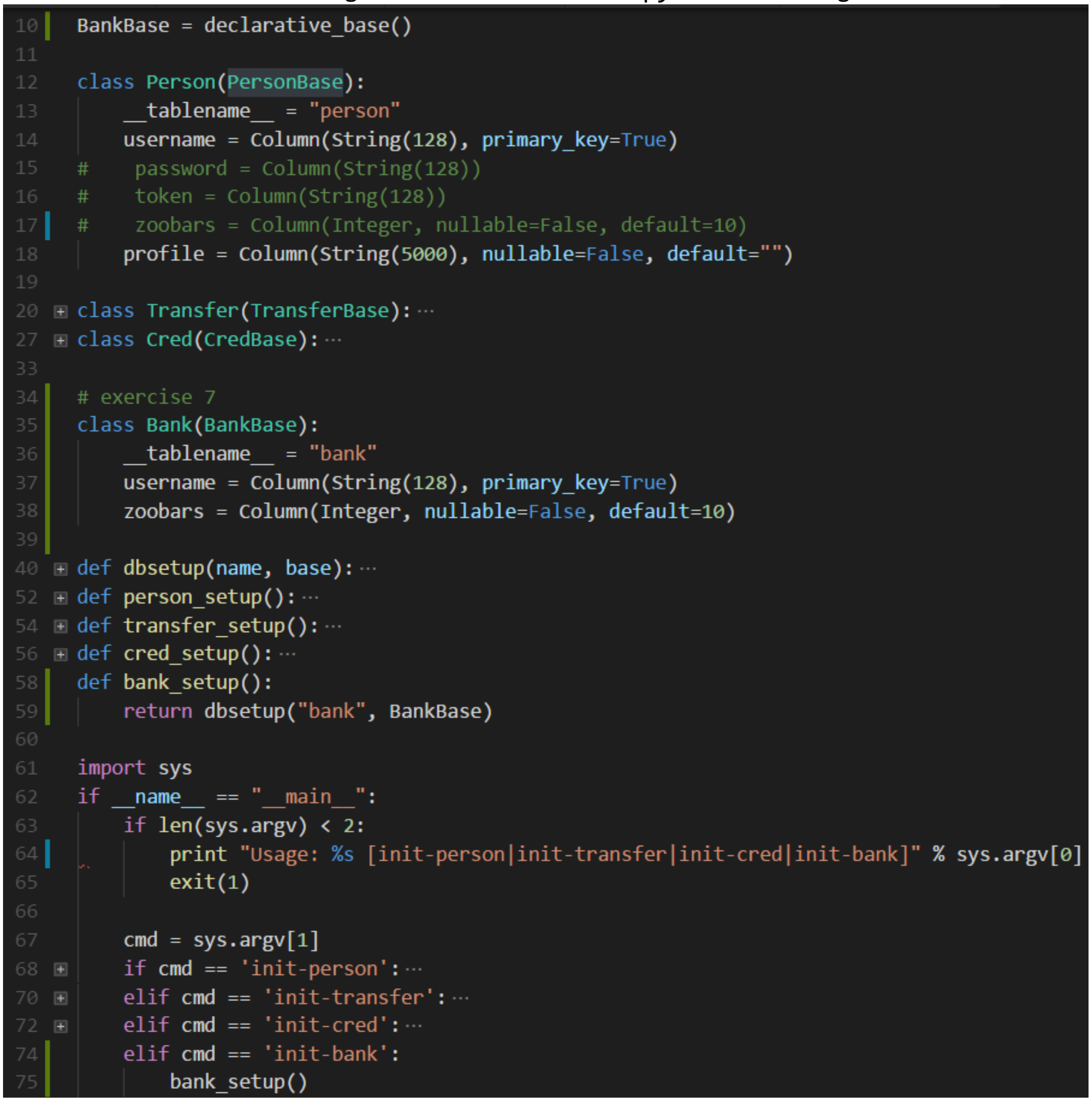
E) Privilege-separating the bank in Zoobar

Exercise 7:

I thought privilege-separating the login service in Zoobar was tedious, but privilege-separating the bank in Zoobar was even more tedious. 9 individual files needed to be modified:

1. `zoodb.py`

A new `Bank` database had to be created and the zoobars column was removed from the old `Person` database. These changes were made in `zoodb.py` as shown in Figure 16.

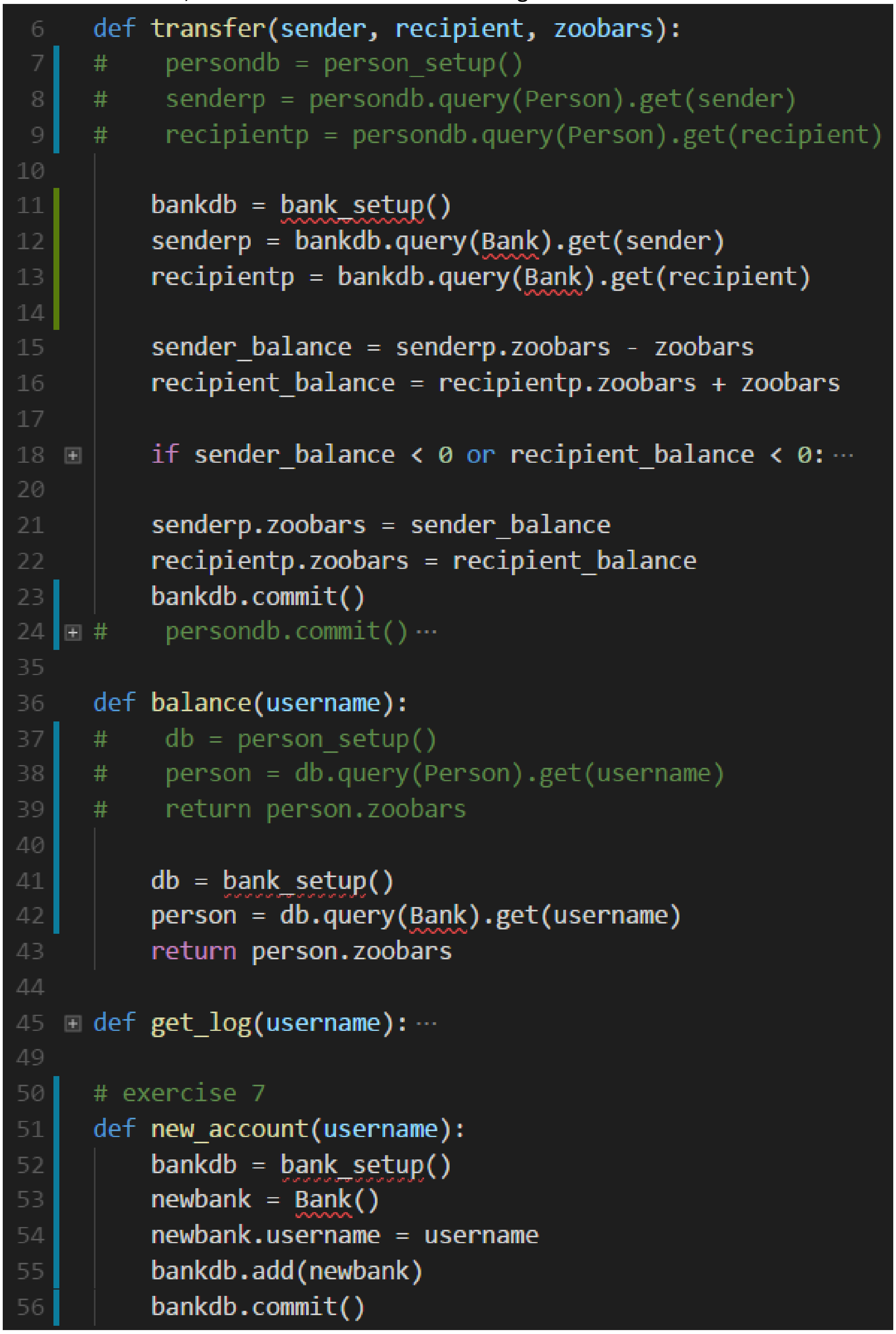


```python
BankBase = declarative_base()

class Person(PersonBase):
    __tablename__ = "person"
    username = Column(String(128), primary_key=True)
#    password = Column(String(128))
#    token = Column(String(128))
#    zoobars = Column(Integer, nullable=False, default=10)
    profile = Column(String(5000), nullable=False, default="")

class Transfer(TransferBase):...
class Cred(CredBase):...

# exercise 7
class Bank(BankBase):
    __tablename__ = "bank"
    username = Column(String(128), primary_key=True)
    zoobars = Column(Integer, nullable=False, default=10)

def dbsetup(name, base):...
def person_setup():...
def transfer_setup():...
def cred_setup():...
def bank_setup():
    return dbsetup("bank", BankBase)

import sys
if __name__ == "__main__":
    if len(sys.argv) < 2:
        print "Usage: %s [init-person|init-transfer|init-cred|init-bank]" % sys.argv[0]
        exit(1)

    cmd = sys.argv[1]
    if cmd == 'init-person':...
    elif cmd == 'init-transfer':...
    elif cmd == 'init-cred':...
    elif cmd == 'init-bank':
        bank_setup()
```

Figure 16: zoodb.py modifications

2. bank.py

auth.py was modified to have the appropriate information returned and stored in the correct databases, Cred or Person as shown in Figure 17.

