-Shellcode

실행중인 프로그램에 삽입된 코드를 뜻함, 작성할 때 메모리상 배치, 세그먼트에 신경쓰지 않아도 됨. 명령 shell을 실행하여 공격자가 해당 시스템을 제어함.

-Assembly code

| Instructions | Meaning |
|---|---|
| mov destination, source | 목표 피연산자에 소스 피연산자를 복사합니다. |
| PUSH value | stack에 Value 값을 저장합니다. |
| POP register | stack 상위의 값을 레지스터에 저장합니다. |
| CALL function_name(address) | 리턴을 위해 CALL 명령어의 다음 명령주소를 스택에 저장한 후 함수의 위치로 점프를 합니다. |
| ret | 스택으로 부터 리턴 주소를 팝하고 그 곳으로 점프하여 함수에서 리턴 합니다. |
| inc destination | 목표 피연산자를 1증가 시킵니다. |
| dec destination | 목표 피연산자를 1 감소 시킵니다. |
| add destination, value | 목표 피연산자에 value 값을 더합니다. |
| sub destination, value | 목표 피연산자에 value 값을 뺍니다. |
| or destination, value | 비트 or 논리 연산을 한다. 최종 결과는 목표 피연산자에 저장됩니다. |
| and destination, value | 비트 and 논리 연산을 한다. 최종 결과는 목표 피연산자에 저장됩니다. |
| xor destination, value | 비트 xor 논리 연산을 한다. 최종 결과는 목표 피연산자에 저장됩니다. |
| lea destination, source | 목표 피연산자에 소스 피연산자의 유효 주소를 로드합니다. |

이외에도 시스템 함수를 호출하는 명령어 "int 0x80","syscall"은 각각 Eax,Rax에 저장된 시스템 함수를 호출한다.

ASM32.asm

```
section .data                              ; 데이터 세그먼트
    msg db  "Hello, world!",0x0a, 0x0d     ; 문자열과 새 줄 문자, 개행 문자 바이트

section .text                              ; 텍스트 세그먼트
    global  _start                         ; ELF 링킹을 위한 초기 엔트리 포인트

_start:
    ; SYSCALL: write(1,msg,14)
    mov eax, 4      ; 쓰기 시스템 콜의 번호 '4' 를 eax 에 저장합니다.
    mov ebx, 1      ; 표준 출력를 나타내는 번호 '1'을 ebx에 저장합니다.
    mov ecx, msg    ; 문자열 주소를 ecx에 저장니다.
    mov edx, 14     ; 문자열의 길이 '14'를 edx에 저장합니다.
    int 0x80        ; 시스템 콜을 합니다.

    ; SYSCALL: exit(0)
    mov eax, 1      ; exit 시스템 콜의 번호 '1'을 eax 에 저장합니다.
    mov ebx, 0      ; 정상 종료를 의미하는 '0'을 ebx에 저장 합니다.
    int 0x80        ; 시스템 콜을 합니다.
```

다음 코드를 보면 데이터 세그먼트 부분에 개행 문자(0x0a)가 있고 텍스트 세그먼트 부분에는 실제 어셈블리 명령어가 있다. Global _start는 ELF바이너리를 생성할 때 링커에게 시작점을 알려준다.

ASM32.s

```
BITS 32                         ; nasm에게 32비트 코드임을 알린다

call helloworld                 ; 아래 mark_below의 명령을 call한다.
db "Hello, world!", 0x0a, 0x0d  ; 새 줄 바이트와 개행 문자 바이트

helloworld:
    ; ssize_t write(int fd, const void *buf, size_t count);
    pop ecx         ; 리턴 주소를 팝해서 exc에 저장합니다.
    mov eax, 4      ; 시스템 콜 번호를 씁니다.
    mov ebx, 1      ; STDOUT 파일 서술자
    mov edx, 15     ; 문자열 길이
    int 0x80        ; 시스템 콜: write(1,string, 14)

    ; void _exit(int status);
    mov eax,1       ;exit 시스템 콜 번호
    mov ebx,0       ;Status = 0
    int 0x80        ;시스템 콜: exit(0)
```

Call 명령어에 의해 함수가 호출될 때 call 명령어 다음 명령어의 주소(리턴주소)를 스택에 push 합니다. 함수가 끝난 후에 ret명령어를 이용해 스택에 push된 주소를 EIP에 저장합니다. 여기서 ret주소를 변경하면 실행의 흐름을 변경할 수 있습니다. Call 명령어 뒤에 메시지를 저장하면 pop 명령어를 이용해 ecx 레지스터에 주소값을 전달할 수 있습니다.

**Convert output format of shellcode**

```
lazenca0x0@ubuntu:~/ASM$ python
Python 2.7.12 (default, Nov 20 2017, 18:23:56)
[GCC 5.4.0 20160609] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> f = open('ASM32','r')
>>> data = f.read()
>>> data
'\xe8\x0f\x00\x00\x00Hello, world!\n\rY\xb8\x04\x00\x00\x00\xbb\x01\x00\x00\x
>>>
```