UP: Sécurité des Réseaux Informatiques d'Entreprise
RUP: Raphael VIERA

Document Title

TP0 – PyFlaSQL Framework

Document Description

This guide provides step-by-step instructions on how to install and utilize the PyFlaSQL framework during practical sessions.

Last update

May 04, 2023

Contributors

Raphael Viera (raphael.viera@emse.fr)

TP0 – PyFlaSQL Framework

PyFlaSQL (Python – Flask – SQLite) is a framework specifically designed for cybersecurity courses in the ISMIN cycle of Mines Saint-Etienne. However, it can also be utilized for various other purposes. The primary objective of this tool is to provide ISMIN students with a solid foundation to create their own penetration testing tool. It enables them to evaluate the security aspects of local networks, web services, or applications. With its robust features and flexibility, PyFlaSQL serves as a valuable resource for developing custom pen testing solutions in the field of cybersecurity education.

Installing PyFlaSQL Framework

Requirement

Linux: Kali Linux is preferred as it comes with several tools already installed but you can use another distribution.
Python3 + Flask + MySQLite (packages will be installed automatically, keep reading).

Create working environment

$ mkdir <user_path>/srie
$ mkdir <user_path>/srie/pyenvs
$ mkdir <user_path>/srie/repos
$ python3 -m venv <user_path>/srie/pyenvs/pyflasql

If the above command does not work, do:
$ sudo apt update
$ sudo apt install python3-venv
Then try again:
$ python3 -m venv <user_path>/srie/pyenvs/pyflasql

Activate the environment

$ source <user_path>/srie/pyenvs/pyflasql/bin/activate

You should see now something similar to:

$ **(pyflasql)** …

Clone the git repository

$ cd <user_path>/srie/repos/
$ git clone https://gitlab.emse.fr/raphael.viera/pyflasql.git

Git repository (in case you want to download it or add to VSCode directly):
https://gitlab.emse.fr/raphael.viera/pyflasql

```
$ cd pyflasql
$ pip3 install -r requirements.txt
$ python3 run.py
```

Open the framework in a web browser

http://127.0.0.1:4990 (you can change the port in the run.py file)
You must register a new user to use the framework. There is no predefined user as the database is initially empty.

Installing VSCode in Kali or Ubuntu (optional)

```
$ sudo apt install code-oss
```
Then search in the menu by code. Launch it.
In VSCode

- Click on "File > Open the folder", navigate to <user_path>/srie/repos/pyflasql
- Click on Terminal > New Terminal
- Check which folder you are:
  - $ pwd
    - You should be at: <user_path>/srie/repos/pyflasql
    - Example: /home/kali/srie/repos/pyflasql
- Activate the environment in the terminal:
  - $ source <user_path>/srie/pyenvs/pyflasql/bin/activate
  - $ python3 run.py

Technology

Python

Python is a high-level, general-purpose programming language.
https://www.python.org

Flask

Flask is used for developing web applications using python.
https://flask.palletsprojects.com

SQLite

SQLite is a database engine written in the C programming language. It is not a standalone app; rather, it is a library that software developers embed in their apps.
https://sqlite.org/index.html

SQLite is used to store username and password of a user in PyFlaSQL but you will probably not need to implement anything else, only if you want to store results in the database.

Architecture

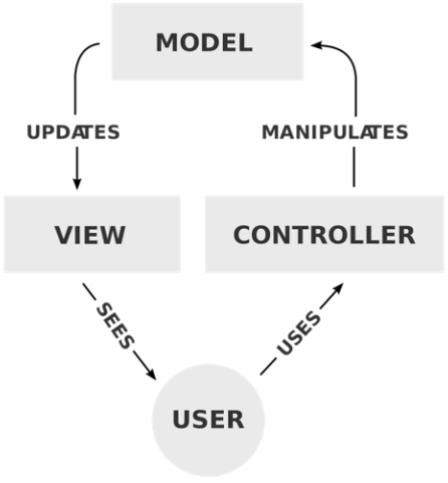
MVC – Model View Controller

PyFlaSQL uses the Model–view–controller (MVC) architecture which is a popular way of organizing your code. The big idea behind MVC is that each section of your code has a purpose, and those purposes are different. Some of your code holds the data of your app, some of your code makes your app look nice, and some of your code controls how your app functions.

MVC is a way to organize your code's core functions into their own, neatly organized boxes. This makes thinking about your app, revisiting your app, and sharing your app with others much easier and cleaner.

