시스템 해킹 스터디

과제 (01)

구본현

목차

64 bit	32bit	16bit	8bit
RAX	EAX	AX	AH,AL
RBX	EBX	BX	BH,BL
RCX	ECX	CX	CH,CL
RDX	EDX	DX	DH,DL

1. 레지스터별 역할 정리

A. 범용 레지스터

EAX(Accumulator) : 산술연산 때 상수 또는 변수 그리고 반환 값이 들어간다.

EBX(Base) : 메모리의 주소가 들어간다.

ECX(Counter) : 반복연산 때 반복된 횟수가 들어간다

EDX(Data) :부호확장, 큰 곱하기 나누기 연산 때 EAX 와 같이 사용된다.

B. 색인 레지스터

ESI(Source) : 복사를 할 때 원본의 주소가 들어간다.

EDI(Destination): 복사를 할 때 대상의 주소가 들어간다.

C. 포인트 레지스터

ESP(Stack) : 스택의 맨 위의 주소가 저장된다.

EBP(Base) : 스택의 맨 아래 주소가 저장된다.

EIP(Instruction) : 다음 실행할 명령어 또는 그 명령어의 주소가 들어간다.

2. 32bit 함수 호출 규약정리 및 비교

가변인자 함수란 c 의 printf 와 같이 함수의 인자의 개수가 변할 수 있는 함수를 뜻한다.

세 방식 모두 반환값은 4byte 하위는 EAX 4byte 상위는 EDX 에 저장된다.

Caller : 호출한(부른) 함수
Callee : 호출된(불러진) 함수

Fastcall 방식은 ECX 와 EDX 에 두 인자가 스택 대신 들어가 활용되기 때문에 빠른 호출 규약이다.

규약 이름	전달인자(파라미터) 해석	스택 정리	기타
Cdecl	역순으로 스택에 들어감 Func1(int a, int b) 일 때 a=1, b=2 이면 스택에 2, 1 PUSH	가변인자 함수를 지원하기 때문에 끝난 후 ESP 의 위치가 같아야 함 Caller 가 Callee 의 스택을 정리함 ex : <add esp, (size)>	Visual Studio 2015 기준 기본형식임
Stdcall		Callee 가 자신의 스택을 직접 정리함 ex : <ret>	Win 32 API 의 기본형식임
Fastcall	처음 두 인자는 순서대로 ECX, EDX 에 들어가며 나머지 인자는 역순으로 스택에 들어감 Func1(int a, int b, int c, int d) 일 때 a=1, b=2, c=3, d=4, 이면 ECX=1 EDX=2 그리고 스택에 4 와 3 PUSH		64bit 에선 Fastcall(의 확장) 만 사용됨 아직 비표준화임, 컴파일러마다 조금씩 다름

```
main:

push    ebp
push    ebp, esp

push    2
push    3
call    _sum
add     esp, 8
```

```
sum:

push    ebp
push    ebp,

sub     esp,
mov     eax,
add     eax,
mov     [ebp-
mov     eax,

mov     esp,
pop     ebp
ret     8
```

[좌] cdecl 로 호출된 함수 sum 이 끝난 뒤에 스택정리가 이루어지는 모습

[우] stdcall 로 호출된 함수 sum 이 끝나기 전 스택정리가 이루어지는 모습

3. 64bit 함수 호출 규약 정리 및 비교

두 방식 모두 반환값은 정수일 땐 RAX, RDX 를 실수일 땐 XMM0~1 을 사용한다.

윈도우의 경우 인자가 레지스터에 담기지만 미리 레지스터의 공간 (4 개)가 스택에 할당된다.

운영체제	전달인자(파라미터) 해석	
	정수형	실수형
리눅스	처음 6 개의 인자는 RDI, RSI, RDX, RCX, R8, R9 를 사용, 나머지는 스택	처음 8 개의 인자는 XMM0~XMM7 를 사용 나머지는 스택
윈도우	처음 4 개의 인자는 RCX, RDX, R8, R9 를 사용, 나머지는 스택	처음 4 개의 인자는 XMM0~XMM3 를 사용 나머지는 스택

커널 : 운영체제의 중요한 핵심 부분, 운영체제의 여러부분 및 응용프로그램의 수행에 필요한 여러가지 서비스를 제공함

유저모드 : 유저가 접근할 수 있는 영역을 제한적으로 두고 중요한 자원에 접근하지 못하는 모드

커널모드 : 모든 자원(드라이버, 메모리, CPU 등)에 접근, 명령 가능

4. 리눅스 시스템콜의 역할과 syscall 인자 전달 방법 정리

A. 개요

서로 분리된 관계인 사용자모드와 커널모드을 이어주는 인터페이스이다.

흡사 프로그래밍 언어에서 클래스에 접근시키기 위해 메소드를 공개하는 것과 유사한 개념이다.

프로세스 제어, 파일조작, 장치관리, 시스템 정보 및 자원관리, 통신 관련등의 종류가 있다.

B. 인자 전달 방법

1. 레지스터를 통한 전송

레지스터의 크기보다 큰 인수가 존재 할 수 있기 때문에 좋은 방법이 아니다.

2. 블록 또는 테이블을 통한 전송

인수가 메모리 내의 블록이나 테이블에 저장되고, 해당 주소가 레지스터의 인수로 전달된다.

가장 많이 사용되는 방법이다.

3. 스택을 통한 전송

인수는 프로그램에 의해 스택에 넣어지고, 운영체제에 의해 꺼내지게 된다.

5. 예제 어셈블리어가 어떤 코드인지 분석해오기

```
mov rax, 1
mov rdi, 1
mov rsi, message
mov rdx, 14
syscall
mov rax, 60
xor rdi, rdi
syscall
```

message 는 문자열이 들어간 전역변수이다.

(1) rax 에는 시스템 콜 고유번호가 오게 된다. rax 가 사용되고 있음에 따라 64bit 환경이며 64bit System Call Table 에 1 번은 write 기능을 하고 있음을 알 수 있다.

%rax	System call	%rdi	%rsi	%rdx
0	sys_read	unsigned int fd	char *buf	size_t count
1	sys_write	unsigned int fd	const char *buf	size_t count

(2-5) 인자를 대입해보면 공백, 개행 문자가 포함된 문자열 14 자 만큼 'message'를 표준출력 하겠다는 것을 알 수 있다.

(6) 60 번은 exit 의 기능을 하고 있다.

(7) xor 은 두 연산자가 다를 때 1, 같을 때 0 을 destination 연산자에 넣는다. 즉 똑같은 레지스터를 넣어줌으로 인해 그 레지스터를 초기화 한다.

(8) 종료기능이 수행되어 종료된다.

참고자료
http://blog.rchapman.org/posts/Linux_System_Call_Table_for_x86_64/