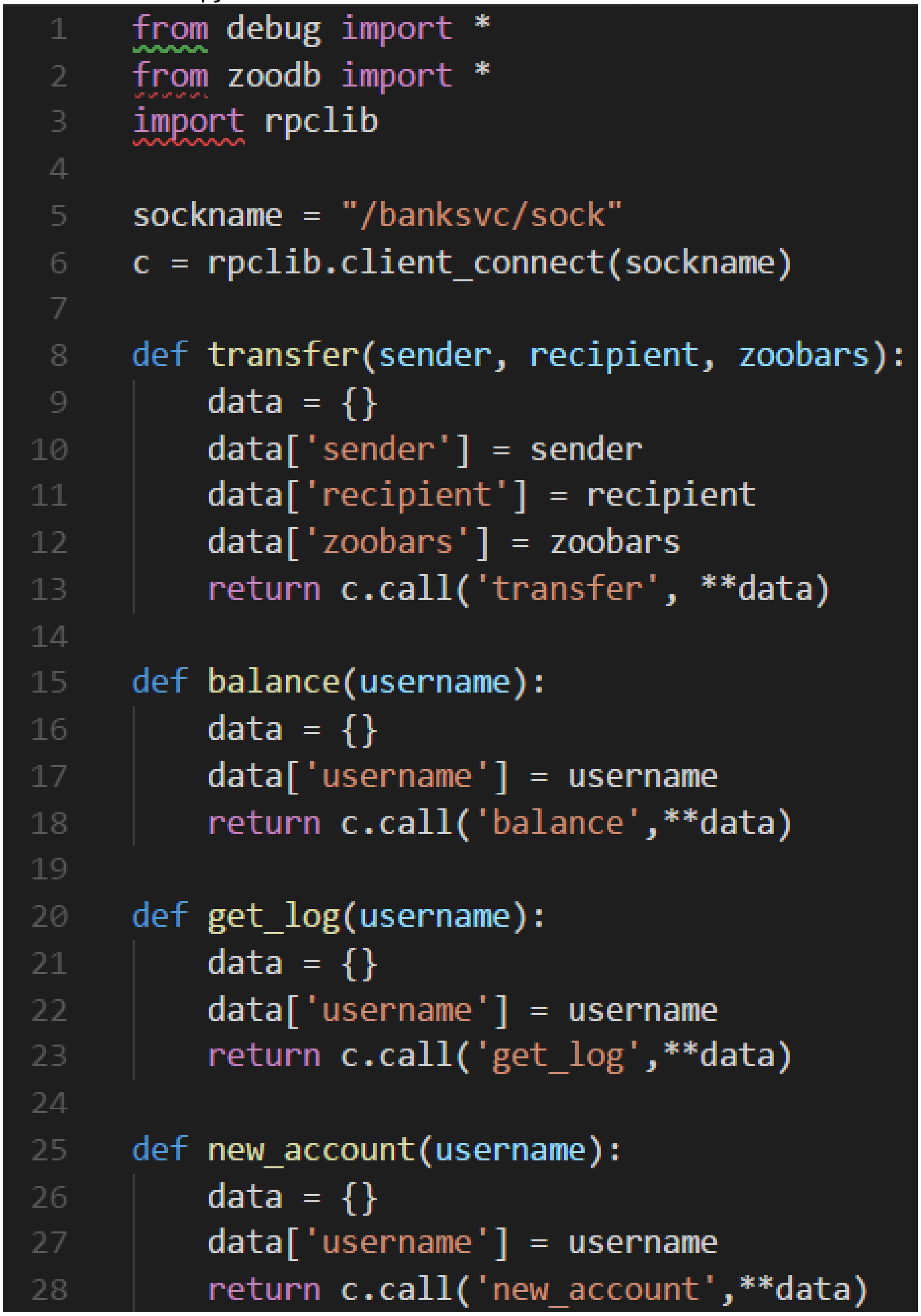


```
def transfer(sender, recipient, zoobars):
#    persondb = person_setup()
#    senderp = persondb.query(Person).get(sender)
#    recipientp = persondb.query(Person).get(recipient)

    bankdb = bank_setup()
    senderp = bankdb.query(Bank).get(sender)
    recipientp = bankdb.query(Bank).get(recipient)

    sender_balance = senderp.zoobars - zoobars
    recipient_balance = recipientp.zoobars + zoobars

    if sender_balance < 0 or recipient_balance < 0: ···

    senderp.zoobars = sender_balance
    recipientp.zoobars = recipient_balance
    bankdb.commit()
#    persondb.commit() ···

def balance(username):
#    db = person_setup()
#    person = db.query(Person).get(username)
#    return person.zoobars

    db = bank_setup()
    person = db.query(Bank).get(username)
    return person.zoobars

def get_log(username): ···

# exercise 7
def new_account(username):
    bankdb = bank_setup()
    newbank = Bank()
    newbank.username = username
    bankdb.add(newbank)
    bankdb.commit()
```

Figure 17: bank.py modifications

3. bank_client.py

A new bank_client.py was created with the RPC stubs for the client in zoobar/bank_client.py and completed as shown in Figure 18, using the existing functions in bank.py.

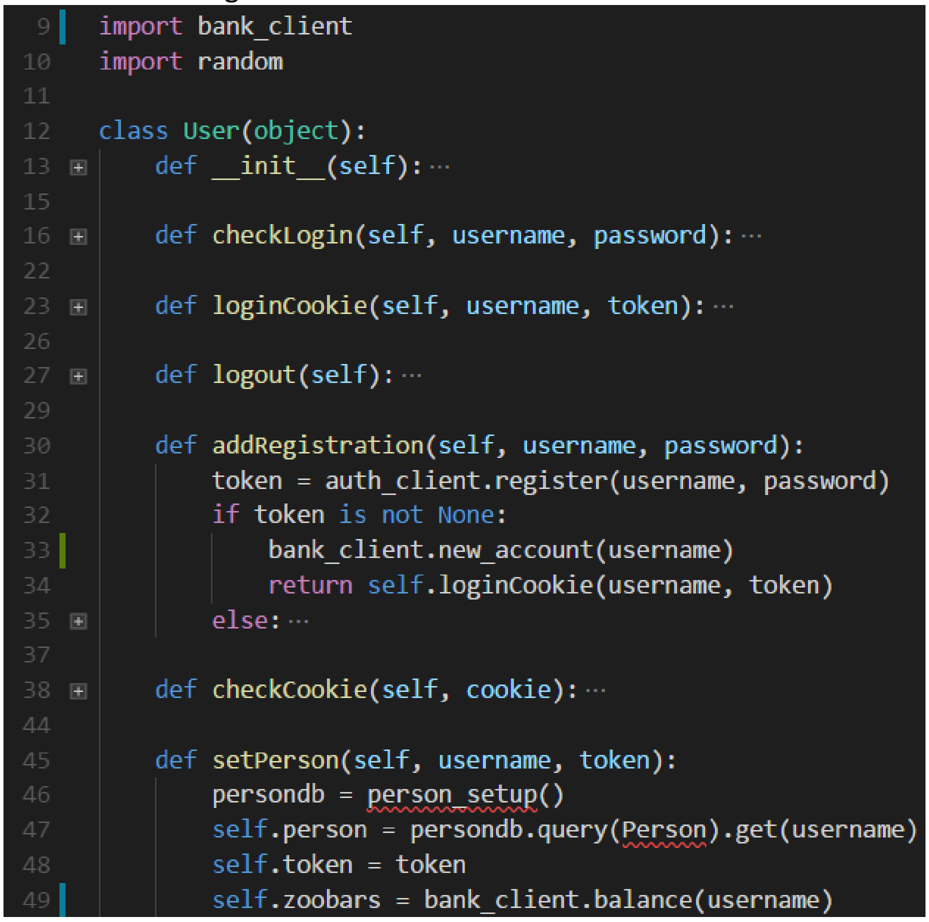


```python
from debug import *
from zoodb import *
import rpclib

sockname = "/banksvc/sock"
c = rpclib.client_connect(sockname)

def transfer(sender, recipient, zoobars):
    data = {}
    data['sender'] = sender
    data['recipient'] = recipient
    data['zoobars'] = zoobars
    return c.call('transfer', **data)

def balance(username):
    data = {}
    data['username'] = username
    return c.call('balance',**data)

def get_log(username):
    data = {}
    data['username'] = username
    return c.call('get_log',**data)

def new_account(username):
    data = {}
    data['username'] = username
    return c.call('new_account',**data)
```

Figure 18: bank_client.py creation

4. `login.py`

The login code in `login.py` was modified to invoke our new bank service instead of calling `bank.py` directly by importing `bank_client` and replacing the instance of bank in line 49 with `bank_client` instead as shown in Figure 19. We also enable the creation of a new account when a new user needs to get an initial 10 zoobars, invoking out bank service in line 33 as shown in Figure 19.

