SecurityTube Linux Assembly Expert (SLAE^{32})

SecurityTube Linux Assembly Expert

Training: http://www.SecurityTube-Training.com

Community: http://www.SecurityTube.net

Vivek Ramachandran
SWSE, SMFE, SPSE, SGDE, SISE, SLAE^{32} Course Instructor

SecurityTube Linux Assembly Expert

Course Introduction

Vivek Ramachandran
SWSE, SMFE, SPSE, SGDE, SISE, SLAE^{32} Course Instructor

http://SecurityTube-Training.com

SecurityTube Linux Assembly Expert

1. What is the SLAE?

The **SecurityTube Linux Assembly Expert (SLAE)** is an online course and certification which focuses on teaching the basics of 32-bit assembly language for the Intel Architecture (IA-32) family of processors on the Linux platform and applying it to Infosec. Once we are through with the basics, we will look at writing shellcode, encoders, decoders, crypters and other advanced low level applications.

Why learn Assembly Language?

Assembly Language is the gateway to understanding exploitation techniques, reverse engineering, shellcoding and other low level fields in information security. Without a good grasp and knowledge of assembly language it will not be possible to master these fields effectively.

This covers starts from the very basics of programming in Assembly Language and does not expect students to have prior programming experience.

http://securitytube-training.com/online-courses/securitytube-linux-assembly-expert/

Course Syllabus – Assembly Basics

A non-exhaustive list of topics to be covered include:

- Computer Architecture Basics
- IA-32/64 Family
- Compilers, Assemblers and Linkers
- CPU Modes and Memory Addressing
- Tools of the trade
  - Nasm, Ld, Objdump, Ndisasm etc.
- IA-32 Assembly Language
  - Registers and Flags
  - Program Structure for use with nasm
  - Data Types
  - Data Movement Instructions
  - Arithmetic instructions
  - Reading and Writing from memory
  - Conditional instructions
  - Strings and Loops
  - Interrupts, Traps and Exceptions
  - Procedures, Prologues and Epilogues
  - Syscall structure and ABI for Linux
  - Calling standard library functions
  - FPU instructions
  - MMX, SSE, SSE2 etc. instruction sets

Application to Infosec

- Shellcoding on Linux
  - Execution environment
  - Exit and Execve shellcode
  - Bind Shell and Reverse TCP
  - Staged Shellcode
  - Egg Hunter
  - Using 3rd party shellcode
  - Simulating shellcode
    - locating syscalls
    - graphing shellcode execution
- Encoders, Decoders and Crypters on Linux
  - Purpose of encoding and crypting
  - XOR encoders
  - Custom encoding
    - Random sequencing and scrambling
    - mapping functions
  - Crypters
- Polymorphism
  - Why polymorphism?
  - Polymorphic engines
  - Techniques and Tools
- Roadmap for Further Study
- Mock Exam and Certification Exam details

Registered Students Benefit

A registered student will get the following:

- HD Download of Course Theory Videos
- HD Download of Course Exercise Videos
- PDF Slides of the full course
- All Code samples used in the course
- Mock Exam
- Certification Exam
- PDF copy of certificate if you pass the exam

Please note that there is no student forum associated with this low-priced course.

Future Courses

- 64-bit Assembly on Linux
- 32/64-bit Assembly on Windows
- ARM Assembly

SecurityTube Linux Assembly Expert (SLAE)

SecurityTube Linux Assembly Expert

http://www.securitytube.net

Vivek Ramachandran
Course Instructor

Notice

This video is part of the **SecurityTube Linux Assembly Expert** course. A limited number of videos in this course will be released on SecurityTube.net

For full access to all theory and exercise videos, PDF slides, Code snippets etc. Please register as a student by visiting:

