01.The basics technic of Shellcode

Unknown macro: 'html'

List

Shellcode

- **Shellcode shell .**
  - Shellcode Machine code .
  - Shellcode .
  - .
  - .

The basics of shellcode(ubuntu-16.04)

C ASM Machine code

- **Shellcode .**
  - C Assembly, Machine code .
    - C code Assembly code Assembly code Machine code
- **Machine code .**
  - , Shell .
  - Shellcode Assembly, Machine code .

C ASM Machine code

C code	Assembly code	Machine code

Helloworld.c
`#include<stdio.h>`
`int main(){`
`  printf("Hello world!\n");`
`  return 1;`
`}`
	Helloworld.s
	`0000000000400526 <main>:`
	`  400526: push %rbp`
	`  400527: mov %rsp,%rbp`
	`  40052a: mov $0x4005c4,%edi`
	`  40052f: callq 400400 <puts@plt>`
	`  400534: mov $0x1,%eax`
	`  400539: pop %rbp`
	`  40053a: retq`
		Machine code
		`400526: 55`
		`400527: 48 89 e5`
		`40052a: bf c4 05 40 00`
		`40052f: e8 cc fe ff ff`
		`400534: b8 01 00 00 00`
		`400539: 5d`
		`40053a: c3`

Assembly code

- **Shellcode** **Assembly** .
  - Instructions .

Intel syntax

Instructions	Meaning
mov destination, source	.
PUSH value	stack Value .
POP register	stack .
CALL function_name(address)	CALL .
ret	.
inc destination	1 .
dec destination	1 .
add destination, value	value .
sub destination, value	value .
or destination, value	or . .
and destination, value	and . .
xor destination, value	xor . .
lea destination, source	.

- **Assembly code** **"int 0x80", "syscall"** .
  - "int" 0x80 EAX .
  - "syscall" RAX .

INT 0x80 & SYSCALL

Instructions	Meaning	Architecture
INT <Operand 1>	Call to interrupt	x86, x86_64
SYSCALL	SYStem call	x86_64

X86 Assembly/Interfacing with Linux

- x86 instruction listings
- https://en.wikibooks.org/wiki/X86_Assembly/Interfacing_with_Linux

Linux system call in assembly

- **Shellcode Assembly code system .**
- **C .**
  - .
- **.**
  - , .
- **System .**
  - System call , .
  - .
  - .
- **Ubuntu 32bit, 64bit .**

unistd_32.h

```
lazenca0x0@ubuntu:~/$ cat /usr/include/x86_64-linux-gnu/asm/unistd_32.h
#ifndef _ASM_X86_UNISTD_32_H
#define _ASM_X86_UNISTD_32_H 1

#define __NR_restart_syscall 0
#define __NR_exit 1
#define __NR_fork 2
#define __NR_read 3
#define __NR_write 4
#define __NR_open 5
#define __NR_close 6
#define __NR_waitpid 7
#define __NR_creat 8
#define __NR_link 9
#define __NR_unlink 10
#define __NR_execve 11
#define __NR_chdir 12
#define __NR_time 13
#define __NR_mknod 14
#define __NR_chmod 15
#define __NR_lchown 16
#define __NR_break 17
#define __NR_oldstat 18
#define __NR_lseek 19
#define __NR_getpid 20
...
```

unistd_64.h

```
lazenca0x0@ubuntu:~/$ cat /usr/include/x86_64-linux-gnu/asm/unistd_64.h
#ifndef _ASM_X86_UNISTD_64_H
#define _ASM_X86_UNISTD_64_H 1

#define __NR_read 0
#define __NR_write 1
#define __NR_open 2
#define __NR_close 3
#define __NR_stat 4
#define __NR_fstat 5
#define __NR_lstat 6
#define __NR_poll 7
#define __NR_lseek 8
#define __NR_mmap 9
#define __NR_mprotect 10
#define __NR_munmap 11
#define __NR_brk 12
#define __NR_rt_sigaction 13
#define __NR_rt_sigprocmask 14
#define __NR_rt_sigreturn 15
#define __NR_ioctl 16
#define __NR_pread64 17
#define __NR_pwrite64 18
#define __NR_readv 19
#define __NR_writev 20
...
```

Ubuntu 16.04

- 32bit : /usr/include/x86_64-linux-gnu/asm/unistd_32.h
- 64bit : /usr/include/x86_64-linux-gnu/asm/unistd_64.h

Save argument value in registers

- .
  - System call EAX, RAX .
  - System .
  - Kernel cdecl , "System V ABI" Calling Convention .
  - Shellcode "System V ABI" Calling Convention .