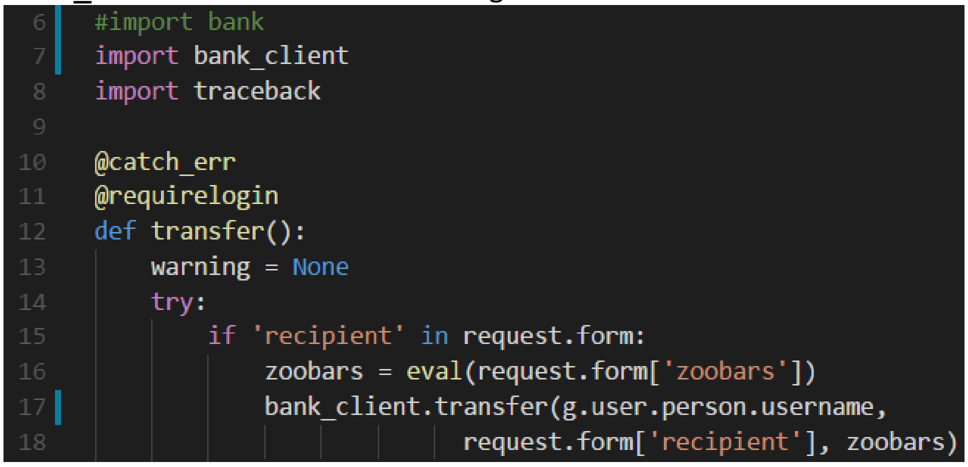


```python
 9  import bank_client
10  import random
11
12  class User(object):
13      def __init__(self):···
15
16      def checkLogin(self, username, password):···
22
23      def loginCookie(self, username, token):···
26
27      def logout(self):···
29
30      def addRegistration(self, username, password):
31          token = auth_client.register(username, password)
32          if token is not None:
33              bank_client.new_account(username)
34              return self.loginCookie(username, token)
35          else:···
37
38      def checkCookie(self, cookie):···
44
45      def setPerson(self, username, token):
46          persondb = person_setup()
47          self.person = persondb.query(Person).get(username)
48          self.token = token
49          self.zoobars = bank_client.balance(username)
```

Figure 19: login.py modifications

5. `transfer.py`

The transfer code in `transfer.py` was modified to invoke our new bank service instead of calling `bank.py` directly by importing `bank_client` and replacing all instances of bank with `bank_client` instead as shown in Figure 20.

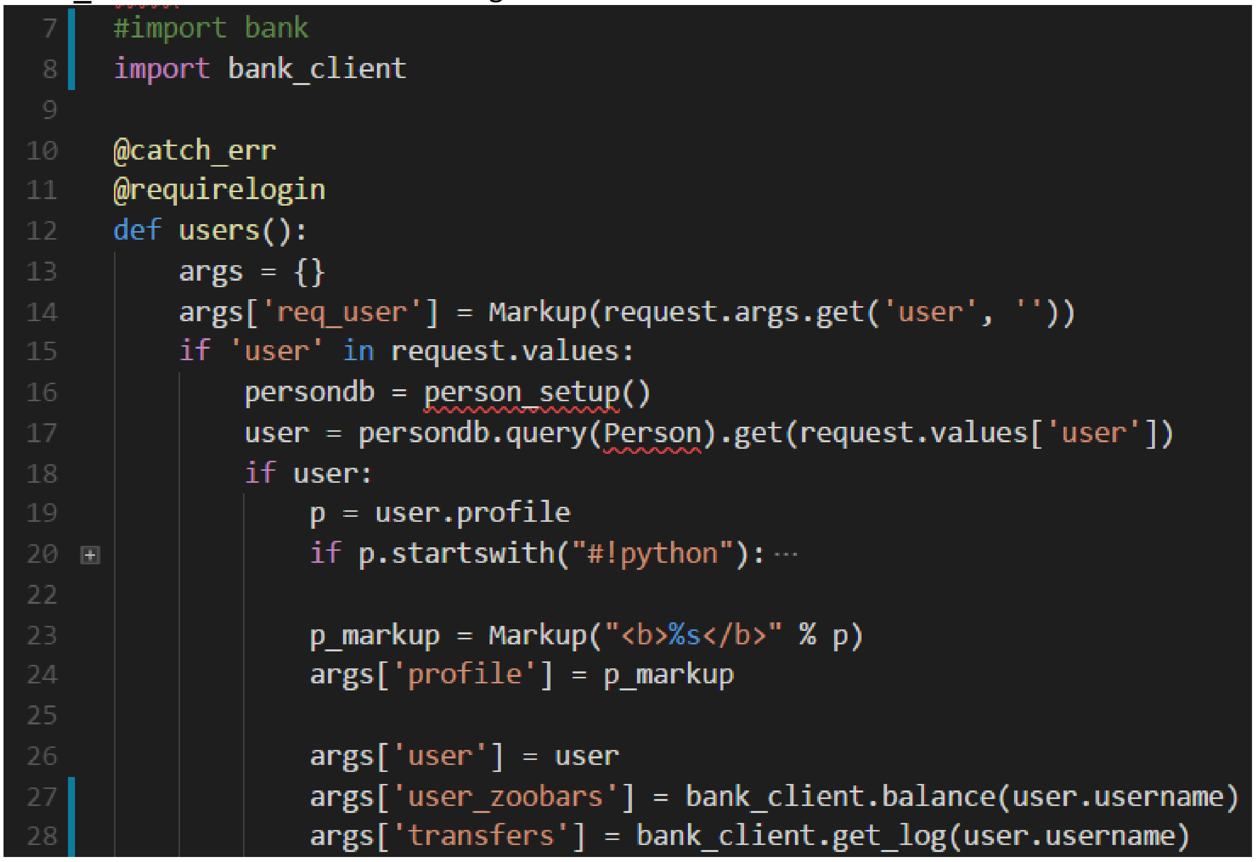


```python
#import bank
import bank_client
import traceback

@catch_err
@requirelogin
def transfer():
    warning = None
    try:
        if 'recipient' in request.form:
            zoobars = eval(request.form['zoobars'])
            bank_client.transfer(g.user.person.username,
                                 request.form['recipient'], zoobars)
```

Figure 20: transfer.py modifications

6. `users.py`

The users code in `users.py` was modified to invoke our new bank service instead of calling `bank.py` directly by importing `bank_client` and replacing all instances of bank with `bank_client` instead as shown in Figure 21.