http://securitytube-training.com

Module 1: 32-Bit ASM on Linux

1. What is Assembly Language?

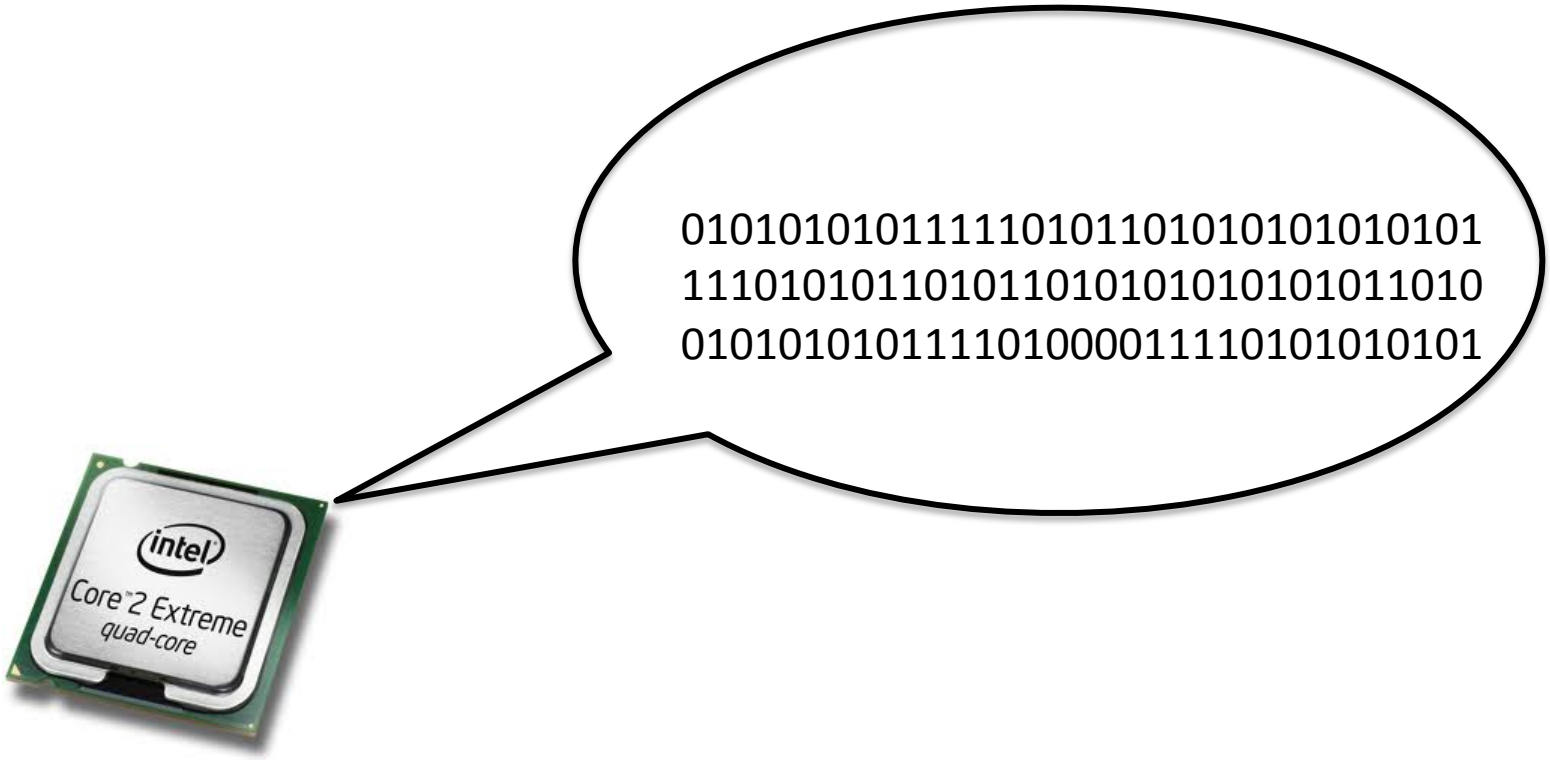
Vivek Ramachandran
SWSE, SMFE, SPSE, SGDE, SISE, SLAE^{32} Course Instructor

http://SecurityTube-Training.com

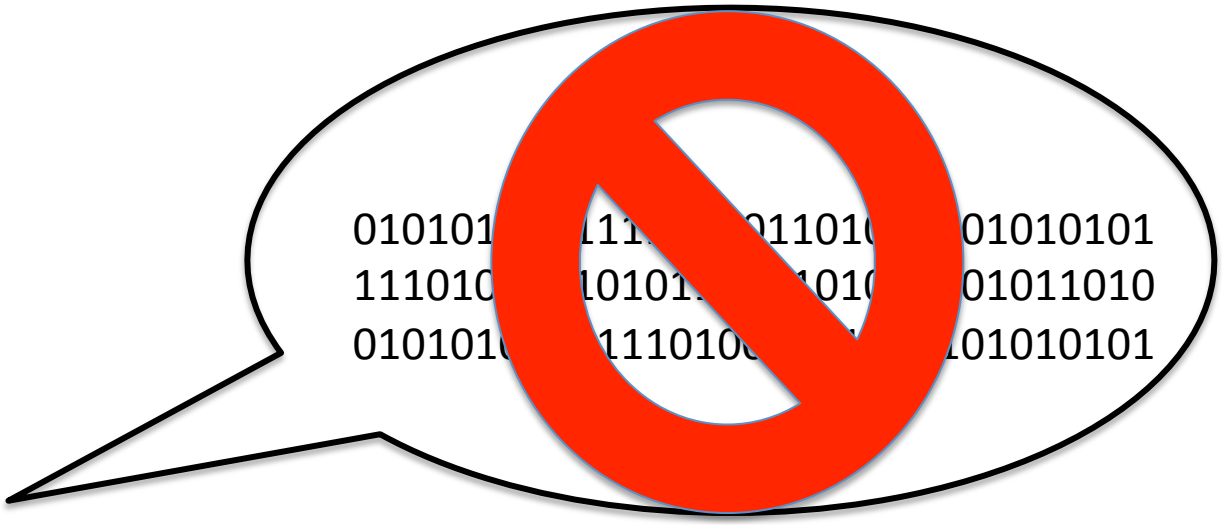
What is Assembly Language?

