SUID vs Capabilities

2017-12-07

#GNU/Linux #security #SUID #capabilities

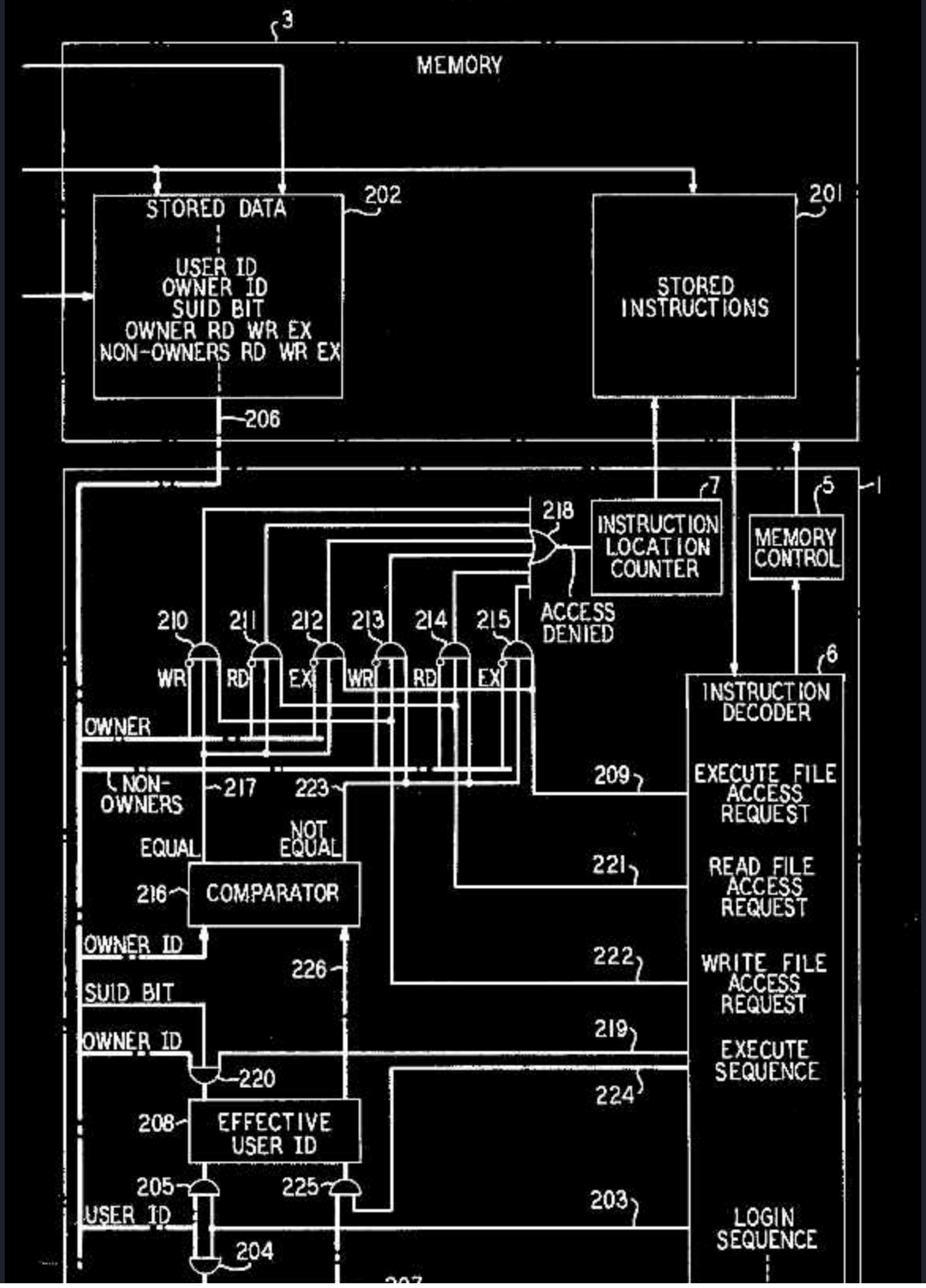
Review SUID vs Capabilities to gain privileges in practice;

Intro

As you know, in UNIX based systems access model not everything from user level is allowed to do. There are some actions which require extra privileges (open RAW network socket, update system passwords database, mount volume block device, connect to VPN, turn power off, etc.). All these actions are not allowed to do from regular “user” level by default and can be requested trough special services like PolicyKit, but there are some ways to do it without any services.