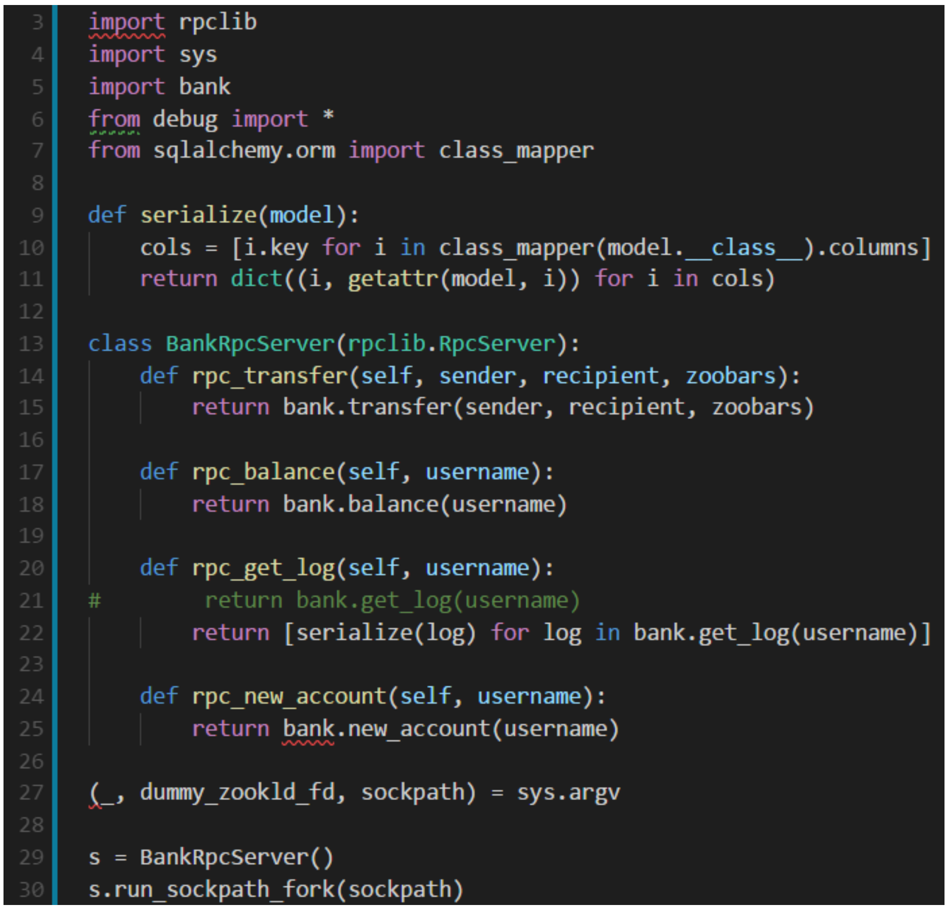


```python
#import bank
import bank_client

@catch_err
@requirelogin
def users():
    args = {}
    args['req_user'] = Markup(request.args.get('user', ''))
    if 'user' in request.values:
        persondb = person_setup()
        user = persondb.query(Person).get(request.values['user'])
        if user:
            p = user.profile
            if p.startswith("#!python"): ···

            p_markup = Markup("<b>%s</b>" % p)
            args['profile'] = p_markup

            args['user'] = user
            args['user_zoobars'] = bank_client.balance(user.username)
            args['transfers'] = bank_client.get_log(user.username)
```

Figure 21: users.py modifications

7. `bank-server.py`

The new `bank_svc` service for user authentication was created by replicating and modifying the initial file `zoobar/auth_server.py` to obtain the new `zoobar/bank_server.py` as shown in Figure 22. Once again, using the existing functions in `bank.py`. The get_log function requires additional formatting as the original SQLAlchemy query object is not JSON serializable. As such, the additional serialize method from lines 9 to 11 in Figure 22 was added to assist with that.

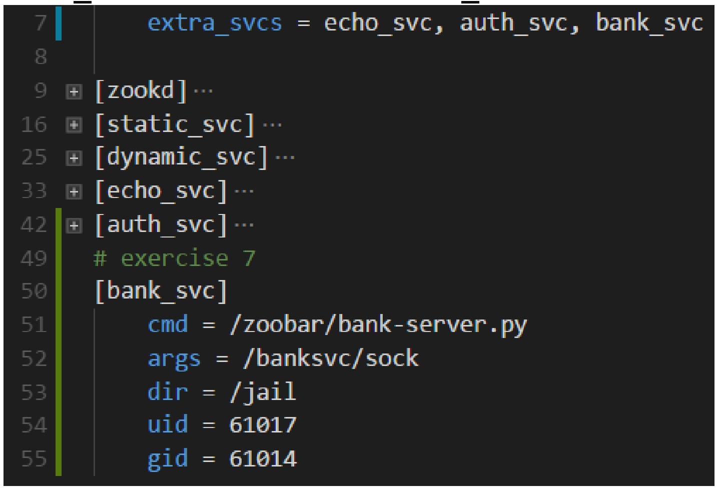


```
 3  import rpclib
 4  import sys
 5  import bank
 6  from debug import *
 7  from sqlalchemy.orm import class_mapper
 8  
 9  def serialize(model):
10      cols = [i.key for i in class_mapper(model.__class__).columns]
11      return dict((i, getattr(model, i)) for i in cols)
12  
13  class BankRpcServer(rpclib.RpcServer):
14      def rpc_transfer(self, sender, recipient, zoobars):
15          return bank.transfer(sender, recipient, zoobars)
16  
17      def rpc_balance(self, username):
18          return bank.balance(username)
19  
20      def rpc_get_log(self, username):
21  #         return bank.get_log(username)
22          return [serialize(log) for log in bank.get_log(username)]
23  
24      def rpc_new_account(self, username):
25          return bank.new_account(username)
26  
27  (_, dummy_zookld_fd, sockpath) = sys.argv
28  
29  s = BankRpcServer()
30  s.run_sockpath_fork(sockpath)
```

Figure 22: bank-server.py modifications

8. `zook.conf`

In order to start the `bank_server`, `zook.conf` was modified as shown in Figure 23. `bank_svc` was added as an `extra_svc` service to be run as shown in line 7.

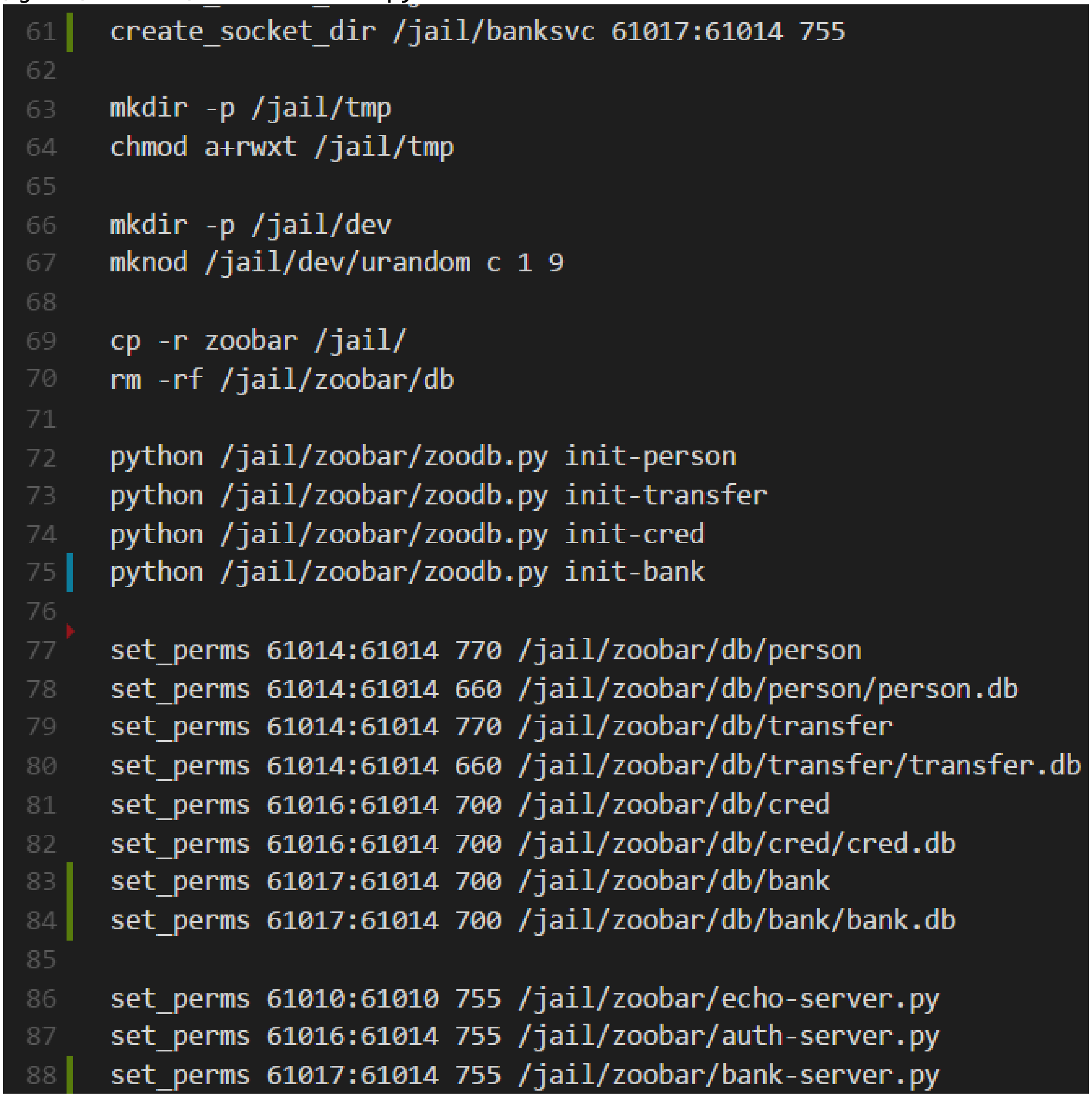


```
 7      extra_svcs = echo_svc, auth_svc, bank_svc
 8  
 9  [zookd]···
16  [static_svc]···
25  [dynamic_svc]···
33  [echo_svc]···
42  [auth_svc]···
49  # exercise 7
50  [bank_svc]
51      cmd = /zoobar/bank-server.py
52      args = /banksvc/sock
53      dir = /jail
54      uid = 61017
55      gid = 61014
```

