System Security Lab 2

tags: `SUTD` `Security` `Lab`

Exercise 1

How It Works

- New users can register by supplying a username and password in the login page and are given 10 zoobars to begin with.
- In the "Home" tab, users can view their current balance of zoobars.
- In the "Transfer" tab, users can then transfer zoobars to other users by supplying the amount to be transferred and the username of the intended user.
- In the "Users" tab, users can search for other registered users to see their zoobar balance. A history of all the transactions the searched user has been involved in is displayed as well.

Interesting Observations

- Sending negative amounts (int or float) of zoobars will result in "theft", where zoobars are added into your balance and deducted from the target recipient's balance.

Exercise 2

First, we have to use `chroot` to create the jail. The jail should be at the directory specified by the configuration file, which is `/jail`, specified by the variable `dir`. After which, we have to use `chdir` to change the working directory of the program to the root of the jail, which is "/".

The following code is inserted at line 171 in `zookld.c`.

```
if ((dir = NCONF_get_string(conf, name, "dir")))
{
    /* chroot into dir */
    chroot(dir);
    chdir("/");
}
```

The code is verified as seen from the following screenshot.

```
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
```

Exercise 3

The system calls of `chroot`, `setgid`, and `setgroups` can only be called from privileged processes. If `setresuid` is called before any of these functions, they will not execute properly, which may lead to vulnerabilities. Hence, `setresuid` is ordered last in execution.

`chroot` is called first to jail the process before setting the relevant group IDs.

The order of running `setgid` and `setgroups` does not matter as long as they are executed before `setresuid`. This is because `setgid` sets both the real and effective group ID of the process to the new gid value while `setgroups` sets the process's supplementary group IDs, assuming the process executing this is privileged. (reference)

Finally, `setresuid` is called last to drop the privileges of the processes since no other processes after this requires privileged execution.

```
if ((dir = NCONF_get_string(conf, name, "dir")))
{
    /* chroot into dir */
    chroot(dir);
    chdir("/");
}

if (NCONF_get_number_e(conf, name, "gid", &gid))
{
    /* change real, effective, and saved gid to gid */
    setresgid(gid, gid, gid);
    warnx("setgid %ld", gid);
}

if ((groups = NCONF_get_string(conf, name, "extra_gids")))
{
    ngids = 0;
    CONF_parse_list(groups, ',', 1, &group_parse_cb, NULL);
    /* set the grouplist to gids */
    setgroups(ngids, gids);
    for (i = 0; i < ngids; i++)
        warnx("extra gid %d", gids[i]);
}


if (NCONF_get_number_e(conf, name, "uid", &uid))
{
    /* change real, effective, and saved uid to uid */
    setresuid(uid, uid, uid);
    warnx("setuid %ld", uid);
}
```

For non-root privileges, possible gids were identified to be 61011, 61012 or 61013 as commented in `zook.conf`:

```
# You can set supplementary groups with the extra_gids key.
# extra_gids = 61011, 61012, 61013
```

As `zookd` and `zookfs_svc` are separate services which should have separate permissions, they are assigned unique non-root uids and gids as shown below. Root uid or gid is 0.

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

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

As the contents of the databases `person.db` and `transfer.db` should be written to only by `zookfs_svc`, the uid and gid of relevant files in the database are set to be the same as `zookfs_svc`, which is 61012. This is as shown below. The permissions are set to 755 (rwxr-xr-x) such that non-root users are able to read and execute, but not write to it. This is expected behaviour as non-root users should only be able to read these sensitive data and not casually write to it.