- Low-level programming language
- Communicate with microprocessor
- Specific to the processor family
- An almost one-one correspondence with machine code

I only speak binary!



Humans cannot speak binary



Assembly Language

Assembly Language

```
mov eax, ebx
xor  eax, eax
add  eax, 0xff
```

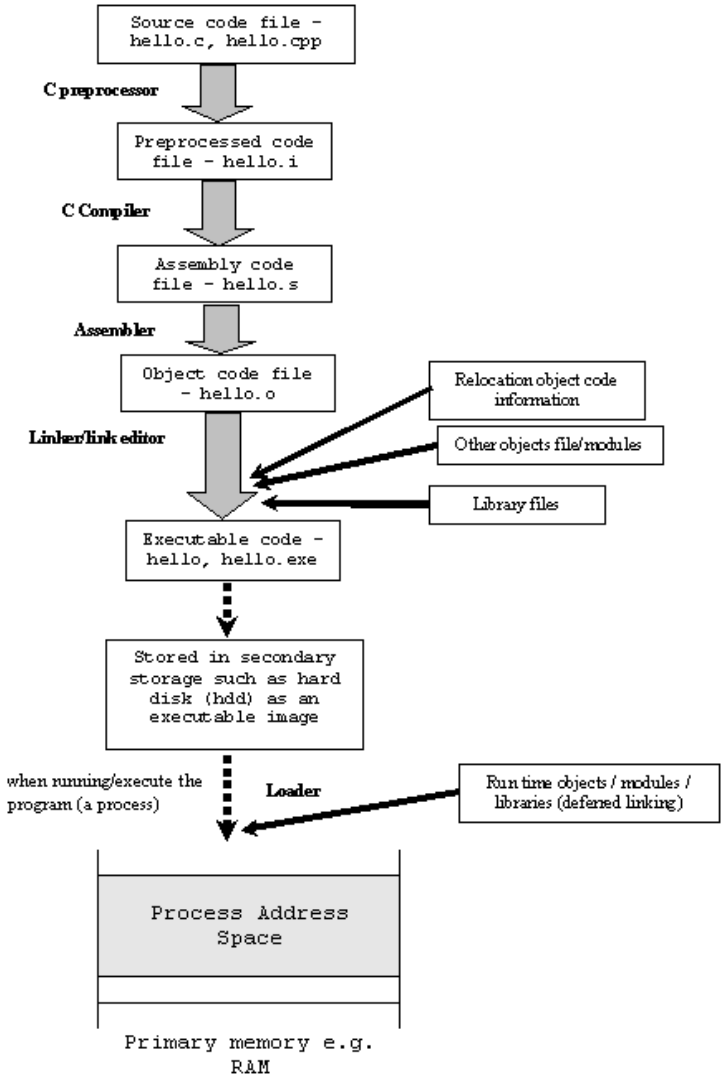
Assembler + Linker

Translator

Machine Language

```
010110100101
111010101010
101010101010
```

Correlation with HLLs



http://www.tenouk.com/ModuleW_files/ccompilerlinker001.png

Different Processors – Different Assembly Language

- Intel
- ARM
- MIPS

Intel Architecture

- IA-32
- IA-64

SecurityTube Linux Assembly Expert^{32}

- IA-32 Assembly on the Linux OS

- Future courses:
  - IA-64 on Linux
  - IA-32 on Windows
  - IA-64 on Windows
  - ARM Assembly

Why IA-32?