Figure 23: zook.conf modifications

9. `chroot-setup.sh`

The `chroot-setup.sh` file was modified as shown in Figure 24. Line 61 was added to create the socket for the new `bank_svc`. Lines 83 to 84 were added to set the permissions on the Bank database appropriately, applying the execute permissions on the binary `/jail/zoobar/bank-server.py` as well in line 88.



```
61  create_socket_dir /jail/banksvc 61017:61014 755
62
63  mkdir -p /jail/tmp
64  chmod a+rwxt /jail/tmp
65
66  mkdir -p /jail/dev
67  mknod /jail/dev/urandom c 1 9
68
69  cp -r zoobar /jail/
70  rm -rf /jail/zoobar/db
71
72  python /jail/zoobar/zoodb.py init-person
73  python /jail/zoobar/zoodb.py init-transfer
74  python /jail/zoobar/zoodb.py init-cred
75  python /jail/zoobar/zoodb.py init-bank
76
77  set_perms 61014:61014 770 /jail/zoobar/db/person
78  set_perms 61014:61014 660 /jail/zoobar/db/person/person.db
79  set_perms 61014:61014 770 /jail/zoobar/db/transfer
80  set_perms 61014:61014 660 /jail/zoobar/db/transfer/transfer.db
81  set_perms 61016:61014 700 /jail/zoobar/db/cred
82  set_perms 61016:61014 700 /jail/zoobar/db/cred/cred.db
83  set_perms 61017:61014 700 /jail/zoobar/db/bank
84  set_perms 61017:61014 700 /jail/zoobar/db/bank/bank.db
85
86  set_perms 61010:61010 755 /jail/zoobar/echo-server.py
87  set_perms 61016:61014 755 /jail/zoobar/auth-server.py
88  set_perms 61017:61014 755 /jail/zoobar/bank-server.py
```

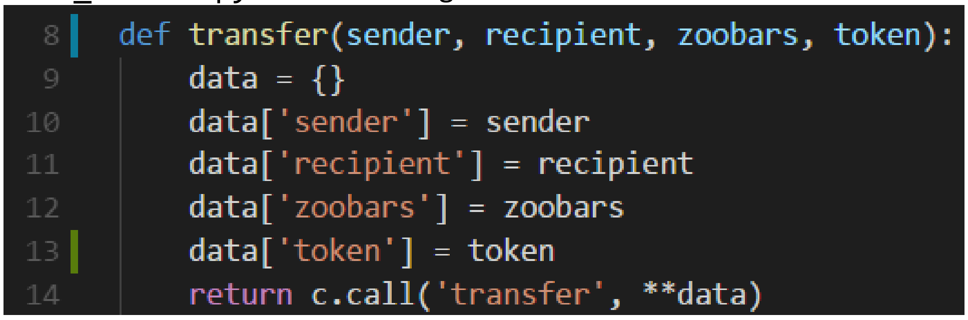
Figure 24: chroot-setup.sh modifications

After making the necessary modifications to each of these 9 files, we had to run `sudo make setup` again before running `sudo make check` to verify that we successfully completed the activity, successfully privilege-separating the bank service in Zoobar.

<u>Exercise 8:</u>
To add authentication to the transfer RPC in the bank service, we must modify four files.

1. `bank_client.py`

We must include an additional key-value pair which stores the user's token in `bank_client.py` as shown in Figure 25.

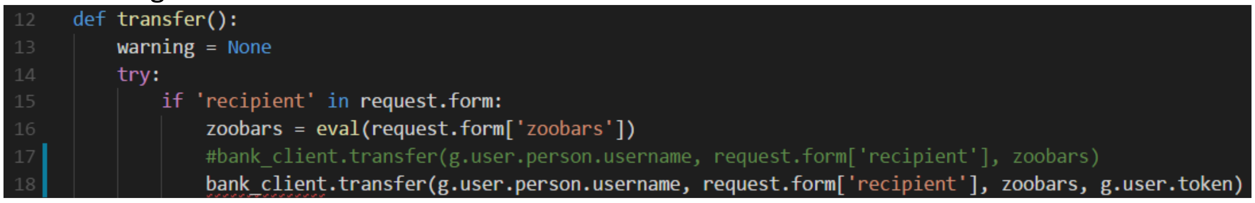


```python
def transfer(sender, recipient, zoobars, token):
    data = {}
    data['sender'] = sender
    data['recipient'] = recipient
    data['zoobars'] = zoobars
    data['token'] = token
    return c.call('transfer', **data)
```

Figure 25: bank_client.py modifications

2. `transfer.py`

We must also include the user token in `transfer.py`, retrieving it using `g.user.token` as shown in Figure 26.

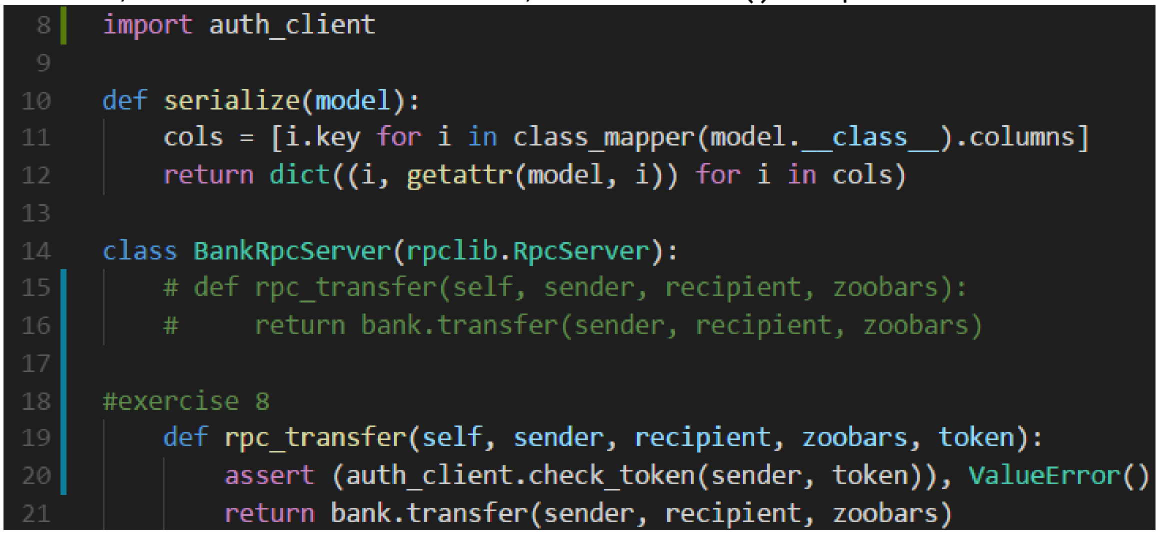


```python
def transfer():
    warning = None
    try:
        if 'recipient' in request.form:
            zoobars = eval(request.form['zoobars'])
            #bank_client.transfer(g.user.person.username, request.form['recipient'], zoobars)
            bank_client.transfer(g.user.person.username, request.form['recipient'], zoobars, g.user.token)
```

Figure 26: transfer.py modifications

3. `bank-server.py`

First, we must import `auth_client` in `bank-server.py` to allow us to add authentication to the transfer RPC in the bank server as shown in line 8 in Figure 27. Then, if a valid token is provided, the assert statement in line 20 will be true and the transfer will occur as per normal. However, if the assert statement is false, a `ValueError()` exception will be raised.



```python
 8  import auth_client
 9
10  def serialize(model):
11      cols = [i.key for i in class_mapper(model.__class__).columns]
12      return dict((i, getattr(model, i)) for i in cols)
13
14  class BankRpcServer(rpclib.RpcServer):
15      # def rpc_transfer(self, sender, recipient, zoobars):
16      #     return bank.transfer(sender, recipient, zoobars)
17
18  #exercise 8
19      def rpc_transfer(self, sender, recipient, zoobars, token):
20          assert (auth_client.check_token(sender, token)), ValueError()
21          return bank.transfer(sender, recipient, zoobars)
```