The parts of MVC

Model: Model code typically reflects real-world things. This code can hold raw data (database), or it will define the essential components of your app. For instance, if you were building a To-do app, the model code would define what a "task" is and what a "list" is – since those are the main components of a todo app.
View: View code is made up of all the functions that directly interact with the user (html, css, javascript etc in our case). This is the code that makes your app look nice, and otherwise defines how your user sees and interacts with it.
Controller: Controller code acts as a liaison between the Model and the View, receiving user input and deciding what to do with it. It is the brain of the application, and ties together the model and the view.



Folder structure

For more detailed information, refer to the files located in the project folder.

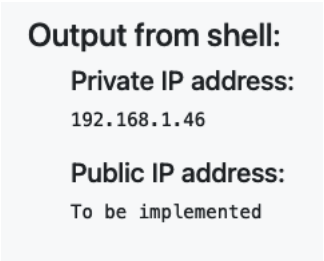
```
<user_path>/srie/repos/pyflasql  # The root folder
    LICENSE
    README.md
    __init__.py
    config.py
    requirements.txt
    run.py

    > app  # The main application folder
        __init__.py  # tells python this is a package
        > assets  # Database and formulary models
            > css # css files for the user interface
                *.css
            > js # javascript files for the user interface
                *.js
            > uploads # any file uploaded by the user
                *.*

        > models  # Database and formulary models
            app.py
            auth.py
            sql.py
            > srie
                > tp1_recon_footprint
                    forms.py
                > tp2_scanning_networks
                    forms.py
                > tp3_enumeration
                    forms.py
                > tp4_gaining_access
                    forms.py
            > toolbox
                forms.py

        > View  # User interface
            > routes  # Generate routes (paths) for URLs
                blueprint.py
            > templates  # HTML pages for the user interface
                about.html
                base.html
                dashboard.html
                index.html
                login.html
                register.html
                > srie
                    home.html
                    > tp1_recon_footprint
                        home.html
                        ipaddr.html
                        pingaddr.html
```

```
            > tp2_scanning_networks
                home.html
            > tp3_enumeration
                home.html
            > tp4_gaining_access
                home.html
            > user_profile
                user_profile.html
        > toolbox
            home.html
            > database
                home.html
                insert_data.html
            > wtforms
                home.html
                upload_form.html
                user_reg_form.html

> controllers  # Contains the business logic
    __init__.py
    controller.py
    utils.py
    > srie
        > tp1_recon_footprint
            controller.py
        > tp2_scanning_networks
            controller.py
        > tp3_enumeration
            controller.py
        > tp4_gaining_access
            controller.py
        > user_profile
            controller.py
    > toolbox
        > database
            controller.py
        > wtforms
            controller.py


> instance  # folder created by flask containing the SQLite database (do not edit it)
    database.db
```

Adding a functionality

Show private and public IP address

In this exercise, we will enhance a function that enables checking both private and public IP addresses. As observed in the image below, the current implementation only retrieves the private IP address:



Files you need to check for this example:

app/controllers/srie/tp1_recon_footprint/controller.py
app/view/templates/srie/tp1_recon_footprint/ipaddr.html

In the *app/controllers/srie/tp1_recon_footprint/controller.py* file you must use shell to execute the following command: "*curl -s https://raphaelviera.fr/ismin/toolbox/my_ip.php*".

```
@login_required
def srie_tp1_ipaddr():
   content = {"ip_address_private": "x.x.x.x",
              "ip_address_public: "x.x.x.x",
              "cmd_ip_address_private": f"hostname -I",
              "cmd_ip_address_public": f"curl -s https://raphaelviera.fr/ismin/toolbox/my_ip.php"
             }
   content["ip_address_private"] = get_shell_output(content["cmd_ip_address_private"])
   content["ip_address_public"]  = get_shell_output(content["cmd_ip_address_public"])
   return render_template(url_for('blueprint.srie_tp1_ipaddr')+'.html', content=content)
```

There is no need to change the app/templates/srie/tp1_recon_footprint/ipaddr.html file, however, check it anyway to see how the commands are called in the .html file.

Save the files, log in again and test what you've implemented so far.

Implementing WHOIS

In this exercise, you will be implementing the WHOIS command, which is an internet service that provides information about a domain name.