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

However, this did not work and instead, crashed the 'check' script as shown below.

```
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
ERROR: Traceback (most recent call last):
  File "./check lab2.py", line 314, in main
    check ex0()
  File "./check lab2.py", line 300, in check ex0
    x = z client.check()
  File "/home/httpd/labs/lab2 priv separation/z client.py", line 84, in check
    html1, cookies1 = register("test1", "supersecretpassword")
  File "/home/httpd/labs/lab2 priv separation/z client.py", line 52, in register
    return login page("register", user, password)
  File "/home/httpd/labs/lab2 priv separation/z client.py", line 47, in login page
    postdata, "--keep-session-cookies"])
  File "/home/httpd/labs/lab2 priv separation/z client.py", line 22, in run wget
    raise Exception("wget failed: %s" % p.stderr.read())
Exception: wget failed: --2021-07-01 11:48:22--  http://localhost:8080/zoobar/index.cgi/login
Resolving localhost (localhost)... ::1, 127.0.0.1
Connecting to localhost (localhost)|::1|:8080... failed: Connection refused.
Connecting to localhost (localhost)|127.0.0.1|:8080... connected.
HTTP request sent, awaiting response... 500 INTERNAL SERVER ERROR
2021-07-01 11:48:23 ERROR 500: INTERNAL SERVER ERROR.



+ restoring /jail; test /jail saved to /jail.check..
Makefile:15: recipe for target 'check' failed
make: *** [check] Error 1
```

As such, the databases' directories' permissions were set as well.

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

The code is verified as seen from the following screenshot.

```
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
```

Exercise 4

For `dynamic_svc`: - The `url` value is set to "/zoobar/index.cgi.*" to pattern match exactly the path to `index.cgi` and the following files it serves. This restricts the access of `dynamic_svc` to only files accessible from the given `index.cgi` of the webpage. - The `uid` and `gid` values are set to 61012, consistent with Exercise 3's original `zookfs_svc` `uid` and `gid` values. - The `args` field is set to 61013, which is different from 61012 to ensure that the execution of `index.cgi` will not be using the same privileges as `uid=61012 gid=61012`

For `static_svc`: - The `url` value is set to "/zoobar/(media|templates)/.*.(html|css|jpg|js)" to pattern match exactly to the folders containing the allowable static files to be served, including their extensions. - The `uid` and `gid` values are set different from the same field values in `dynamic_svc` to separate their privileges. - The `args` field is set to 61015, different from the the `uid` and `gid` fields to ensure further privilege separation.

```
[dynamic_svc]
    cmd = zookfs
    url = /zoobar/index\.cgi.*
    uid = 61012
    gid = 61012
    dir = /jail
    args = 61013 61013

[static_svc]
    cmd = zookfs
    url = /zoobar/(media|templates)/.*\.(html|css|jpg|js)
    uid = 61014
    gid = 61014
    dir = /jail
    args = 61015 61015
```

The `http_svcs` variable in `zook.conf` is then amended to use the newly added configurations above.

```
http_svcs  = static_svc, dynamic_svc
```

The static service `static_svc`, should not be able to read database files. As such, the permissions for the database files for `static_svc` are set as follows:

```
set_perms 61014:61014 700 /jail/zoobar/db/person/person.db
set_perms 61014:61014 700 /jail/zoobar/db/transfer/transfer.db
set_perms 61014:61014 700 /jail/zoobar/db/person
set_perms 61014:61014 700 /jail/zoobar/db/transfer
```

`700` is set so that `static_svc` has no access to the database files at all. `61014` is both the uid and gid of the `static_svc`. For the `dynamic_svc`, it should be able to read the database files for its normal uid and gid of `61012`. This is set as shown below.

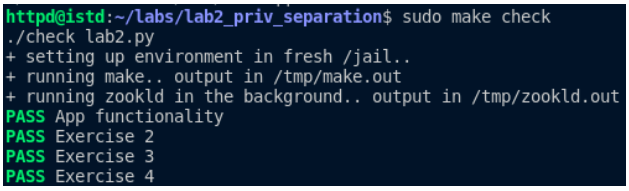
```
set_perms 61012:61012 744 /jail/zoobar/db/person/person.db
set_perms 61012:61012 744 /jail/zoobar/db/transfer/transfer.db
set_perms 61012:61012 755 /jail/zoobar/db/person
set_perms 61012:61012 755 /jail/zoobar/db/transfer
```

`744` is set so as to only allow read permissions for the database files for the normal process, as according to the The Principle of Least Privilege. `755` is set to allow both read and execute permissions for the database directories as there execute permission is required to access with contents within it.

