###Reducing Instruction Size and Removing Nulls

1. In Shellcoding
    - Reduce the instruction size
    - Remove nulls (Null signifies end of string)

###Data Types

- Byte - 8 bits
- Word - 16 bits
- Double word - 32 bits
- Quad word - 64 bits
- Double Quad Word - 128 bits

###Nasm..

- Case sensitive syntax
- Accessing memory refernce with []
    - `message db 0xAA, 0xBB, 0xCC ...` (defines series of bytes with label , `message`)
    - `mov rax, message` ← moves address into `rax`
    - `mov rax, [message]` ← moves value into `rax`

Defining Initialized Data in NASM

Feature	Description
`db 0x55`	Just the byte 0x55
`db 0x55, 0x56, 0x57`	ghout the database.
`db 'a', 0x55`	character constants are OK.
`db 'hello', 13,10, '$'`	so are string contants
`dw 0x1234`	`0x34` `0x12`
`dw 'a'`	`0x61` `0x00` (It's just a number)
`dw 'ab'`	`0x61` `0x62` (character constant)
`dw 'abc'`	`0x61` `0x62` `0x63` (string)
`dw 0x12345678`	`0x78` `0x56` `0x34` `0x12`

Defining Uninitialized Data in NASM

Feature	Description
`buffer: resb 64`	Reserve 64 bytes
`wordvar: resw 1`	Reserve a word

Special Tokens

1. `$` - evaulates to the current line
2. `$$` - evaulates to the beginning of current section

Endianess

- Order in which bytes are stored

Low address **High address**

Address	0	1	2	3	4	5	6	7
Little-endian	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
Big-endian	Byte 7	Byte 6	Byte 5	Byte 4	Byte 3	Byte 2	Byte 1	Byte 0

Memory content

0x11	0x22	0x33	0x44	0x55	0x66	0x77	0x88

64 bit value in Little-endian

```
0x8877665544332211
```

64 bit value in Big-endian

```
0x1122334455667788
```

```
x86 and x86_x64  both uses Littl-endian format
```

Assembly Code

Sample code

```
global _start
section .text

;start like main
_start:
        mov rax, 1     ;  1 for write to screen
        mov rdi, 1     ;  1 for write to screen
        mov rsi, hello_world
        mov rdx, length
        syscall
        ; exit gracefully
        mov rax, 60     ;60 for exit
        mov rdi, 11     ; exit code can be anything 0 OR 1 OR any
        syscall  ; system call

section .data
    hello_world: db 'Hello World to the Pentester academy'
    length: equ $-hello_world ; calculate the length of the hello_world
```

- complie steps

```
    nasm -felf64 HelloWorld.nasm -o HelloWorld.o
    ld HelloWorld.o -o HelloWorld     //linking
    ./HelloWorld
```

- `rax` takes 48 bytes
- comand used to display through object dump
  `objdump -M intel -d HelloWorld.o`

```
HelloWorld.o:     file format elf64-x86-64


Disassembly of section .text:

0000000000000000 <_start>:
   0:   b8 01 00 00 00          mov    eax,0x1
   5:   bf 01 00 00 00          mov    edi,0x1
   a:   48 be 00 00 00 00 00    movabs rsi,0x0
  11:   00 00 00
  14:   ba 24 00 00 00          mov    edx,0x24
  19:   0f 05                   syscall
  1b:   b8 3c 00 00 00          mov    eax,0x3c
  20:   bf 0b 00 00 00          mov    edi,0xb
  25:   0f 05                   syscall
```

- Variables (Datatypes)

```
global _start
section .text

;start like main
_start:
        mov rax, 1      ;  1 for write
        mov rdi, 1      ;  1 for write
        mov rsi, hello_world
        mov rdx, length
        syscall

        mov rax, var4
        mov rax, [var4]

        ; exit gracefully
        mov rax, 60     ;60 for exit
        mov rdi, 11     ; exit code can be anything 0 OR 1 OR any
        syscall  ; system call

section .data
    hello_world: db 'Hello World to the Pentester academy'
    length: equ $-hello_world ; calculate the length of the hello_world
```

```
    var1: db 0x11, 0x22  ; define bytes
    var2: dw 0x3344      ; word
    var3: dd 0xaabbccdd    ; 4 bytes
    var4: dq 0xaabbccdd11223344 ; 8bytes

    repaet_buffer: times 128 db 0xAA

section .bss ; reserving uninitiliazed datra
    buffer: resb 64 ; reserve 64 bytes
```