Argument

-	Register(32bit)	Register(64bit)
System call	EAX	RAX
Argument 1	EBX	RDI
Argument 2	ECX	RSI
Argument 3	EDX	RDX
Argument 4	ESI	R10
Argument 5	EDI	R8
Argument 6	EBP	R9

Calling conventions

- http://www.int80h.org/bsdasm/#alternate-calling-convention
- https://wiki.osdev.org/System_V_ABI#i386
- http://refspecs.linuxfoundation.org/elf/x86_64-abi-0.99.pdf
- https://stackoverflow.com/questions/2535989/what-are-the-calling-conventions-for-unix-linux-system-calls-on-i386-and-x86-6

Assembly Code Example("Hello, world!")

Build assembly code(32 bit)

- .
  - "Hello, world!" (0x0a) .
- .
  - ELF global _start .

ASM32.asm

```
section .data                                                   ;
        msg        db        "Hello, world!",0x0a, 0x0d        ;    ,

section        .text                                                    ;
        global        _start                                           ; ELF

_start:
        ; SYSCALL: write(1,msg,14)
        mov        eax, 4                ;     '4'  eax  .
        mov ebx, 1                ;     '1' ebx .
        mov ecx, msg        ;   ecx .
        mov edx, 14                ;   '14' edx .
        int 0x80                ;   .

        ; SYSCALL: exit(0)
        mov eax, 1                ; exit    '1' eax  .
        mov ebx, 0                ;    '0' ebx  .
        int 0x80                ;   .
```

- -f elf nasm helloworld.asm ELF (object) .
- ld a.out .

Build

```
lazenca0x0@ubuntu:~/ASM$ nasm -f elf ASM32.asm
lazenca0x0@ubuntu:~/ASM$ ld -m elf_i386 -o hello ASM32.o
lazenca0x0@ubuntu:~/ASM$ ./hello
Hello, world!
lazenca0x0@ubuntu:~/ASM$ file ./hello
./hello: ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV), statically linked, not stripped
lazenca0x0@ubuntu:~/ASM$
```

Install nasm

sudo apt-get install nasm

Build assembly code(64 bit)

ASM64.asm

```
section .data                                                                   ;
        msg db      "hello, world!",0x0a, 0x0d        ;    ,

section .text                                                                   ;
    global _start                                                           ; ELF

_start:
        ; SYSCALL: write(1,msg,14)
    mov     rax, 1                ;     '1'  rax  .
    mov     rdi, 1                ;     '1' rdi .
    mov     rsi, msg        ;   rsi .
    mov     rdx, 14                 ;   '14' rdx .
    syscall                             ;   .

        ; SYSCALL: exit(0)
    mov    rax, 60                ; exit    '60' eax  .
    mov    rdi, 0                ;    '0' ebx  .
    syscall                             ;   .
```

Build

```
lazenca0x0@ubuntu:~/ASM$ nasm -f elf64 ASM64.asm
lazenca0x0@ubuntu:~/ASM$ ld -o hello64 ASM64.o
lazenca0x0@ubuntu:~/ASM$ ./hello64
hello, world!
lazenca0x0@ubuntu:~/ASM$ file ./hello64
./hello64: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), statically linked, not stripped
lazenca0x0@ubuntu:~/ASM$
```

Change to Shellcode format

Code

- **Shellcode .**
- **.**
  - , .
  - call "helloworld" .
  - Write .
- **.**
  - Write 2 .
  - Assembly Code mov .
  - **call, ret .**
    - call call () (push) .
    - ret () eip .
    - RET .
    - , call pop ecx .
- **.**