As for the forked process for executing the `.cgi` files, executing privileges are required, as such `755` is set. The uid and gid for this is `61013`. As such, the permissions are set as follows:

```
set_perms 61013:61013 755 /jail/zoobar/index.cgi
```

The modifications made is verified as seen from the screenshot below:



Exercise 5

> Decide what interface your authentication service should provide (i.e., what functions it will run for clients). Look at the code in `login.py` and `auth.py`, and decide what needs to run in the authentication service, and what can run in the client (i.e., be part of the rest of the zoobar code). Keep in mind that your goal is to protect both passwords and tokens. We have provided initial RPC stubs for the client in the file `zoobar/auth_client.py`.

The authentication service should provide an interface which includes the functions of `login`, `register` and `check_token`. These functions should take in the relevant arguments and send it through the RPC socket accordingly with the relevant keys and values. As such, the `zoobar/auth_client.py` should be set up as follows:

```
sockname = "/authsvc/sock"

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

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

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

> Create a new `auth_svc` service for user authentication, along the lines of `echo-server.py`. We have provided an initial file for you, `zoobar/auth-server.py`, which you should modify for this purpose. The implementation of this service should use the existing functions in `auth.py`.

As for the `zoobar/auth-server.py`, it needs to be able to receive the function calls as the client as well, which are `login`, `register` and `check_token`. Based on the specifications of the RpcServer class, the functions have to be preceded with 'rpc_'. This is specified in `rpclib.py` as seen below:

```
class RpcServer(object):
    def run_sock(self, sock):
        lines = buffered_readlines(sock)
        for req in lines:
            (method, kwargs) = parse_req(req)
            m = self.__getattribute__('rpc_' + method)
            ret = m(**kwargs)
            sock.sendall(format_resp(ret) + '\n')
```

As such, `AuthRpcServer` in `zoobar/auth-server.py` is defined as shown below.

```
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)
```

> Modify `zook.conf` to start the `auth-server` appropriately (under a different UID).

A new entry for `auth-svc` is initialised in `zook.conf` with the following values: - the cmd value will point to the `auth-server.py` code that will be executed - args will be set to the path to the socket that `auth_client.py` will be listening to, which should be the same value as the `sockname` defined in `auth_client.py`. - uid value should be unique - gid value is set to the same value as the gid of `dynamic_svc` as `dynamic_svc` will utilise the auth service when it serves the login page from `index.cgi`

```
[auth_svc]
    cmd = /zoobar/auth-server.py
    args = /authsvc/sock
    dir = /jail
    uid = 61016
    gid = 61012
```

> Split the user credentials (i.e., passwords and tokens) from the `Person` database into a separate `Cred` database, stored in `/zoobar/db/cred`. Don't keep any passwords or tokens in the old `Person` database.

A new database named `Cred` is first initialised in `zoodb.py`. It should contain username, password, and token field, while the `Person` database should not have the password and token field.

```
CredBase = declarative_base()

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

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

...

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)
...
    elif cmd == 'init-cred':
        cred_setup()
```

After which in `zoobar/auth.py`, code is modified to access `Person` and `Cred` databases separately.

```
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):
    person_db = person_setup()
    cred_db = cred_setup()
    person = person_db.query(Person).get(username)
    cred = cred_db.query(Cred).get(username)
    if not person or not cred:
        return None
    if cred.password == password:
        return newtoken(cred_db, cred)
    else:
        return None

def register(username, password):
    person_db = person_setup()
    cred_db = cred_setup()
    person = person_db.query(Person).get(username)
    cred = cred_db.query(Cred).get(username)
    if person or cred:
        return None

    newcred = Cred()
    newcred.username = username
    newcred.password = password
    cred_db.add(newcred)
    cred_db.commit()

    newperson = Person()
    newperson.username = username
    person_db.add(newperson)
    person_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
```

> Modify `chroot-setup.sh` to set permissions on the `cred` database appropriately, and to create the socket for the auth service.

First, a socket directory for `auth_svc` is created in the directory `/jail/authsvc`. This is created with the uid and gid of `authavc`, which are `61016` and `61012` respectively. As read and execute permissions are required, the `755` flag is used.