Probably, the oldest way to gain more privileges in UNIX system is SUID bit. It was invented by Dennis Ritchie in 1979 and it even has the patent US4135240 which is currently on a public domain.

FIG. 1B


3
MEMORY
STORED DATA
202
USER ID
OWNER ID
SUID BIT
OWNER RD WR EX
NON-OWNERS RD WR EX
206
STORED INSTRUCTIONS
201
1
7
INSTRUCTION LOCATION COUNTER
218
5
MEMORY CONTROL
ACCESS DENIED
210
211
212
213
214
215
WR
RD
EX
WR
RD
EX
OWNER
NON-OWNERS
217
223
EQUAL
NOT EQUAL
216
COMPARATOR
OWNER ID
226
SUID BIT
OWNER ID
220
208
EFFECTIVE USER ID
205
225
USER ID
204
6
INSTRUCTION DECODER
209
EXECUTE FILE ACCESS REQUEST
221
READ FILE ACCESS REQUEST
222
WRITE FILE ACCESS REQUEST
219
EXECUTE SEQUENCE
224
203
LOGIN SEQUENCE


```
SUID/Setuid = "set user ID upon execution"
```

After the process will be started and called setuid() function process UID will be changed to the same which is set for the file on the file system level. If a file owner is “root” then UID will be changed to “root” despite the file was started from regular user (example: “Joe”).

As a real example “passwd” (standard utility in UNIX like systems designed to change user password) definitely will require privileges to open file “/etc/shadow” owned by root and without read permissions from others (root:shadow -rw-r--) .

```
user@host$ ls -lah `which passwd`
-rwsr-xr-x 1 root root 59K May 17  2017 /usr/bin/passwd
   |
    -- s = SUID bit is set
```

Let’s find out what is good and bad in this method:

[+] Why SUID is good?

- Easy to use;
- It was tested by time;
- It’s widely supported in all Unix-like systems;
- Suid support is enabled by default and with default mount options;
- It don’t require extended attributes support (vs Capabilities);

- If the process was developed good it will drop privileges by setuid(uid) immediately after it done with privileged operation to minimize security risk;

Some of SUID related commands

```
# Find setuid-root files:
$ find /usr/bin /usr/lib -perm /4000 -user root
# Find setgid-root files:
$ find /usr/bin /usr/lib -perm /2000 -group root
```

Mount options related with suid

```
    suid   Allow set-user-identifier or set-group-identifier bits to take ef
    nosuid Do not allow set-user-identifier or set-group-identifier bits to
```

[-] Why SUID is bad?

SUID is like a hammer compared with a scalpel in way how it give privileges. All or nothing. Deal with it. Here are two questions:

- Should we trust privileges escalation to process code itself?
- Do every process require all root possibilities to perform one privileged action?

Obviously, answer is “no” for both questions.

Enough with theory! Let’s Practice!

