COMPSYS 302 CherryPy Lab

University of Auckland, 2019

PURPOSE

The goal of this lab is to complete a partially-built web-app in CherryPy. Your application will enable local control of a remote web application.

PRELIMINARIES

1. To begin with, go to Canvas and download “cherrypy-starter.zip”
2. Unzip it to a directory of your choice
    a. *This could be the directory for your Python Github repository, e.g. 2019-Python-abcd123 (where abcd123 is your UPI)*
3. Navigate to the directory in the command prompt
4. Run `ls`
5. You should see something like the following:

```
hpea485@en-337568:~/Documents/2019-Python-hpea485$ ls
config.py  main.py  server.py  static
hpea485@en-337568:~/Documents/2019-Python-hpea485$
```

6. Before you get started, we need to install some Python packages
7. In Python3, this is managed by Pip3
8. Confirm this by running `pip3 --version`

```
hpea485@en-337568:~/Documents/2019-Python-hpea485$ pip3 --version
pip 8.1.1 from /usr/lib/python3/dist-packages (python 3.5)
```

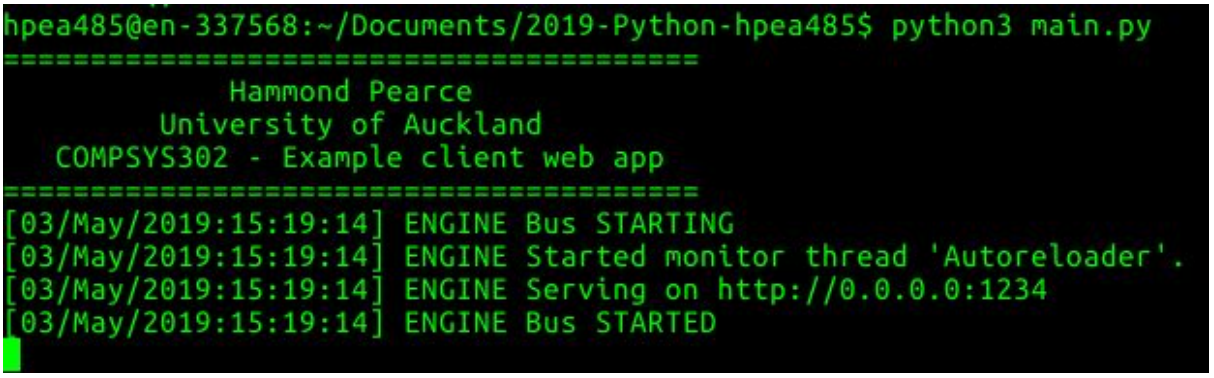
9. Note that your version number might differ.
10. Run the following commands **(ONE BY ONE)**:
    a. pip3 install cherrypy --user
    b. pip3 install jinja2 --user
    c. pip3 install pynacl --user
11. Confirm that the packages installed correctly by running the following

```
hpea485@en-337568:~/Documents/2019-Python-hpea485$ python3
Python 3.5.2 (default, Nov 12 2018, 13:43:14)
[GCC 5.4.0 20160609] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import cherrypy
>>> print(cherrypy.__version__)
18.0.1
>>> import jinja2
>>> print(jinja2.__version__)
2.10
>>> import nacl
>>> print(nacl.__version__)
1.2.1
>>>
```

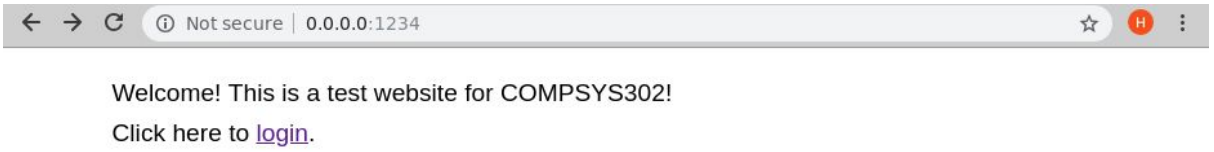
12. If the packages haven’t imported, check that you ran all three commands above, and if so, ask a TA for assistance.

GETTING STARTED

1. To begin with, open `main.py`, and `server.py` in a code editor
    a. E.g. pycharm, gedit, pycrust, visual studio code …
2. The program is started by running `python3 main.py`
    a. Make sure you can find the entry point for the program!
3. Examine how main.py interacts with the other file
    a. How does it access the class specified in `server.py`?
4. Start the program by running `python3 main.py`



5. Now open a web browser and go to the listen URL (http://0.0.0.0:1234)



6. Inspect server.py and try and work out where this HTML is being generated
    a. Inspect the `static` directory and take a look at the basic CSS
7. Try to sign in and out. Use “username”=”user”, and “password”=”password”
8. Try and change the bonus text that is displayed after sign in
9. Try and change the default username and password
10. Try to configure the system to have two sets of usernames and passwords

TALKING TO THE LOGIN SERVER

1. Now you have successfully altered the web application
2. Let’s try and authenticate with the remote web app
3. Go to http://cs302.kiwi.land
4. Go to “log in”
5. Using “username”=”abcd123” and “password”=”[github-username]_[012345689]”, where
    a. abcd123 is your UPI
    b. [github-username] is your github username
    c. and [0123456789] is your university ID number
6. Sign in.
7. Now that you have confirmed your credentials, try open http://cs302.kiwi.land/api/ping
8. Inspect how this is done with `example-api-access.py`
9. Now, open and finish `example-authorised-api-access.py`
    a. Inspect the API documentation to complete steps 1-3

WEBAPP AUTHENTICATING WITH THE LOGIN SERVER

1. Integrate what you have made in example-authorised-api-access with the login process in `server.py`
2. You can keep the username and password around after signing in and use it to send further API requests!

CREATE AND SEND A BROADCAST

1. Open the documentation for PyNaCl at https://pynacl.readthedocs.io/en/stable/signing/
2. Inspect how **private** (i.e. signing) and **public** (i.e. verifying) keys are made using the nacl library
3. Inspect how a key is used to sign a message
4. Using the sample code, either
    a. construct a standalone python file (in the spirit of the `example...access.py` files),
    b. Or, integrate code directly into `server.py` (or appropriate)
        i. You could try and create a new endpoint

5. Try and get your code to call the /api/add_pubkey and the /api/rx_broadcast endpoints!
    a. The API documentation for the login server describes what needs to be signed and the format of the requests

    b. Hint: When constructing your JSON, you'll need to convert your public key from a list of bytes and into a string. One way to do this is to convert your signing key to a hex string representation. You can do this by using the following function chain:

    ```
    signing_key.encode(nacl.encoding.HexEncoder).decode('utf-8')
    ```

    This first converts the key into hex (but still of type bytes), and then decodes that into a hex-encoded string.

    c. You can also do something similar when you do `signing_key.sign()`*, by adding the same argument:* `encoder=nacl.encoding.HexEncoder`*.*

    d. A further hint is over the page… but try and complete the lab without using it!

Additional hint example code for accessing /api/add_pubkey

Once you have got a signing_key correctly generated (or loaded from a file), you may use the following snippet to compute the pubkey and the signature:

```
pubkey_hex = signing_key.verify_key.encode(encoder=nacl.encoding.HexEncoder)
pubkey_hex_str = pubkey_hex.decode('utf-8')

message_bytes = bytes(pubkey_hex_str + username, encoding='utf-8')

signed = signing_key.sign(message_bytes, encoder=nacl.encoding.HexEncoder)
signature_hex_str = signed.signature.decode('utf-8')
```