```
create_socket_dir /jail/authsvc 61016:61012 755
```

Next, `Cred` database is initialised with the following line, similar to how the `Person` and `Transfer` databases are initialised.

```
python /jail/zoobar/zoodb.py init-cred
```

For the `Person` and `Transfer` databases, the group permissions need to be increased as write permissions are required for this exercise. As such, the group permission in the permission flag is increased by 2 to `764` and `775` respectively for the database files and directories respectively.

```
set_perms 61012:61012 764 /jail/zoobar/db/person/person.db
set_perms 61012:61012 764 /jail/zoobar/db/transfer/transfer.db
set_perms 61012:61012 775 /jail/zoobar/db/person
set_perms 61012:61012 775 /jail/zoobar/db/transfer
```

To reduce the permissions for `Cred` database as much as possible, `700` is set, meaning only root is able to read, write or execute it.

```
set_perms 61016:61012 700 /jail/zoobar/db/cred/cred.db
set_perms 61016:61012 700 /jail/zoobar/db/cred
```

As for `/jail/zoobar/echo-server.py` and `/jail/zoobar/auth-server.py`, root should be able to have full permissions while other users should have read and execute permissions. As such, `755` flag is set as shown below.

```
set_perms 61010:61010 755 /jail/zoobar/echo-server.py
set_perms 61016:61012 755 /jail/zoobar/auth-server.py
```

> Modify the login code in `login.py` to invoke your auth service instead of calling `auth.py` directly.

Replace the invocations of `auth` in `login.py` with `auth_client`. For instance, `import auth` is changed to `import auth_client`, and every call to `auth`'s function is changed to `auth_client`'s functions.

The modifications made is verified as seen from the successful check below.

```
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
```

Exercise 6

A new `salt` field is initialised in the `Cred` class to store the salt value.

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

Some functions in `auth.py` is updated as well, particularly for those that takes in `password` as a paramater: - `login`: The password parameter is transformed by hashing it with the existing salt value stored in the `Cred` database, before performing the password check. - `register`: On registering a new user, a random salt value of arbitrary length 47 is initialised and stored in the salt field of the `Cred` database for the new user. The user's password is also transformed by hashing it with the newly initialised salt value and storing the hashed version in the password field.

```
from pbkdf2 import PBKDF2

def login(username, password):
    person_db = person_setup()
    cred_db = cred_setup()
    person = person_db.query(Person).get(username)
    cred = cred_db.query(Cred).get(username)
    if not person or not cred:
        return None

    # transform the password
    password = PBKDF2(password, cred.salt).hexread(32)
    if cred.password == password:
        return newtoken(cred_db, cred)
    else:
        return None

def register(username, password):
    person_db = person_setup()
    cred_db = cred_setup()
    person = person_db.query(Person).get(username)
    cred = cred_db.query(Cred).get(username)
    if person or cred:
        return None

    newcred = Cred()
    newcred.username = username
    newcred.salt = os.urandom(47).encode('base-64') # length of salt
    newcred.password = PBKDF2(password, newcred.salt).hexread(32)
    cred_db.add(newcred)
    cred_db.commit()

    newperson = Person()
    newperson.username = username
    person_db.add(newperson)
    person_db.commit()

    return newtoken(cred_db, newcred)
```

The hashing implementation is verified as seen from the screenshot below:

```
httpd@istd:~/labs$ 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
```

Exercise 7

> You will need to split the `zoobar` balance information into a separate `Bank` database (in `zoodb.py`); implement the bank server by modifying `bank-server.py`; add the bank service to `zook.conf`; modify `chroot-setup.sh` to create the new `Bank` database and the socket for the bank service, and to set permissions on both the new `Bank` and the existing `Transfer` databases accordingly; create client RPC stubs for invoking the bank service; and modify the rest of the application code to invoke the RPC stubs instead of calling `bank.py`'s functions directly.

In `zoodb.py`, the `Bank` database, is defined to hold each `username` and the respective `zoobars`. On the other hand, `zoobars` is removed from the

`Person` database. The rest of the code to set up the database is similar to that of the `Cred` database. As such, modifications to `zoodb.py` are made as seen below.