Let’s create simplest sudo replacement with suid-bit set on executable file and setuid() function from unistd.h:

Preparation on system 1

```
# file system should support SUID
cd /media/usb1
cat > s.c << EOF
// s.c
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <unistd.h>
int main(int argc, char **argv, char **env)
{
    if (setuid(0))
    {
        perror("Error with setuid(0)");
        return 1;
    }
    if (argc<2)
    {
        printf("Required command is missing.\n Try use as:\n %s /bin/sh\n",
        return 1;
    }
    system(argv[1]);
    return 0;
}
EOF
$ gcc -o s s.c
# Let's test it without SUID to see error message:
$ ./s
Error with setuid(0): Operation not permitted
# Let's set SUID and change the owner
root@host# sudo chown root s
root@host# sudo chmod u+s s
```

```
# Let's test now
$ whoami
user
./s /bin/bash
$ whoami
root
```

Locally it works ok, same way as it should work on some different system:

```
$ whoami
user
$ /$media/usb1/s /bin/bash
$ whoami
root
```

Mini local sudo is ready :)

> **Warning**
>
> If file system mounted with “nosuid” option set - this will not work.

SUID and vulnerabilities:

There can be vulnerabilities in sensitive locations, for example:

> **CVE Details**

* CVE-201701000253

> “An unprivileged local user with access to SUID (or otherwise privileged) PIE binary could use this flaw to escalate their privileges on the system.”

Real use cases when it can be used

ping:

```
$ ping 8.8.8.8
ping: socket: Operation not permitted
$ sudo chmod u+x /bin/ping
$ ping 8.8.8.8
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.
64 bytes from 8.8.8.8: icmp_seq=1 ttl=58 time=10.0 ms
^C
--- 8.8.8.8 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 10.057/10.057/10.057/0.000 ms
```

Ok. So, what about other way?

Capabilities

What is it?

From wikipedia:

Capability-based security refers to the principle of designing user programs such that they directly share capabilities with each other according to the principle of least privilege, and to the operating

system infrastructure necessary to make such transactions efficient and secure. Capability-based security is to be contrasted with an approach that uses hierarchical protection domains.

Is it related with some standard?

Yes.

- POSIX 1003.1e

“Ping” example again

```
$ ping 8.8.8.8
ping: socket: Operation not permitted
#Setting capabilities for NET_RAW sockets:
$ sudo setcap "cap_net_raw+p" /bin/ping
$ ping 8.8.8.8
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.
64 bytes from 8.8.8.8: icmp_seq=1 ttl=58 time=27.7 ms
^C
--- 8.8.8.8 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 27.769/27.769/27.769/0.000 ms
```

if required to remove privileges you can do:

```
$ /sbin/setcap -r /bin/ping
```

How they are stored on block device

If you are curious and want to know how they are actually stored on hard disk you can perform some investigations. Of course, you don't have to do this if you just want to make them work. :)

```
getfattr -d -m- /bin/ping
getfattr: Removing leading '/' from absolute path names
# file: bin/ping
security.capability=0sAAAAAgAgAAAAAAAAAAAAAAAAAA=
```

List of available capabilities

> **Warning**
>
> Your list can be different. Check *your* kernel version first.

```
$ grep "define *CAP" /usr/src/linux-headers-`uname -r`-common/include/uapi/
#define CAP_CHOWN            0
#define CAP_DAC_OVERRIDE     1
#define CAP_DAC_READ_SEARCH  2
#define CAP_FOWNER           3
#define CAP_FSETID           4
#define CAP_KILL             5
#define CAP_SETGID           6
#define CAP_SETUID           7
#define CAP_SETPCAP          8
#define CAP_LINUX_IMMUTABLE  9
#define CAP_NET_BIND_SERVICE 10
#define CAP_NET_BROADCAST    11
#define CAP_NET_ADMIN        12
#define CAP_NET_RAW          13
#define CAP_IPC_LOCK         14
#define CAP_IPC_OWNER        15
```

```
#define CAP_SYS_MODULE      16
#define CAP_SYS_RAWIO       17
#define CAP_SYS_CHROOT      18
#define CAP_SYS_PTRACE      19
#define CAP_SYS_PACCT       20
#define CAP_SYS_ADMIN       21
#define CAP_SYS_BOOT        22
#define CAP_SYS_NICE        23
#define CAP_SYS_RESOURCE    24
#define CAP_SYS_TIME        25
#define CAP_SYS_TTY_CONFIG  26
#define CAP_MKNOD           27
#define CAP_LEASE           28
#define CAP_AUDIT_WRITE     29
#define CAP_AUDIT_CONTROL   30
#define CAP_SETFCAP         31
#define CAP_MAC_OVERRIDE    32
#define CAP_MAC_ADMIN       33
#define CAP_SYSLOG          34
#define CAP_WAKE_ALARM          35
#define CAP_BLOCK_SUSPEND   36
#define CAP_AUDIT_READ        37
(...)
```