ASM32.s

```
BITS 32                                                 ; nasm 32

call helloworld                                 ;  mark_below  call.
db "Hello, world!", 0x0a, 0x0d          ;

helloworld:
        ; ssize_t write(int fd, const void *buf, size_t count);
        pop ecx                         ;    exc .
        mov eax, 4                ;    .
        mov ebx, 1                ; STDOUT
        mov edx, 15                ;
        int 0x80                ;  : write(1,string, 14)

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

- **nasm "Hello, world!" shellocode .**

Build & Disassemble

```
lazenca0x0@ubuntu:~/ASM$ nasm ASM32.s
lazenca0x0@ubuntu:~/ASM$ ndisasm -b32 ASM32
00000000  E80F000000        call dword 0x14
00000005  48                dec eax
00000006  656C              gs insb
00000008  6C                insb
00000009  6F                outsd
0000000A  2C20              sub al,0x20
0000000C  776F              ja 0x7d
0000000E  726C              jc 0x7c
00000010  64210A            and [fs:edx],ecx
00000013  0D59B80400        or eax,0x4b859
00000018  0000              add [eax],al
0000001A  BB01000000        mov ebx,0x1
0000001F  BA0F000000        mov edx,0xf
00000024  CD80              int 0x80
00000026  B801000000        mov eax,0x1
0000002B  BB00000000        mov ebx,0x0
00000030  CD80              int 0x80
lazenca0x0@ubuntu:~/ASM$ hexdump -C ASM32
00000000  e8 0f 00 00 00 48 65 6c  6c 6f 2c 20 77 6f 72 6c  |.....Hello, worl|
00000010  64 21 0a 0d 59 b8 04 00  00 00 bb 01 00 00 00 ba  |d!..Y...........|
00000020  0f 00 00 00 cd 80 b8 01  00 00 00 bb 00 00 00 00  |................|
00000030  cd 80                                            |..|
00000032
lazenca0x0@ubuntu:~/ASM$
```

Test

- **shellcode python shellcode .**

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\x00\xba\x0f\x00\x00\x00\xcd\x80\xb8\x01\x00\x00\x00\xbb\x00\x00\x00\x0
0\xcd\x80'
>>>
```

- **shellcode .**
  - shellcode shellcode .
  - strlen() shellcode .
  - strcpy() shellcode code .
  - code "void (*function)()" .
  - function() data shellcode .

shellcode.c

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

unsigned char shellcode [] = "\xe8\x0f\x00\x00\x00Hello, world!
\n\rY\xb8\x04\x00\x00\x00\xbb\x01\x00\x00\x00\xba\x0f\x00\x00\x00\xcd\x80\xb8\x01\x00\x00\x00\xbb\x00\x00\x00\x0
0\xcd\x80";
unsigned char code[];

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

- .

Build & Run

```
lazenca0x0@ubuntu:~/ASM$ gcc -o shellcode -fno-stack-protector -z execstack --no-pie -m32 shellcode.c
test.c:5:15: warning: array 'code' assumed to have one element
 unsigned char code[];
               ^
lazenca0x0@ubuntu:~/ASM$ ./shellcode
Shellcode len : 2
Segmentation fault (core dumped)
lazenca0x0@ubuntu:~/ASM$
```

- **64bit 32bit .**

Ubuntu 64 bit

```
sudo apt-get install libx32gcc-5-dev libc6-dev-i386
```

Debugging

- .
  - strcpy() breakpoint .
  - strcpy() code (0x804a074) .
  - shellcode .