```
BankBase = declarative_base()

...

class Person(PersonBase):
    __tablename__ = "person"
    username = Column(String(128), primary_key=True)
    profile = Column(String(5000), nullable=False, default="")

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

...

def bank_setup():
    return dbsetup("bank", BankBase)

import sys
if __name__ == "__main__":
    ...
    elif cmd == 'init-bank':
        bank_setup()
```

In `bank-server.py`, the server-side function calls through RPC are set up as follows.

```
class BankRpcServer(rpclib.RpcServer):
    def rpc_transfer(self, sender, recipient, zoobars):
        return bank.transfer(sender, recipient, zoobars)

    def rpc_balance(self, username):
        return bank.balance(username)

    def rpc_get_log(self, username):
        logs = []
        for row in bank.get_log(username):
            row_dict = row.__dict__
            row_dict.pop('_sa_instance_state', None)
            logs.append(row_dict)
        return logs
```

Note that as `bank.get_log(username)` returns a query, serialisation needs to be done to return the results of the query in a JSON serialisable form. This is done in the code above.

In `zook.conf`, the `bank_svc` is defined and added to `zook` as follows.

```
[zook]
    ...
    extra_svcs = echo_svc, auth_svc, bank_svc

[bank_svc]
    cmd = /zoobar/bank-server.py
    args = /banksvc/sock
    dir = /jail
    uid = 61017
    gid = 61012
```

In the `chroot-setup.sh`, the following lines are added as well, similar to the lines added for `cred`.

```
create_socket_dir /jail/banksvc 61017:61012 755

python /jail/zoobar/zoodb.py init-bank

set_perms 61017:61012 755 /jail/zoobar/bank-server.py

set_perms 61017:61012 700 /jail/zoobar/db/bank/bank.db
set_perms 61017:61012 700 /jail/zoobar/db/bank
```

`bank_client.py` is created to send requests through RPC as shown below.

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

sockname = "/banksvc/sock"

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

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

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

After the above modifications were made, invocations of `bank` should be changed to `bank_client`. These occur in the following files: `login.py`, `profile-server.py`, `transfer.py` and `users.py`.

> Don't forget to handle the case of account creation, when the new user needs to get an initial 10 zoobars. This may require you to change the interface of the bank service.

In the `bank.py` script, the following `new_account` function is defined to create the new bank entry for new users when they register.

```
def new_account(username):
    bank_db = bank_setup()
    newbank = Bank()
    newbank.username = username
    bank_db.add(newbank)
    bank_db.commit()
```

On the server defined by `bank-server.py`, the above function is called when "new_account" is requested through RPC.

```
def rpc_new_account(self, username):
    return bank.new_account(username)
```

In `bank_client.py`, `new_account` is defined as follows.

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

Immediately after the users has been successfully registered, the bank entry should be created. Thus, in `login.py`, `bank_client.new_account(username)` is called is the registration is successful and a user token is obtained.

```
def addRegistration(self, username, password):
    token = auth_client.register(username, password)
    if token:
        bank_client.new_account(username)
```

> Run **sudo make check** to verify that your privilege-separated bank service passes our tests.

```
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
```

Exercise 8

> Add authentication to the transfer RPC in the bank service.

A new parameter called `sender_input_token` is included into the `rpc_transfer` function in `bank-server.py` which will be verified using `auth_client.py`'s `check_token` function, which already has the appropriate permissions to access the `Cred` database.

```
import auth_client

class BankRpcServer(rpclib.RpcServer):
    def rpc_transfer(self, sender, recipient, zoobars, sender_input_token):
        if not auth_client.check_token(sender, sender_input_token):
            raise ValueError("Invalid token. IMPOSTER!!")
```

`bank_client.py` will have to supply the `sender_input_token` value, hence a new data field is added.

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

Finally, the `bank_client.transfer` call in `transfer.py` will have to be updated to take in the current logged in user's token value by using `g.user.token`.