- Large number of machines out there still running IA-32
- Logical progression to IA-64
- Shellcoding, Encoders, Decoders, Packers etc. implementation difference

Exercise 1.1: Lab Setup

- Ubuntu 12.04 LTS 32-bit Edition
- Installed in Virtualbox

Please Register for this course to receive the solution video for this exercise.
http://SecurityTube-Training.com

Exercise 1.2: Understanding your CPU

- Find CPU details on the Ubuntu System
- How do you know if you are on a 32/64 bit CPU?
- How do you know your CPUs additional capabilities such as FPU, MMX, SSE, SSE2 etc.

Please Register for this course to receive the solution video for this exercise.
http://SecurityTube-Training.com

SecurityTube Linux Assembly Expert (SLAE^{32})

SecurityTube Linux Assembly Expert

Training: http://www.SecurityTube-Training.com

Community: http://www.SecurityTube.net

Vivek Ramachandran
SWSE, SMFE, SPSE, SGDE, SISE, SLAE^{32} Course Instructor

Module 1: 32-Bit ASM on Linux

Exercise 1.1
Topic: What is Assembly Language?

Vivek Ramachandran
SWSE, SMFE, SPSE, SGDE, SISE, SLAE^{32} Course Instructor

http://SecurityTube-Training.com

Exercise 1.1: Lab Setup

- Ubuntu 12.04 LTS 32-bit Edition
- Installed in Virtualbox

Ubuntu

- 32-bit Ubuntu Desktop Edition 12.10 used for the course

http://www.ubuntu.com/download/desktop

- Install Virtualbox

https://www.virtualbox.org/

Installation

- Nasm

- Code files:
  - SLAE-Code.zip

SecurityTube Linux Assembly Expert (SLAE^{32})

SecurityTube Linux Assembly Expert

Training: http://www.SecurityTube-Training.com

Community: http://www.SecurityTube.net

Vivek Ramachandran
SWSE, SMFE, SPSE, SGDE, SISE, SLAE^{32} Course Instructor

Module 1: 32-Bit ASM on Linux

Exercise 1.2
Topic: What is Assembly Language?

Vivek Ramachandran
SWSE, SMFE, SPSE, SGDE, SISE, SLAE^{32} Course Instructor

http://SecurityTube-Training.com

Exercise 1.2: Understanding your CPU

- Find CPU details on the Ubuntu System
- How do you know if you are on a 32/64 bit CPU?
- How do you know your CPUs additional capabilities such as FPU, MMX, SSE, SSE2 etc.

Find CPU Details

```
securitytube@securitytube-VirtualBox:~$ lscpu
Architecture:          i686
CPU op-mode(s):        32-bit, 64-bit
Byte Order:            Little Endian
CPU(s):                1
On-line CPU(s) list:   0
Thread(s) per core:    1
Core(s) per socket:    1
Socket(s):             1
Vendor ID:             GenuineIntel
CPU family:            6
Model:                 37
Stepping:              5
CPU MHz:               2534.821
BogoMIPS:              5069.64
L1d cache:             32K
L1d cache:             32K
L2d cache:             6144K
securitytube@securitytube-VirtualBox:~$
```

/proc/cpuinfo

```
securitytube@securitytube-VirtualBox:~$ cat /proc/cpuinfo
processor       : 0
vendor_id       : GenuineIntel
cpu family      : 6
model           : 37
model name      : Intel(R) Core(TM) i5 CPU       M 540  @ 2.53GHz
stepping        : 5
cpu MHz         : 2534.821
cache size      : 6144 KB
fdiv_bug        : no
hlt_bug         : no
f00f_bug        : no
coma_bug        : no
fpu             : yes
fpu_exception   : yes
cpuid level     : 5
wp              : yes
flags           : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush mmx
fxsr sse sse2 syscall nx rdtscp lm constant_tsc up pni monitor ssse3 lahf_lm
bogomips        : 5069.64
clflush size    : 64
cache_alignment : 64
address sizes   : 36 bits physical, 48 bits virtual
power management:

securitytube@securitytube-VirtualBox:~$
```

SecurityTube Linux Assembly Expert (SLAE^{32})

SecurityTube Linux Assembly Expert

Training: http://www.SecurityTube-Training.com

Community: http://www.SecurityTube.net

Vivek Ramachandran
SWSE, SMFE, SPSE, SGDE, SISE, SLAE^{32} Course Instructor

Notice

This video is part of the **SecurityTube Linux Assembly Expert** course. A limited number of videos in this course will be released on SecurityTube.net