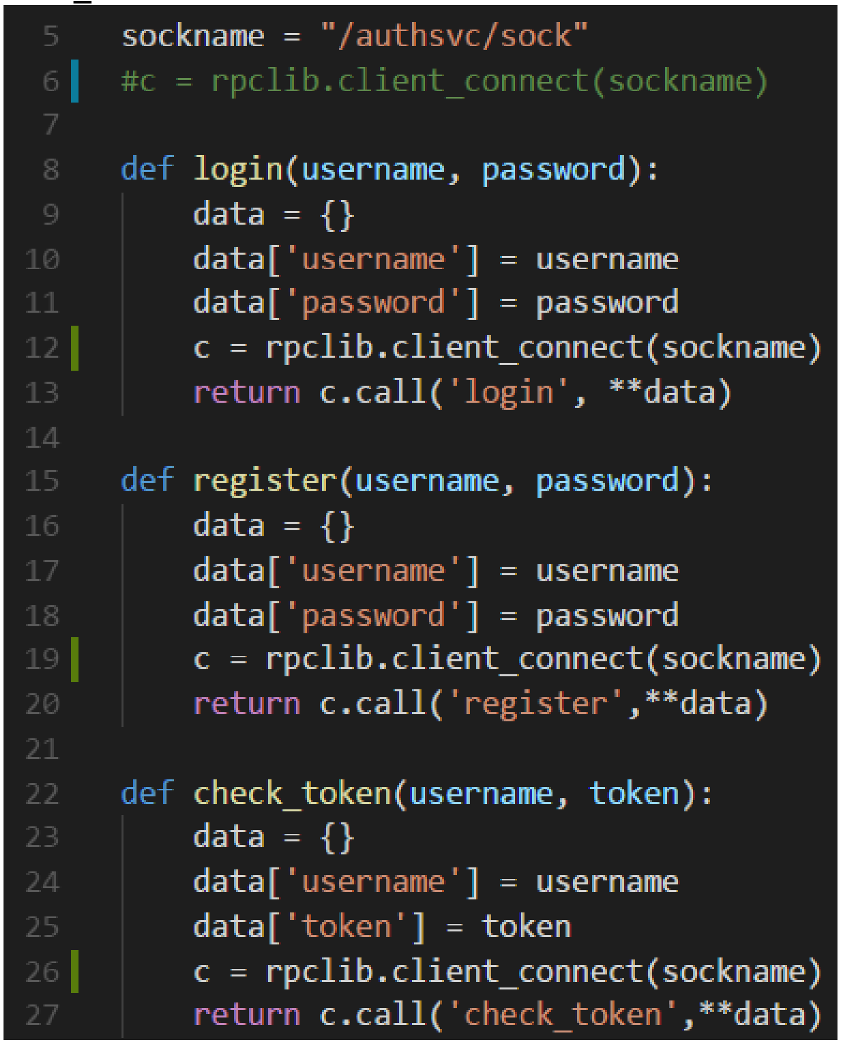
Figure 27: bank-server.py modifications

On hindsight, we did note that a better way to approach it could be to use:

```python
if not auth_client.check_token(sender, token):
      raise ValueError()
```

4. `auth_client.py`

We believe that here, we could be trying to connect to the socket before it has finished setting up which happens when you import `auth_client`, thus we need to move the `rpclib.client_connect` into the function itself instead of the main body of `auth_client`.

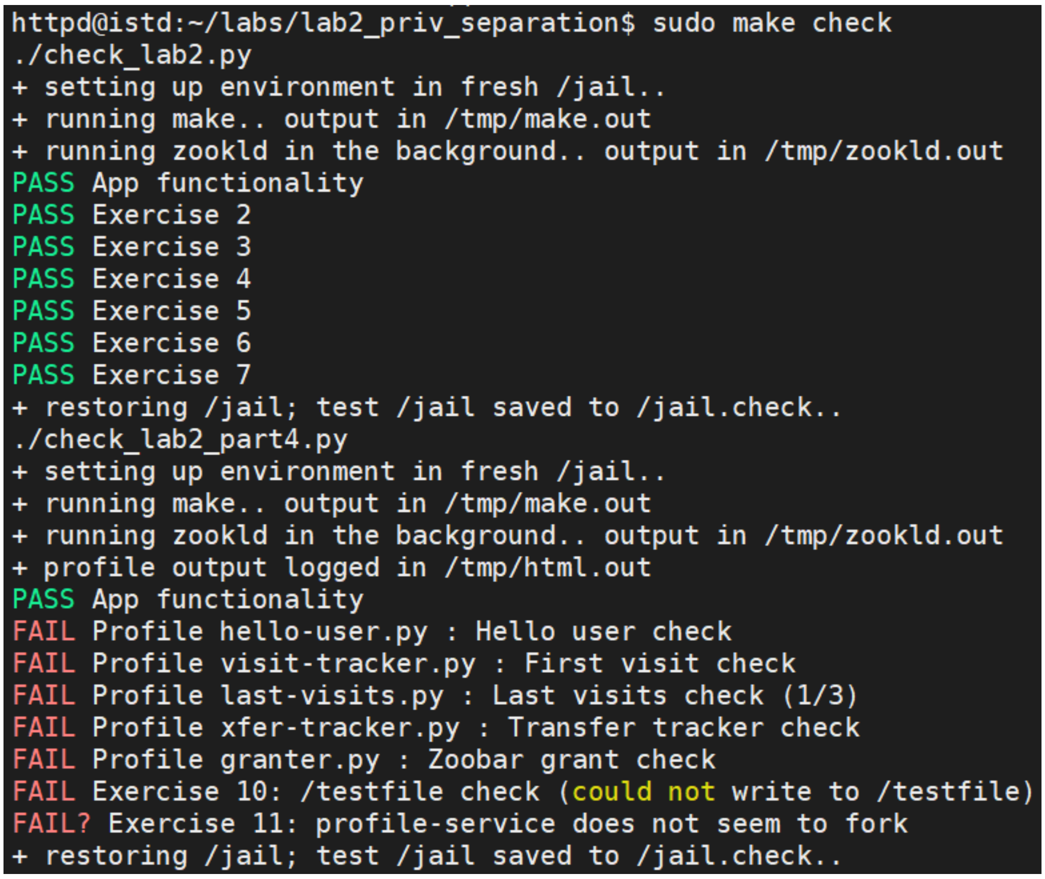


```python
sockname = "/authsvc/sock"
#c = rpclib.client_connect(sockname)

def login(username, password):
    data = {}
    data['username'] = username
    data['password'] = password
    c = rpclib.client_connect(sockname)
    return c.call('login', **data)

def register(username, password):
    data = {}
    data['username'] = username
    data['password'] = password
    c = rpclib.client_connect(sockname)
    return c.call('register',**data)

def check_token(username, token):
    data = {}
    data['username'] = username
    data['token'] = token
    c = rpclib.client_connect(sockname)
    return c.call('check_token',**data)
```

Figure 28: auth_client.py modifications

After making the necessary modifications to each of these 4 files, we had to run `sudo make setup` again before running `sudo make check` to verify that we successfully completed the activity, successfully adding authentication to the transfer RPC in the bank service.

Figure 29 below shows completion of the first 7 exercises.