debugging

```
lazenca0x0@ubuntu:~/ASM$ gdb -q ./shellcode
Reading symbols from ./shellcode...(no debugging symbols found)...done.
gdb-peda$ disassemble main
Dump of assembler code for function main:
   0x0804846b <+0>:        lea    ecx,[esp+0x4]
   0x0804846f <+4>:        and    esp,0xfffffff0
   0x08048472 <+7>:        push   DWORD PTR [ecx-0x4]
   0x08048475 <+10>:        push   ebp
   0x08048476 <+11>:        mov    ebp,esp
   0x08048478 <+13>:        push   ecx
   0x08048479 <+14>:        sub    esp,0x14
   0x0804847c <+17>:        sub    esp,0xc
   0x0804847f <+20>:        push   0x804a040
   0x08048484 <+25>:        call   0x8048340 <strlen@plt>
   0x08048489 <+30>:        add    esp,0x10
   0x0804848c <+33>:        mov    DWORD PTR [ebp-0xc],eax
   0x0804848f <+36>:        sub    esp,0x8
   0x08048492 <+39>:        push   DWORD PTR [ebp-0xc]
   0x08048495 <+42>:        push   0x8048550
   0x0804849a <+47>:        call   0x8048320 <printf@plt>
   0x0804849f <+52>:        add    esp,0x10
   0x080484a2 <+55>:        sub    esp,0x8
   0x080484a5 <+58>:        push   0x804a040
   0x080484aa <+63>:        push   0x804a074
   0x080484af <+68>:        call   0x8048330 <strcpy@plt>
   0x080484b4 <+73>:        add    esp,0x10
   0x080484b7 <+76>:        mov    DWORD PTR [ebp-0x10],0x804a074
   0x080484be <+83>:        mov    eax,DWORD PTR [ebp-0x10]
   0x080484c1 <+86>:        call   eax
   0x080484c3 <+88>:        nop
   0x080484c4 <+89>:        mov    ecx,DWORD PTR [ebp-0x4]
   0x080484c7 <+92>:        leave
   0x080484c8 <+93>:        lea    esp,[ecx-0x4]
   0x080484cb <+96>:        ret
End of assembler dump.
gdb-peda$ b *0x080484af
Breakpoint 1 at 0x80484af
gdb-peda$ r
Starting program: /home/lazenca0x0/ASM/shell
Shellcode len : 2
Breakpoint 1, 0x080484af in main ()
gdb-peda$ x/64bx 0x804a074
0x804a074 <code>:        0x00        0x00        0x00        0x00        0x00        0x00        0x00        0x00
0x00
0x804a07c:        0x00        0x00        0x00        0x00        0x00        0x00        0x00        0x00
0x804a084:        0x00        0x00        0x00        0x00        0x00        0x00        0x00        0x00
0x804a08c:        0x00        0x00        0x00        0x00        0x00        0x00        0x00        0x00
0x804a094:        0x00        0x00        0x00        0x00        0x00        0x00        0x00        0x00
0x804a09c:        0x00        0x00        0x00        0x00        0x00        0x00        0x00        0x00
0x804a0a4:        0x00        0x00        0x00        0x00        0x00        0x00        0x00        0x00
0x804a0ac:        0x00        0x00        0x00        0x00        0x00        0x00        0x00        0x00
gdb-peda$
gdb-peda$ x/64bx 0x804a040
0x804a040 <shellcode>:        0xe8        0x0f        0x00        0x00        0x00        0x48
0x65        0x6c
0x804a048 <shellcode+8>:        0x6c        0x6f        0x2c        0x20        0x77        0x6f
0x72        0x6c
0x804a050 <shellcode+16>:        0x64        0x21        0x0a        0x0d        0x59        0xb8
0x04        0x00
0x804a058 <shellcode+24>:        0x00        0x00        0xbb        0x01        0x00        0x00
0x00        0xba
0x804a060 <shellcode+32>:        0x0f        0x00        0x00        0x00        0xcd        0x80
0xb8        0x01
0x804a068 <shellcode+40>:        0x00        0x00        0x00        0xbb        0x00        0x00
0x00        0x00
0x804a070 <shellcode+48>:        0xcd        0x80        0x00        0x00        0xe8        0x0f
0x00        0x00
0x804a078:        0x00        0x00        0x00        0x00        0x00        0x00        0x00        0x00
gdb-peda$
```

- **shellcode ?**
  - shellcode null byte(0x00) .
    - null byte(0x00) .
    - strlen() shellcode 2 , strcpy() null byte(0x00) code .
  - , shellcode code .
  - shellcode null byte .