```
try:
    if 'recipient' in request.form:
        zoobars = eval(request.form['zoobars'])
        bank_client.transfer(g.user.person.username,
                             request.form['recipient'], zoobars, g.user.token)
        warning = "Sent %d zoobars" % zoobars
```

> Although **make check** does not include an explicit test for this exercise, you should be able to check whether this feature is working or not by manually connecting to your transfer service and verifying that it is not possible to perform a transfer without supplying a valid token.

To test the authentication feature added, a new file `test_ex_8.py` is created to send a transfer request of 10 zoobars from "user1" to "user2" with the token to be used to be passed as an argument.

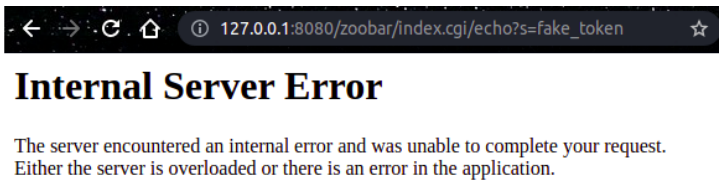
```
import rpclib

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

def transfer_with_token(token):
    data = {}
    data['sender'] = "user1"
    data['recipient'] = "user2"
    data['zoobars'] = 10
    data['sender_input_token'] = token
    return sock.call('transfer', **data)
```

The echo service is used to perform the request, with the `s` argument passed in to the `token` parameter. As such, in `echo-server.py`, the prepared function is called as shown below.

```
def rpc_echo(self, s):
    from test_ex_8 import transfer_with_token
    transfer_with_token(s)
    return 'You said: %s' % s
```

Next, start the server with `sudo make setup` and `sudo make check`. Then, create `user1` and `user2`. To launch a request with a fake token, the following URL can be entered, where the fake token is "fake_token": http://127.0.0.1:8080/zoobar/index.cgi/echo?s=fake_token

As a result, the request fails. As the response is invalid, the page displays an Internal Server Error as shown below.



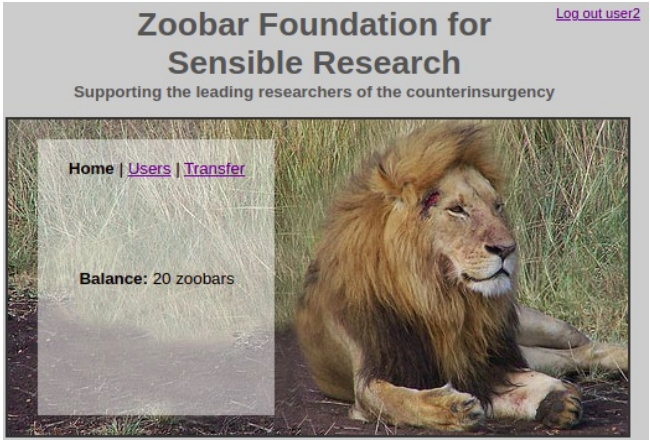
Based on the terminal, the custom error created is observed to be thrown.

```
sender cred.token: 2750fda40ac2903bf5c5ce8a7cda71e9
sender input token: fake token
Traceback (most recent call last):
  File "/zoobar/bank-server.py", line 29, in <module>
    s.run sockpath fork(sockpath)
  File "/zoobar/rpclib.py", line 67, in run sockpath fork
    self.run sock(conn)
  File "/zoobar/rpclib.py", line 43, in run sock
    ret = m(**kwargs)
  File "/zoobar/bank-server.py", line 10, in rpc transfer
    return bank.transfer(sender, recipient, zoobars, sender input token)
  File "/zoobar/bank.py", line 13, in transfer
    raise ValueError("Invalid token. IMPOSTER!!")
ValueError: Invalid token. IMPOSTER!!
```

For the ease of testing, the correct token is printed in the terminal as well. Thus as user1's token = `2750fda40ac2903bf5c5ce8a7cda71e9`, the following URL is entered: http://127.0.0.1:8080/zoobar/index.cgi/echo?s=2750fda40ac2903bf5c5ce8a7cda71e9

The request is valid and the page loads properly as shown below.

To check that the transaction was performed successfully, the home page of user2 is visited. As expected after the transaction, user2 has 20 zoobars.



The transaction can also be observed on the users tab as well when user2 searches for user1, as shown below.

Exercise 9

> Add `profile-server.py` to your web server.

The following is added to `zook.conf` to add a new service `profile_svc` to the web service.

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

[profile_svc]
    cmd = /zoobar/profile-server.py
    args = /profilesvc/sock
    dir = /jail
    uid = 0
    # instructions says need to run as root
    gid = 61012
```