For full access to all theory and exercise videos, PDF slides, Code snippets etc. Please register as a student by visiting:

http://securitytube-training.com

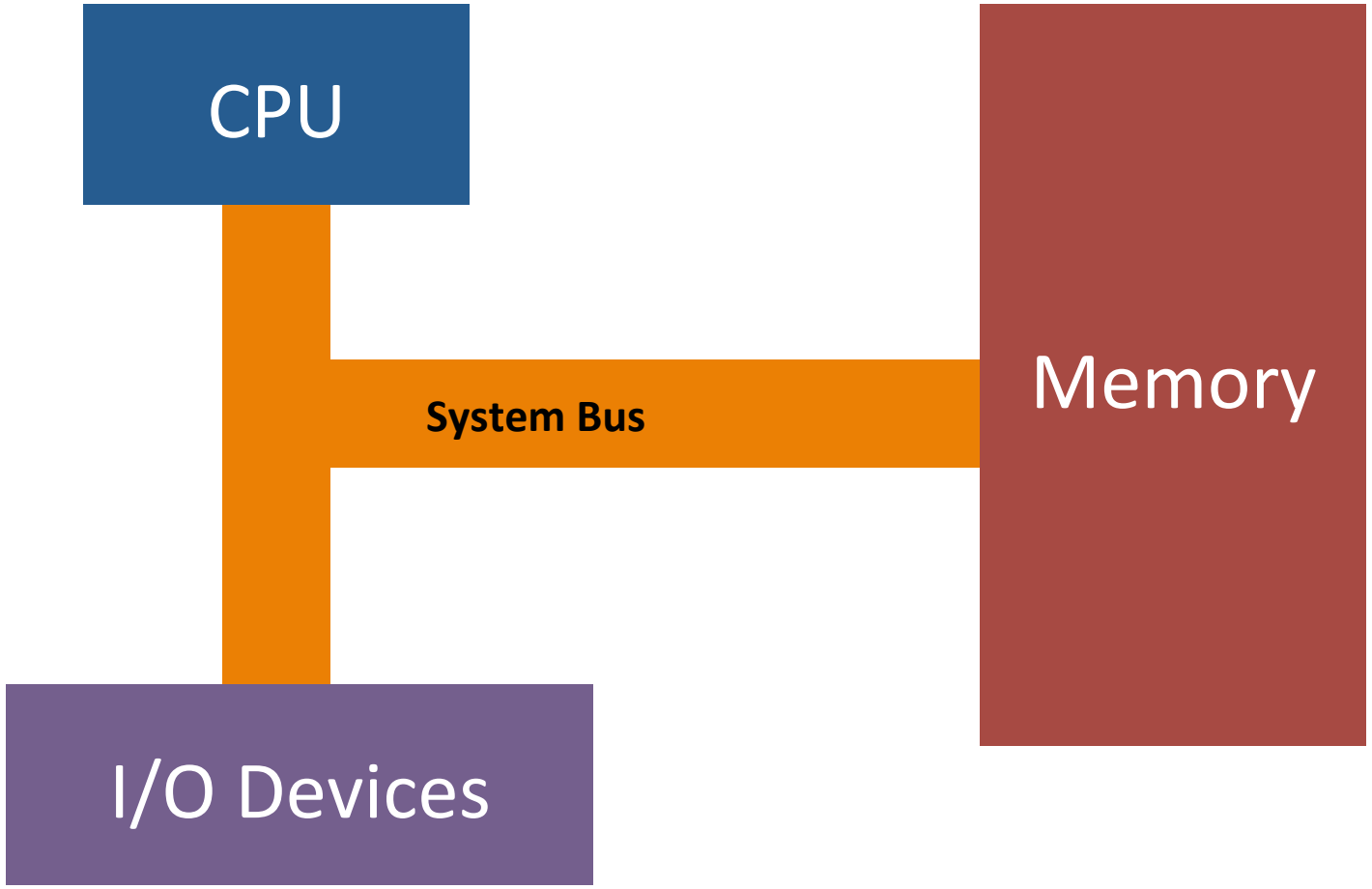
Module 1: 32-Bit ASM on Linux

2. IA-32 Architecture

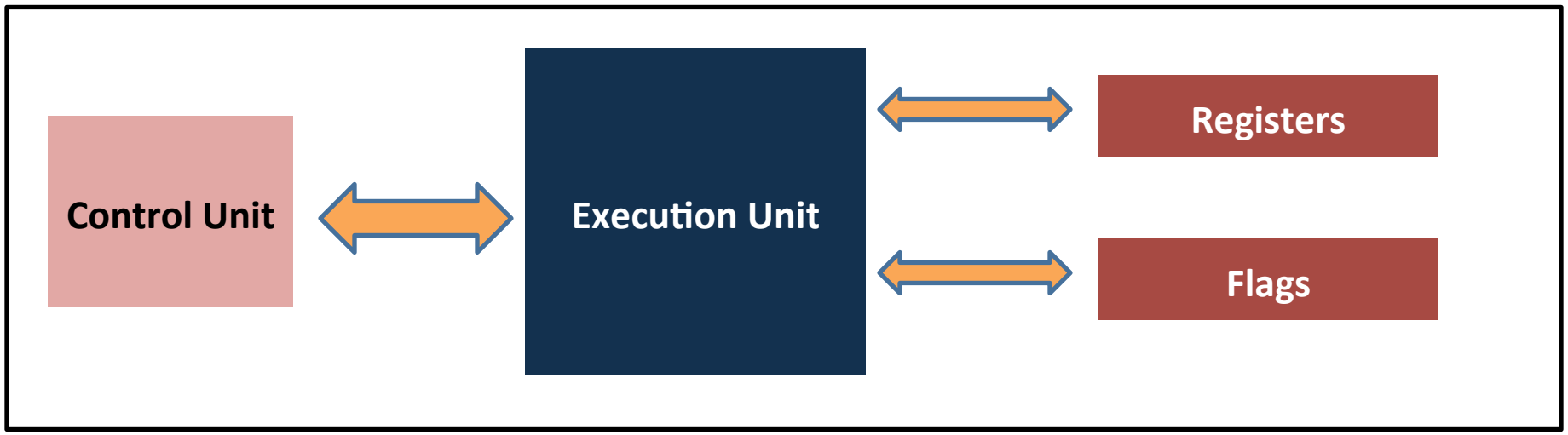
Vivek Ramachandran
SWSE, SMFE, SPSE, SGDE, SISE, SLAE^{32} Course Instructor