Required item is - 13

```
#define CAP_NET_RAW         13
```

To convert RAW format to text readable Capabilities we need to read extended file system attributes first:

```
$ getfattr -d -m- /bin/ping
getfattr: Removing leading '/' from absolute path names
# file: bin/ping
security.capability=0sAAAAAgAgAAAAAAAAAAAAAAAAAA=
```

Let’s remove “0s” and decode it as Base64 encoded 5 integers;

```
# echo -n "AAAAAgAgAAAAAAAAAAAAAAAAAA=" | base64 -d | hexdump -v -e '16/1
\x00\x00\x00\x02\x00\x20\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x0
```

Let’s unpack this data array using Python unpack function:

```
$ python
Python 2.7.13 (default, Jan 19 2017, 14:48:08)
Type "help", "copyright", "credits" or "license" for more information.
>>> import struct;
>>> p='\x00\x00\x00\x02\x00\x20\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00
>>> struct.unpack("<IIIII",p)
(33554432, 8192, 0, 0, 0)
```

Let’s covert dec 8192 to bin

```
echo "obase=2; 8192" | bc
10000000000000
```

- Bit 13 is set 1 (True); All other are unset 0 (False);

```
#define CAP_NET_RAW          13
```

Match!

So now we see that these two actions are equal in our case (with current version)

So next two commands both are working:

```
# 1
setcap cap_net_raw+p /bin/ping
# 2
setfattr -n security.capability -v "0sAAAAAgAgAAAAAAAAAAAAAAAAAA=" /bin/pi
```

```
getfattr -d -m- /bin/ping
# file: bin/ping
security.capability=0sAAAAAgAgAAAAAAAAAAAAAAAAAA=

getcap /bin/ping
/bin/ping = cap_net_raw+p
```

Or there are more easy way:

```
import ctypes

libcap = ctypes.cdll.LoadLibrary("libcap.so")
cap_t = libcap.cap_get_file('/bin/ping')
libcap.cap_to_text.restype = ctypes.c_char_p
libcap.cap_to_text(cap_t, None)
which gives me:
```

```
'= cap_net_raw+p'
```

Rsync / Backup / TAR -ing

Using rsync or tar with saving capabilities

```
tar –selinux –acls –xattrs -cvf file.tar /var/www
```

```
--selinux – Save the SELinux context to the archive called file.tar.
--acls – ASave the ACLs to the archive called file.tar.
--xattrs – Save the user/root xattrs to the archive called file.tar Please
-c – Create a new archive called file.tar.
-v – Verbose output.
-f file.tar – Archive file name.
/var/www – Create archive called file.tar from directory /var/www
```

Conclusion

If you want more secure environment and your system is using modern kernel (after 2.4.xx) and your file system supports extended attributes (ext2,3,4/ etc.) than Capabilities - is your choice;

URL's

- Capability on GRSecurity wiki
- http://man7.org/linux/man-pages/man2/getxattr.2.html
- http://unixetc.co.uk/2016/05/30/linux-capabilities-and-ping/
- Linux kernel docs
- man capabilities
- man cap_from_text

- man cap_set_file
- man getcap
- man getfattr
- man setcap
- Source code linux/capability.h
- nmap using Capabilies

READ OTHER POSTS

← fix Virtualbox on Devuan AS… Some of non technical must … →