This exercise involves:

1. Use an HTML form to collect information from the user, such as the domain name they want to look up. This form can be created using HTML and can include input fields for the user to enter the domain name.
2. Pass the information from the HTML form to a Python script using Flask. Flask can be used to create a route that receives the form data and passes it to a function for processing.
3. In the function, execute the "whois" command in the terminal shell using the subprocess module in Python. The output of the command can be captured and stored in a variable for further processing.
4. Finally, the results of the "whois" command can be displayed on the same web page using HTML and Flask. Flask can be used to pass the results to the HTML template and render it on the page.

By following these steps, you can create a simple web application that collects user input, executes a command-line tool, and displays the results on a web page. This approach can be extended to other command-line tools.

You will implement the following command:

$ curl -s https://raphaelviera.fr/ismin/toolbox/whois/whois.php?domain={domain}

{domain_name} The domain name. Example: raphaelviera.fr

You can also implement the command **$ whois -p {port} {domain_name}**

However, the port used by this command (43) is blocked by MSE's firewall, that's why it is provided as a web service by raphaelviera.fr.

When working with command-line tools, it's important to study the available commands and parameters to understand their functionality and potential use cases. Once you have a basic understanding of the command and its options, you can explore other parameters that may be useful for your specific needs.

To do this, you can consult the command's documentation, either by typing "**man <command>**" in the terminal or by searching for it online. The documentation should provide information about the available parameters, what they do, and how to use them.

To complete this exercise, you can use the "Ping an IP address" functionality as a base. The following figures show step-by-step the files used to implement the "whois" functionality:

Models

app/models/srie/tp1_recon_footprint/forms.py

Class used to create the form, copied adapted from PingAddrForm(FlaskForm)

```python
from flask_wtf import FlaskForm
from wtforms import StringField, PasswordField, BooleanField, SubmitField, IntegerField
from wtforms.validators import DataRequired, Email, InputRequired, Length, ValidationError, NumberRange

class WhoisForm(FlaskForm):
    domain = StringField(validators=[
        InputRequired(), Length(min=2, max=300)], render_kw={"placeholder": "Domain"})

    submit = SubmitField('Submit')
```

Controller

app/controllers/srie/tp1_recon_footprint/controller.py

Function copied from srie_tp2_pingaddr()

In this file you need to:

1. Import the WhoisForm from Models (line 14)
2. Create the function srie_tp1_whois() (lines 74 to 102)

```python
from ....models.srie.tp1_recon_footprint.forms import WhoisForm
```

```python
@login_required
def srie_tp1_whois():
    """
        Handles the logic for view/templates/srie/tp1_recon_footprint/whois.html
        Login is required to view this page

        Print in the user interface private and public IP addresses.

        Args:
            - None.

        Returns:
            - rendered template view/templates/srie/tp1_recon_footprint/whois.html with content passed as a context varia
        """
    # Create a dict to store the formulary and the shell output. This dict is passed to the .html file.
    content = {"form": WhoisForm(),
               "command_executed": "Waiting ...",
               "command_output": "Waiting ..."
               }

    if content["form"].validate_on_submit():
        # Get IP address and number of pings from the user interface (UI)
        domain = content["form"].domain.data
        content["command_executed"] = f"curl -s https://raphaelviera.fr/ismin/toolbox/whois/whois.php?domain={domain}"
        content["command_output"] = get_shell_output(content["command_executed"])
        # print(content["shell_output"])  # for debug only
        return render_template(url_for('blueprint.srie_tp1_whois')+'.html', content=content)

    return render_template(url_for('blueprint.srie_tp1_whois')+'.html', content=content)
```

View

app/view/routes/blueprint.py

In this file you need to:

1. Import the function srie_tp1_whois from app/controllers/srie/tp1_recon_footprint/controller.py
2. Create a route for the whois in the user interface (/srie/tp1_recon_footprint/whois)

```python
from ...controllers.srie.tp1_recon_footprint.controller import srie_home, srie_tp1_recon_footprint, srie_tp1_ipaddr, srie_tp1_whois
```

```python
blueprint.route('/srie/tp1_recon_footprint/whois', methods=['GET', 'POST'])(srie_tp1_whois)
```

app/view/templates/srie/tp1_recon_footprint/home.html

In this file you need to:

1. Add the URL to the whois page provided by blueprint (read more)
2. Add a description about the tool.

```
54            <li class="list-group-item">
55                <h5><a href="{{url_for('blueprint.srie_tp1_whois')}}" class="list-group-item-action">WHOIS</a></h5>
56                <p class="lead">
57                    WHOIS description ...
58                </p>
59            </li>
```

app/view/templates/srie/tp1_recon_footprint/whois.html

In this file you need to:

1. Change the page title (browser bar) – line 29
2. Change the page title (top of the page) – line 32
3. Add descriptions, improve help section, give command example etc.

```
28  {% extends "base.html" %}
29  {% block title %}SRIE > TP1 - Reconnaissance / Footprint > WHOIS{% endblock %}
30  {% block content %}
31  <div class="container p-5 my-5">
32      <h2 class="text-center">SRIE > TP1 - Reconnaissance / Footprint > WHOIS</h2>
```

```
42
43      <div class="container p-5 my-5 border bg-light">
44          <p class="lead">The "whois" tool is used in the command line shell. So far it takes one parameter:</p>
45          <code>{% raw %}$ curl -s https://raphaelviera.fr/ismin/toolbox/whois/whois.php?domain={domain} {% endraw %}</code>
46      </div>
```

```
62          <h5>Command example:</h5>
63          <code>{% raw %}$ curl -s https://raphaelviera.fr/ismin/toolbox/whois/whois.php?domain=raphaelviera.fr {% endraw %}</code>
64      </div>
65
66      <!-- Command Execution section -->
67      <h3 class="mt-5 text-center">Command Execution</h3>
68
69      <div class="container p-5 my-5 border bg-light">
70          <form method="post" action="">
71              {{ content["form"].csrf_token }}
72
73              <div class="form-group mt-2">
74                  <label for="{{ content['form'].domain.id }}">{{ content["form"].domain.label.text }}</label>
75                  {{ content["form"].domain(class="form-control") }}
76              </div>
77
78              <button type="submit" class="btn btn-primary mt-2">Submit</button>
79          </form>
80      </div>
```

Result

Access 127.0.0.1:4990, log in and navigate using the menu until you find the implemented tool (WHOIS) inside TP1's homepage.

127.0.0.1:4990/srie/tp1_recon_footprint/whois

SRIE > TP1 - Reconnaissance / Footprint > WHOIS

WHOIS description...

Command Used

The "whois" tool is used in the command line shell. So far it takes one parameter:

```
$ curl -s https://raphaelviera.fr/ismin/toolbox/whois/whois.php?domain={domain}
```

Command Execution

Command Output

Command executed:

```
curl -s https://raphaelviera.fr/ismin/toolbox/whois/whois.php?domain=raphaelviera.fr
```

Output from shell:

```
%%
%% This is the AFNIC Whois server.
%%
%% complete date format: YYYY-MM-DDThh:mm:ssZ
%%
%% Rights restricted by copyright.
%% See https://www.afnic.fr/en/domain-names-and-support/everything-there-is-to-know-about-domain-names/find-a-domain-name-or-a-holder-
%%
%%

domain:                 raphaelviera.fr
status:                 ACTIVE
eppstatus:              active
hold:                   NO
holder-c:               ANO00-FRNIC
admin-c:                ANO00-FRNIC
tech-c:                 GI767-FRNIC
registrar:              KEY-SYSTEMS GmbH
```

Implementing the examples provided in this exercise will help you to develop the minimum set of skills required to implement other tools such as dnsrecon, traceroute, nmap, dig, and more. By understanding the concepts and techniques used in these examples, you can apply

them to other tools and scenarios to expand your knowledge and capabilities in pen testing and related fields.

In addition to the examples provided in this exercise, you can also explore the toolbox category for a list of building block tools that can be used in the development of your application. These tools can provide useful functionality that can be integrated into your own applications to enhance their capabilities and save development time. By leveraging these tools, you can focus on implementing the specific features and functionality that are unique to your application, rather than re-inventing the wheel for common tasks.

Learning material:

Python: https://www.w3schools.com/python/default.asp
Flask: https://flask.palletsprojects.com/en/2.2.x/
Flask Blueprint: https://flask.palletsprojects.com/en/2.2.x/blueprints/
SQLite using SQLAlchemy: https://www.tutorialspoint.com/sqlalchemy/index.htm
Bootstrap: https://www.w3schools.com/bootstrap/
HTML: https://www.w3schools.com/html/default.asp

If you're not interested in learning all these technologies, don't worry – you can still create your own web application by copying and pasting code to implement your tools. The main goal here is to learn how to build a web application and use building blocks as a foundation to develop your own pen testing framework. By using pre-existing code and tools, you can focus on understanding how they work and customizing them to suit your specific needs. So, whether you prefer to create everything from scratch or use existing code, the most important thing is to gain a solid understanding of the underlying concepts and principles of network pen testing.