> Change the `uid` value in `ProfileServer.rpc_run()` from 0 to some other value compatible with your design.

Since the `ProfileServer.rpu_run()` will run on its own, a new uid value is assigned to it.

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

The following permissions were granted as well in `chroot-setup.sh`.

```
create_socket_dir /jail/profilesvc 61018:61012 755

set_perms 61018:61012 755 /jail/zoobar/profile-server.py
```

In `profile-server.py`, `rpc_get_xfers` is modified based on the format of the dictionaries from `bank_client.get_log(username)`. `rpc_xfer` is

modified as well to take in the username of the user which is sending the `xfer` request through the parameter of `current_user` so as to account of modifications made to the code in Exercise 8.

```
def rpc_get_xfers(self, username):
    xfers = []
    for xfer in bank_client.get_log(username):
        xfers.append({ 'sender': xfer['sender'],
                       'recipient': xfer['recipient'],
                       'amount': xfer['amount'],
                       'time': xfer['time'],
                     })
    return xfers

def rpc_xfer(self, target, zoobars, current_user):
        cred_db = zoodb.cred_setup()
        current_user_cred = cred_db.query(zoodb.Cred).get(current_user)
        current_user_token = current_user_cred.token
        bank_client.transfer(self.user, target, zoobars, current_user_token)
```

In `granter.py`, api call to `xfer` is modified accordingly as well to include the `current_user` argument.

```
api.call('xfer', target=visitor, zoobars=1, current_user=selfuser)
print 'Thanks for visiting.  I gave you one zoobar.'
```

> Run **sudo make check** to verify that your modified configuration passes our tests.

The modified configuration passes the tests as shown below.

```
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
```

Exercise 10

For each user's profile to only access their own files and not allow them to tamper with the files of other user profile, `rpc_run` will need to create unique directories for each user and then set permissions such that only that others will not be able to access the directory. The unique directories are created using each of their usernames where special characters are escaped with regex. The permissions are set such that the directory's owner and group can write and execute, but the permissions for others are not enabled.

```
import re

...

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

        # change the userdir value
        userdir = '/tmp'
        userdir_name = re.escape(user)
        userdir += '/' + userdir_name

        # make directory
        if not os.path.isdir(userdir):
            os.mkdir(userdir)
            os.chmod(userdir, stat.S_IWRITE | stat.S_IEXEC | stat.S_IWGRP | stat.S_IXGRP)
```

Exercise 11

A new `uid` value is assigned to the forked child, different from the `uid` value of the parent, which is 61018. A copy of the current user's token is also retrieved from the cred_db before the privileges are deescalated, which can then be used in `rpc_xfer` later.

```
class ProfileAPIServer(rpclib.RpcServer):
    def __init__(self, user, visitor):
        self.user = user
        self.visitor = visitor
        cred_db = zoodb.cred_setup()
        my_cred = cred_db.query(zoodb.Cred).get(self.user)
        self.my_token = my_cred.token
        os.setuid(61019)
        os.setgid(61012)

    def rpc_xfer(self, target, zoobars):
        bank_client.transfer(self.user, target, zoobars, self.my_token)
```

With the updated `rpc_xfer` function, the `granter.py` is restored.

```
api.call('xfer', target=visitor, zoobars=1)
print 'Thanks for visiting.  I gave you one zoobar.'
```

All the exercises successfully executed.

```
httpd@istd:~/labs$ 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/zookl
d.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/zookl
d.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..
```