Error point

```
gdb-peda$ ni

0x080484b4 in main ()
gdb-peda$ x/32bx 0x804a074
0x804a074 <code>:        0xe8        0x0f        0x00        0x00        0x00        0x00        0x00        0x00
0x00
0x804a07c:        0x00        0x00        0x00        0x00        0x00        0x00        0x00        0x00        0x00
0x804a084:        0x00        0x00        0x00        0x00        0x00        0x00        0x00        0x00        0x00
0x804a08c:        0x00        0x00        0x00        0x00        0x00        0x00        0x00        0x00        0x00
gdb-peda$
```

Remove null byte

- **null byte .**
  - 0xE80F000000 : call dword 0x14
  - 0xB804000000 : mov eax, 4
  - 0xBB01000000 : mov ebx, 1
  - 0xBA0F000000 : mov edx, 15
  - 0xB801000000 : mov eax,1
  - 0xBB00000000 : mov ebx,0

Null byte

```
lazenca0x0@ubuntu:~/ASM$ ndisasm -b32 ASM32
00000000  E80F000000        call dword 0x14
00000005  48                dec eax
00000006  656C              gs insb
00000008  6C                insb
00000009  6F                outsd
0000000A  2C20              sub al,0x20
0000000C  776F              ja 0x7d
0000000E  726C              jc 0x7c
00000010  64210A            and [fs:edx],ecx
00000013  0D59B80400        or eax,0x4b859
00000018  0000              add [eax],al
0000001A  BB01000000        mov ebx,0x1
0000001F  BA0F000000        mov edx,0xf
00000024  CD80              int 0x80
00000026  B801000000        mov eax,0x1
0000002B  BB00000000        mov ebx,0x0
00000030  CD80              int 0x80
lazenca0x0@ubuntu:~/ASM$
```

Call instruction

- **call .**
  - call dword(32bits) .
  - "0x14" null byte .
- .
  - jmp helloworld last .
  - last helloworld , .

ASM32-2.s

```
BITS 32                         ; nasm 32

jmp short last        ;   .
helloworld:
        ; ssize_t write(int fd, const void *buf, size_t count);
        pop ecx         ;    exc .
        mov eax, 4        ;    .
        mov ebx, 1        ; STDOUT
        mov edx, 15        ;
        int 0x80        ;  : write(1,string, 14)

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

last:
        call helloworld        ;      .
        db "Hello, world!", 0x0a, 0x0d        ;
```

- **call null byte .**
  - jmp short 0x20 byte .
    - jmp dword(32bits) .
    - jmp short -128 127 .
  - call helloworld -0x22(0x2 - 0x24) .
    - -0x22 2 (0xFFFFFFDD) .
  - **, 'jmp short', 'call ' null byte .**

ndisasm -b32 ASM32-2

```
lazenca0x0@ubuntu:~/ASM$ nasm ASM32-2.s
lazenca0x0@ubuntu:~/ASM$ ndisasm -b32 ASM32-2
00000000  EB1E              jmp short 0x20
00000002  59                pop ecx
00000003  B804000000        mov eax,0x4
00000008  BB01000000        mov ebx,0x1
0000000D  BA0F000000        mov edx,0xf
00000012  CD80              int 0x80
00000014  B801000000        mov eax,0x1
00000019  BB00000000        mov ebx,0x0
0000001E  CD80              int 0x80
00000020  E8DDFFFFFF        call dword 0x2
00000025  48                dec eax
00000026  656C              gs insb
00000028  6C                insb
00000029  6F                outsd
0000002A  2C20              sub al,0x20
0000002C  776F              ja 0x9d
0000002E  726C              jc 0x9c
00000030  64210A            and [fs:edx],ecx
00000033  0D                db 0x0d
lazenca0x0@ubuntu:~/ASM$
```

Register

- **call null byte Register .**
- **Register null byte .**
  - 64 bit, 32bit, 16 bit null byte .

Register

64 bit	RAX	RBX	RCX	RDX	RSP	RBP	RSI	RDI
32 bit	EAX	EBX	ECX	EDX	ESP	EBP	ESI	EDI
16 bit	AX	BX	CX	DX	SP	BP	SI	DI
8 bit	AH/AL	BH/BL	CH/CL	DH/DL				

- L : (Low byte),H : (High byte)

A null byte generated according to the size of the register

Assessmbly code	Machine code
mov eax,0x4	B8 04 00 00 00
mov ax,0x4	66 B8 04 00
mov al,0x4	B0 04

- **shellcode 64 bit, 32 0 .**
  - shellcode shellcode .
- .
  - write ebx, ecx, edx .
  - null byte bl, cl, dl .
  - shellcode .
    - 3 null byte .