GDB TUI Mode

- TUI (Test User Interface)

`gdb -q ./HelloWorld -tui` to open in TUI mode

MOV

- Most common instruiction in ASM
- Allowed directions
    - Between Registers
    - Memmory to Register and Register to Memory
    - Immediate Data to Register
    - Immediate Data to Memory

LEA

- Moad Effective address - load pointer values
- `LEA RAX, [label]`

XCHG

- Exchange (swap) values
- `XCHG Register, Register`
- `XCHG Register, Memory`

The Stack

- A temporary location in memory where we can store data, while the program is running
- High level programming languages like C , make extensive use of stack
- Stack operations consists of two operations
    - `PUSH` - insert data into the stack
    - `POP` - Remove data from the stack


Stack as Lifo

Sample `Stack.nasm` program

```
;Purpose: Stack instruction in 64 bit CPU

global _start

section .text
_start:
    mov rax, 0x1122334455667788 ; move immediate value into rax
    push rax   ; push the value contained in rax to stack

    push sample   ; address reeferenceby sample i.e db 0xaa....

    push qword [sample]  ; pickup 8 bytes, interpret as qword  and push into
stack

    pop r15
    pop r14
    pop rbx

    ; exit program

    mov rax, 0x3c
    mov rdi, 0
    syscall

section .data

sample : db 0xaa, 0xbb, 0xcc, 0xdd, 0xee, 0xff, 0x11, 0x22
```

Compile: `nasm Stack.nasm -o Stack.o`
Link: `ld Stack.o -o Stack`
Opening in tui mode: `gdb -q ./Stack -tui`

Shellcoding

What is Shellcode?

- machine code with specific purpose
    - spawn a local shell
    - Bind to port and spawn shell
    - create a new account

- Can be executed by the CPU directly - no further assembling /linking or seperate compiling required

How is Shellcode delivered?

- Part of an exploit
  - Size of shellcode important (smaller size = better)
  - Bad characters is a concern
    - 0x00 most common one
- Added into an executable
  - run as seperated thread
  - replace executable functionality
  - Size of shellcode not a concern

Testing Shellcode

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

unsigned char code[] = \"SHELLCODE";

main() {
    printf("Shellcode Lenggth: %d\n", strlen(code));
    int (*ret)() = (int(*)())code;
    ret();
}
```

```
/*
 * Execute /bin/sh - 27 bytes
 * Dad` <3 baboon
;rdi            0x4005c4 0x4005c4
;rsi            0x7fffffffdf40   0x7fffffffdf40
;rdx            0x0      0x0
;gdb$ x/s $rdi
;0x4005c4:        "/bin/sh"
;gdb$ x/s $rsi
;0x7fffffffdf40:  "\304\005@"
;gdb$ x/32xb $rsi
;0x7fffffffdf40: 0xc4    0x05    0x40    0x00    0x00    0x00    0x00    0x00
;0x7fffffffdf48: 0x00    0x00    0x00    0x00    0x00    0x00    0x00    0x00
;0x7fffffffdf50: 0x00    0x00    0x00    0x00    0x00    0x00    0x00    0x00
;0x7fffffffdf58: 0x55    0xb4    0xa5    0xf7    0xff    0x7f    0x00    0x00
;
;=> 0x7ffff7aeff20 <execve>:     mov    eax,0x3b
;   0x7ffff7aeff25 <execve+5>:   syscall
;

main:
    ;mov rbx, 0x68732f6e69622f2f
    ;mov rbx, 0x68732f6e69622fff
    ;shr rbx, 0x8
    ;mov rax, 0xdeadbeefcafe1dea
    ;mov rbx, 0xdeadbeefcafe1dea
```

```
    ;mov rcx, 0xdeadbeefcafe1dea
    ;mov rdx, 0xdeadbeefcafe1dea
    xor eax, eax
    mov rbx, 0xFF978CD091969DD1
    neg rbx
    push rbx
    ;mov rdi, rsp
    push rsp
    pop rdi
    cdq
    push rdx
    push rdi
    ;mov rsi, rsp
    push rsp
    pop rsi
    mov al, 0x3b
    syscall
 */

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

char code[] =
"\x31\xc0\x48\xbb\xd1\x9d\x96\x91\xd0\x8c\x97\xff\x48\xf7\xdb\x53\x54\x5f\x99\x5
2\x57\x54\x5e\xb0\x3b\x0f\x05";

int main()
{
    printf("len:%d bytes\n", strlen(code));
    (*(void(*)()) code)();
    return 0;
}
```