http://SecurityTube-Training.com

System Organization Basics

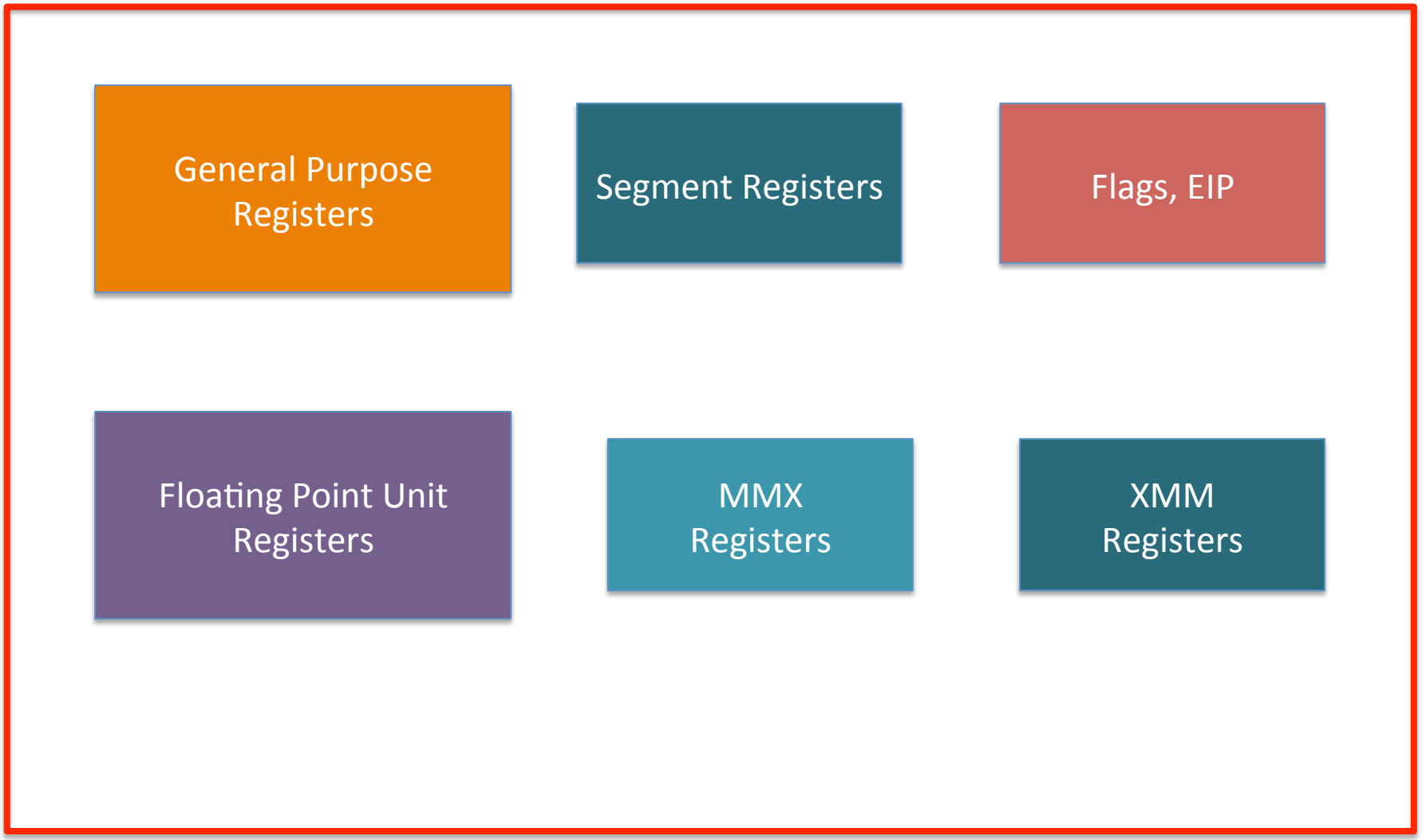


CPU

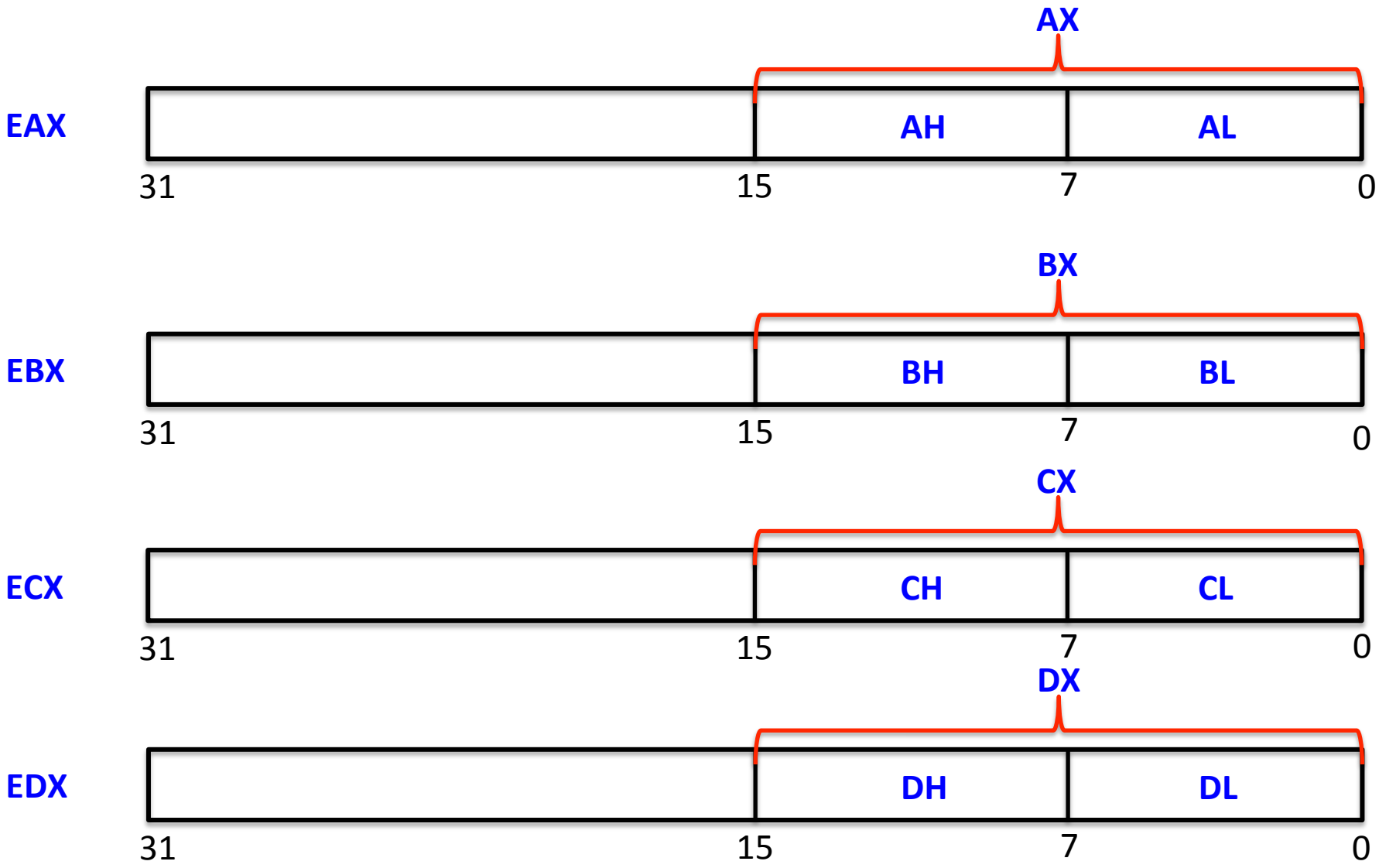


- **Control Unit** – Retrieve / Decode instructions, Retrieve / Store data in memory
- **Execution Unit** – Actual execution of instruction happens here
- **Registers** - Internal memory locations used as “variables”
- **Flags** – Used to indicate various “event” when execution is happening