Problem when register value is not initialized

	Code	EBX
shellcode	-	0xdeaddead
shellcode	mov bl,0x4	0xdeadde04

- , .
- **sub**
  - sub .
  - sub .
  - sub OF, SF, ZF, AF, PF, CF flag .
    - .
- **xor**
  - xor 2 (exclusive OR) .
  - (exclusive OR) 0 .
  - OF, CF flag , SF, ZF, PF flag .
    - AF flag .
  - xor sub flag xor .

Initialize register value

Assessmbly code	Machine code
sub eax, eax	29 C0
xor eax,eax	31 C0

- .

RemoveNullbyte.s

```
BITS 32                         ; nasm 32

jmp short last        ;   .
helloworld:
        ; ssize_t write(int fd, const void *buf, size_t count);
        pop ecx         ;    exc .
        xor eax,eax       ; eax   0 .
        mov al, 4        ;    .
        xor ebx,ebx       ; ebx   0 .
        mov bl, 1        ; STDOUT
        xor edx,edx       ; edx   0 .
        mov dl, 15       ;
        int 0x80        ;  : write(1,string, 14)

        ; void _exit(int status);
        mov al,1        ;exit
        xor ebx,ebx       ;Status = 0
        int 0x80        ; : exit(0)

last:
        call helloworld        ;      .
        db "Hello, world!", 0x0a, 0x0d        ;
```

- **Null byte .**

Removed Null byte

```
lazenca0x0@ubuntu:~/ASM$ nasm RemoveNullbyte.s
lazenca0x0@ubuntu:~/ASM$ ndisasm RemoveNullbyte
00000000  EB15              jmp short 0x17
00000002  59                pop cx
00000003  31C0              xor ax,ax
00000005  B004              mov al,0x4
00000007  31DB              xor bx,bx
00000009  B301              mov bl,0x1
0000000B  31D2              xor dx,dx
0000000D  B20F              mov dl,0xf
0000000F  CD80              int 0x80
00000011  B001              mov al,0x1
00000013  31DB              xor bx,bx
00000015  CD80              int 0x80
00000017  E8E6FF            call word 0x0
0000001A  FF                db 0xff
0000001B  FF4865            dec word [bx+si+0x65]
0000001E  6C                insb
0000001F  6C                insb
00000020  6F                outsw
00000021  2C20              sub al,0x20
00000023  776F              ja 0x94
00000025  726C              jc 0x93
00000027  64210A            and [fs:bp+si],cx
0000002A  0D                db 0x0d
lazenca0x0@ubuntu:~/ASM$
```

- .

shellcode2.c

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

unsigned char shellcode [] =
"\xeb\x15\x59\x31\xc0\xb0\x04\x31\xdb\xb3\x01\x31\xd2\xb2\x0f\xcd\x80\xb0\x01\x31\xdb\xcd\x80\xe8\xe6\xff\xff\xf
fHello, world!\n\r";
unsigned char code[] = "";

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

- **"Hello, world!" .**

Build & Run

```
lazenca0x0@ubuntu:~/ASM$ gcc -o shellcode2 -fno-stack-protector -z execstack --no-pie -m32 shellcode2.c
lazenca0x0@ubuntu:~/ASM$ ./shellcode2
Shellcode len : 43
Hello, world!
lazenca0x0@ubuntu:~/ASM$
```

Related site

- https://syscalls.kernelgrok.com/
- https://w3challs.com/syscalls/
- https://en.wikibooks.org/wiki/X86_Assembly/Data_Transfer
- https://en.wikipedia.org/wiki/INT_(x86_instruction)
- https://en.wikibooks.org/wiki/X86_Assembly/Interfacing_with_Linux
- https://en.wikibooks.org/wiki/X86_Assembly/X86_Architecture
- http://blog.rchapman.org/posts/Linux_System_Call_Table_for_x86_64/
- http://www.cs.utexas.edu/~bismith/test/syscalls/syscalls64_orig.html
- http://0xax.blogspot.jp/2014/08/say-hello-to-x64-assembly-part-1.html
- https://en.wikipedia.org/wiki/Compiler
- Book : https://en.wikipedia.org/wiki/Hacking:_The_Art_of_Exploitation

Comments

Unknown macro: 'html'

Unknown macro: 'html'