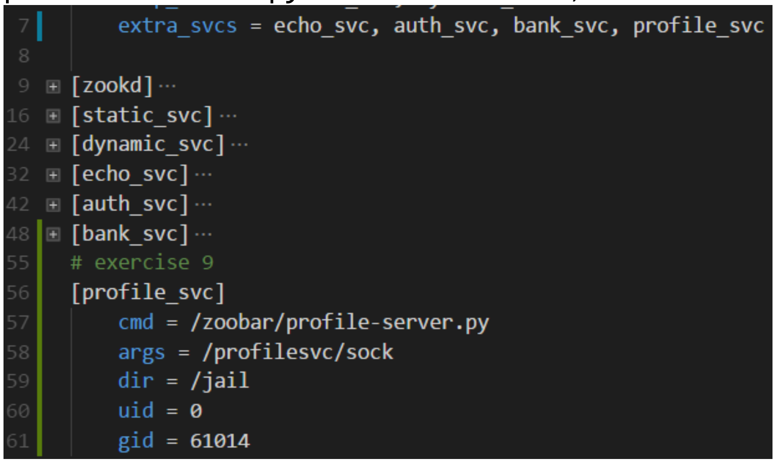
```
httpd@istd:~/labs/lab2_priv_separation$ sudo make check
./check_lab2.py
+ setting up environment in fresh /jail..
+ running make.. output in /tmp/make.out
+ running zookld in the background.. output in /tmp/zookld.out
PASS App functionality
PASS Exercise 2
PASS Exercise 3
PASS Exercise 4
PASS Exercise 5
PASS Exercise 6
PASS Exercise 7
+ restoring /jail; test /jail saved to /jail.check..
./check_lab2_part4.py
+ setting up environment in fresh /jail..
+ running make.. output in /tmp/make.out
+ running zookld in the background.. output in /tmp/zookld.out
+ profile output logged in /tmp/html.out
PASS App functionality
FAIL Profile hello-user.py : Hello user check
FAIL Profile visit-tracker.py : First visit check
FAIL Profile last-visits.py : Last visits check (1/3)
FAIL Profile xfer-tracker.py : Transfer tracker check
FAIL Profile granter.py : Zoobar grant check
FAIL Exercise 10: /testfile check (could not write to /testfile)
FAIL? Exercise 11: profile-service does not seem to fork
+ restoring /jail; test /jail saved to /jail.check..
```

Figure 29: sudo make check

E) Server-side sandboxing for executable profiles

Exercise 9:

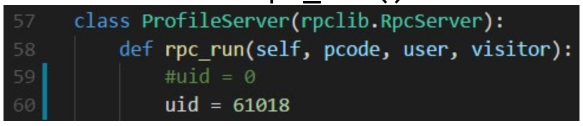
To begin, we add `profile-server.py` to our `zook.conf` as shown in Figure 30. Since the `profile-server.py` needs to run as root, its uid is 0 in its `zook.conf` entry.



```
    extra_svcs = echo_svc, auth_svc, bank_svc, profile_svc

[zookd]···
[static_svc]···
[dynamic_svc]···
[echo_svc]···
[auth_svc]···
[bank_svc]···
# exercise 9
[profile_svc]
    cmd = /zoobar/profile-server.py
    args = /profilesvc/sock
    dir = /jail
    uid = 0
    gid = 61014
```

Figure 30: zook.conf modifications

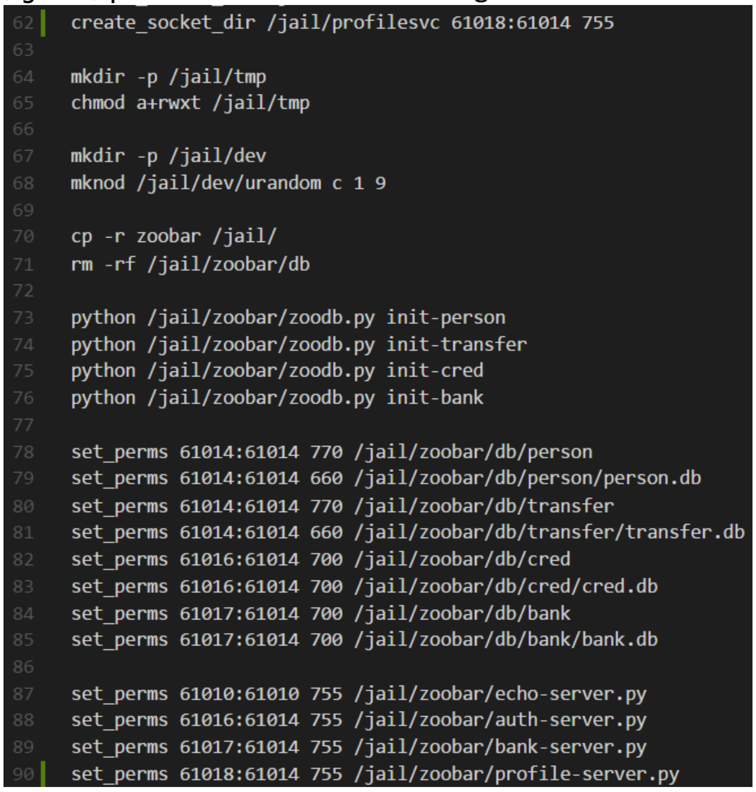
After we add `profile-server.py` to your web server, we change the uid value in `ProfileServer.rpc_run()` from 0 to 61018 as shown in Figure 31.



```
class ProfileServer(rpclib.RpcServer):
    def rpc_run(self, pcode, user, visitor):
        #uid = 0
        uid = 61018
```

Figure 31: profile-server.py modifications

Then, we modify `chroot-setup.sh` to create a directory for its socket, `/jail/profilesvc` as shown in Figure 32.

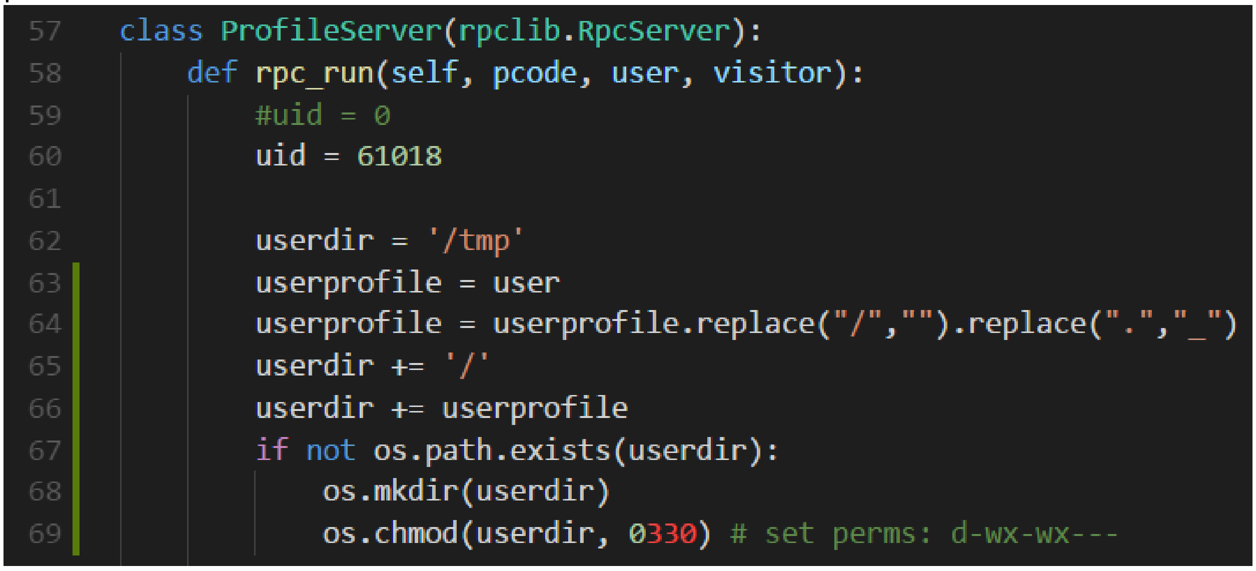


```
create_socket_dir /jail/profilesvc 61018:61014 755

mkdir -p /jail/tmp
chmod a+rwxt /jail/tmp

mkdir -p /jail/dev
mknod /jail/dev/urandom c 1 9

cp -r zoobar /jail/
rm -rf /jail/zoobar/db

python /jail/zoobar/zoodb.py init-person
python /jail/zoobar/zoodb.py init-transfer
python /jail/zoobar/zoodb.py init-cred
python /jail/zoobar/zoodb.py init-bank

set_perms 61014:61014 770 /jail/zoobar/db/person
set_perms 61014:61014 660 /jail/zoobar/db/person/person.db
set_perms 61014:61014 770 /jail/zoobar/db/transfer
set_perms 61014:61014 660 /jail/zoobar/db/transfer/transfer.db
set_perms 61016:61014 700 /jail/zoobar/db/cred
set_perms 61016:61014 700 /jail/zoobar/db/cred/cred.db
set_perms 61017:61014 700 /jail/zoobar/db/bank
set_perms 61017:61014 700 /jail/zoobar/db/bank/bank.db

set_perms 61010:61010 755 /jail/zoobar/echo-server.py
set_perms 61016:61014 755 /jail/zoobar/auth-server.py
set_perms 61017:61014 755 /jail/zoobar/bank-server.py
set_perms 61018:61014 755 /jail/zoobar/profile-server.py
```

Figure 32: chroot-setup.sh modifications

Exercise 10:
All of the user profiles currently run with access to the same files, because `ProfileServer.rpc_run()` sets `userdir` to `/tmp` and passes that as the directory to Sandbox (which chroots the profile code to that directory). As a result, one user's profile can corrupt the files stored by another user's profile. In order to ensure that each user's profile has access to its own files, and cannot tamper with the files of other user profiles, we can modify `rpc_run` in `profile-serve.py` as shown in Figure 33.

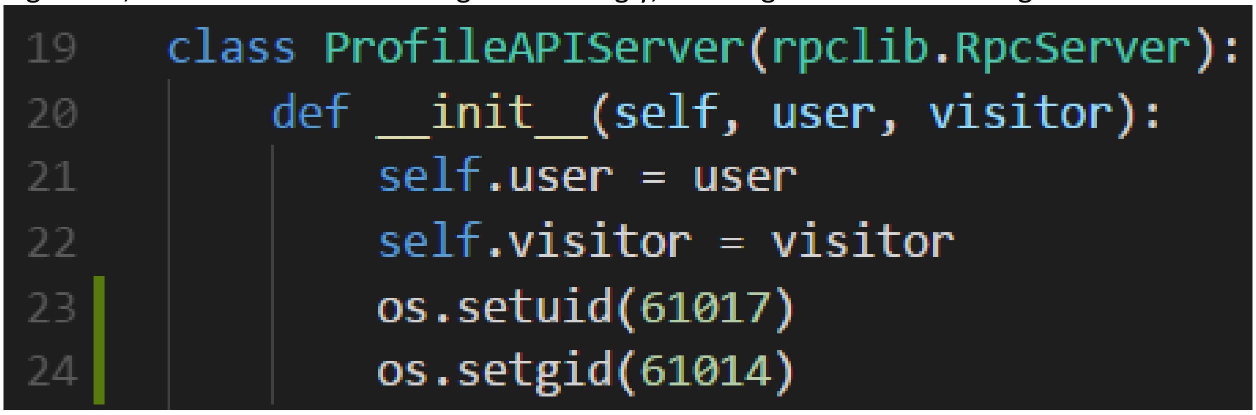
Each user will have a unique directory appended to the original "`/tmp`" as shown in line 66. To ensure that only the user and group can write and execute, the octal value of `0330` is passed to `chmod` in line 69.



```
57  class ProfileServer(rpclib.RpcServer):
58      def rpc_run(self, pcode, user, visitor):
59          #uid = 0
60          uid = 61018
61
62          userdir = '/tmp'
63          userprofile = user
64          userprofile = userprofile.replace("/","").replace(".","_")
65          userdir += '/'
66          userdir += userprofile
67          if not os.path.exists(userdir):
68              os.mkdir(userdir)
69              os.chmod(userdir, 0330) # set perms: d-wx-wx---
```

Figure 33: profile-serve.py modifications

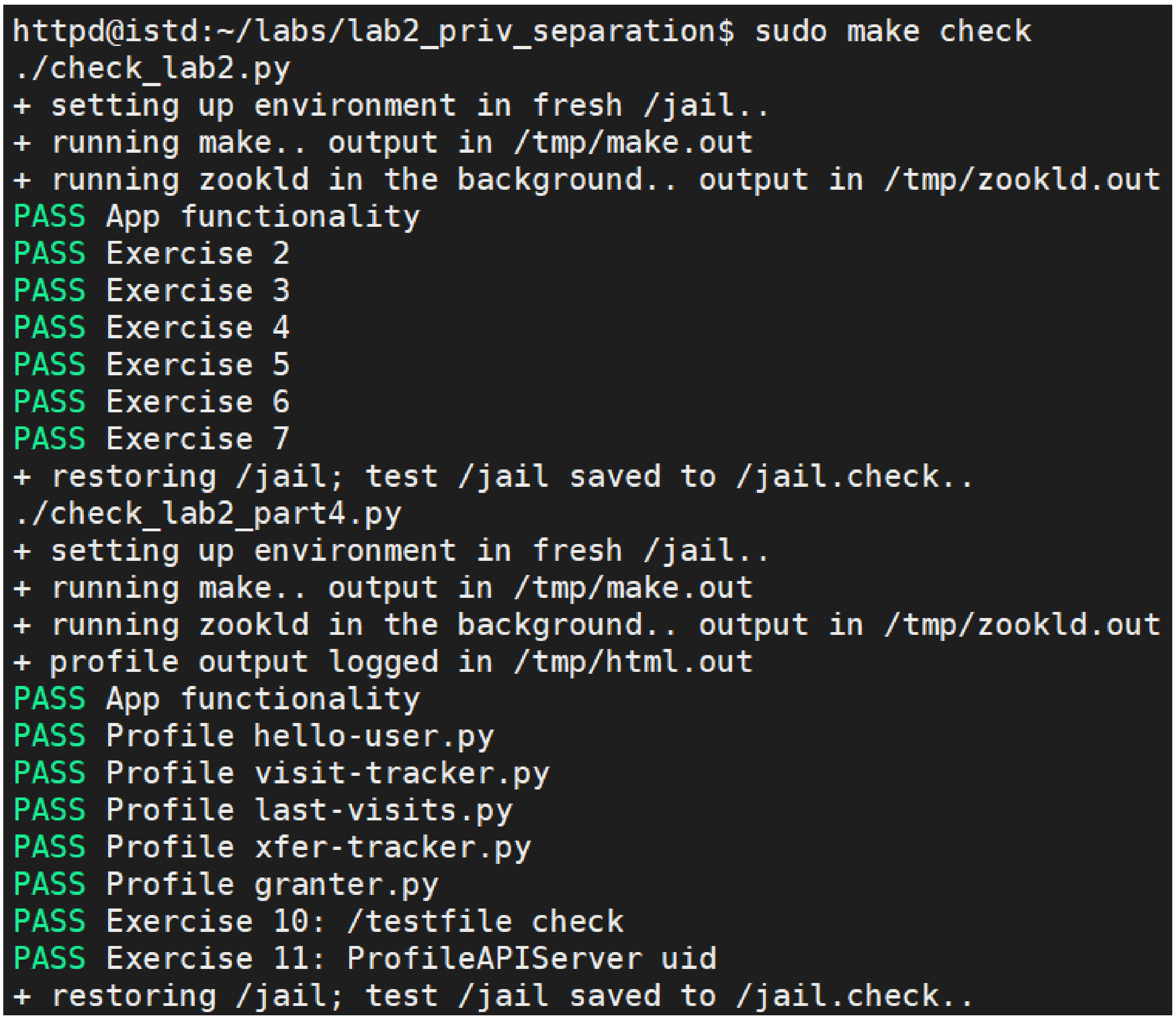
Exercise 11:
To change `ProfileAPIServer` in `profile-server.py` to avoid running as root and since `profile-server.py` forks off a separate child process to run `ProfileAPIServer`, we can switch to a different user ID and group ID in `ProfileAPIServer.__init__`. As shown in Figure 34, we have set the uid and gid accordingly, allowing us to avoid running as root.



```
class ProfileAPIServer(rpclib.RpcServer):
    def __init__(self, user, visitor):
        self.user = user
        self.visitor = visitor
        os.setuid(61017)
        os.setgid(61014)
```

Figure 34: profile-serve.py modifications

As shown in Figure 35, we cleared all the test cases after running `sudo make setup` and then `sudo make check`.



```
httpd@istd:~/labs/lab2_priv_separation$ sudo make check
./check_lab2.py
+ setting up environment in fresh /jail..
+ running make.. output in /tmp/make.out
+ running zookld in the background.. output in /tmp/zookld.out
PASS App functionality
PASS Exercise 2
PASS Exercise 3
PASS Exercise 4
PASS Exercise 5
PASS Exercise 6
PASS Exercise 7
+ restoring /jail; test /jail saved to /jail.check..
./check_lab2_part4.py
+ setting up environment in fresh /jail..
+ running make.. output in /tmp/make.out
+ running zookld in the background.. output in /tmp/zookld.out
+ profile output logged in /tmp/html.out
PASS App functionality
PASS Profile hello-user.py
PASS Profile visit-tracker.py
PASS Profile last-visits.py
PASS Profile xfer-tracker.py
PASS Profile granter.py
PASS Exercise 10: /testfile check
PASS Exercise 11: ProfileAPIServer uid
+ restoring /jail; test /jail saved to /jail.check..
```

Figure 35: sudo make check