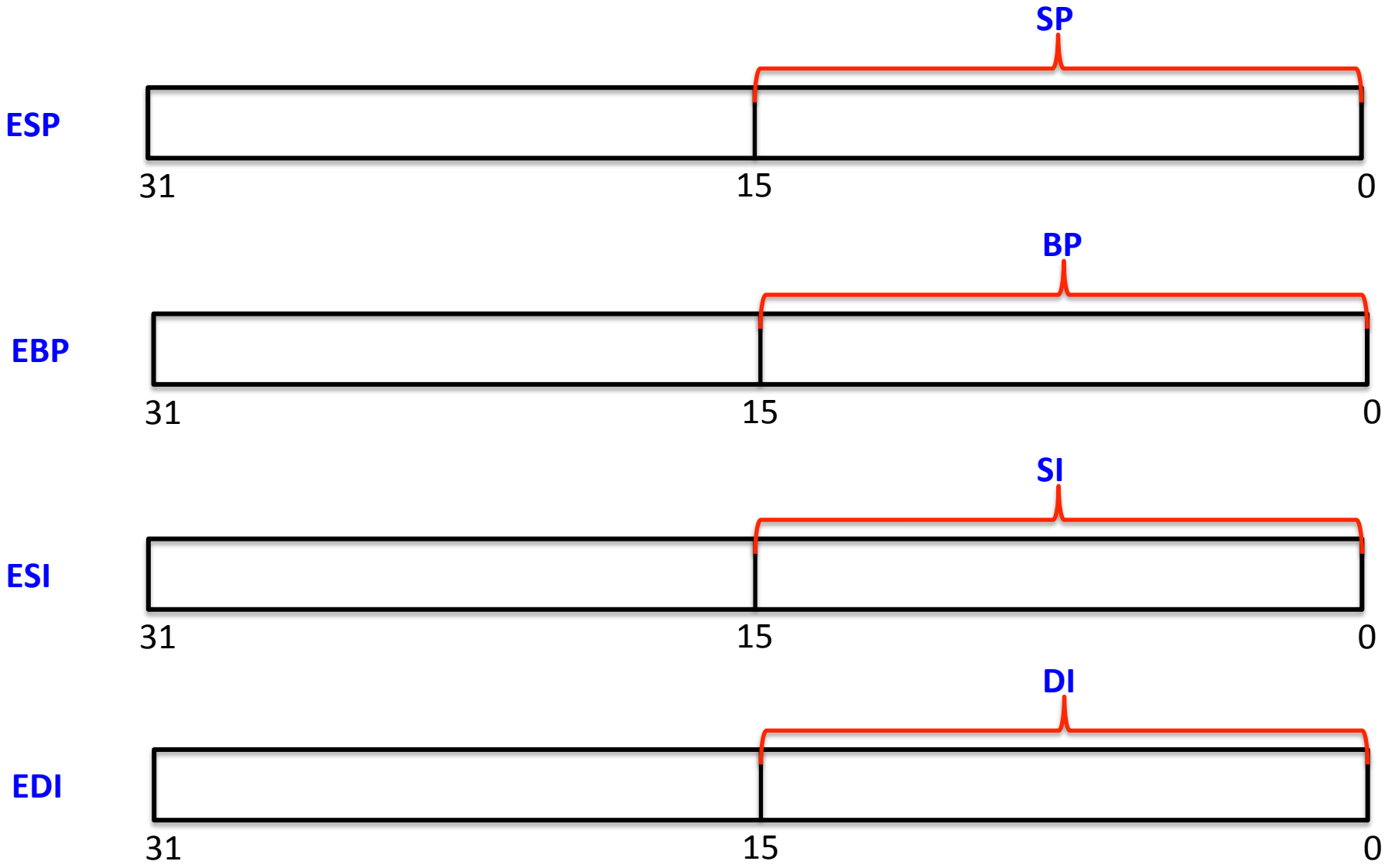
IA-32 Registers (Logical Diagram)



General Purpose Registers



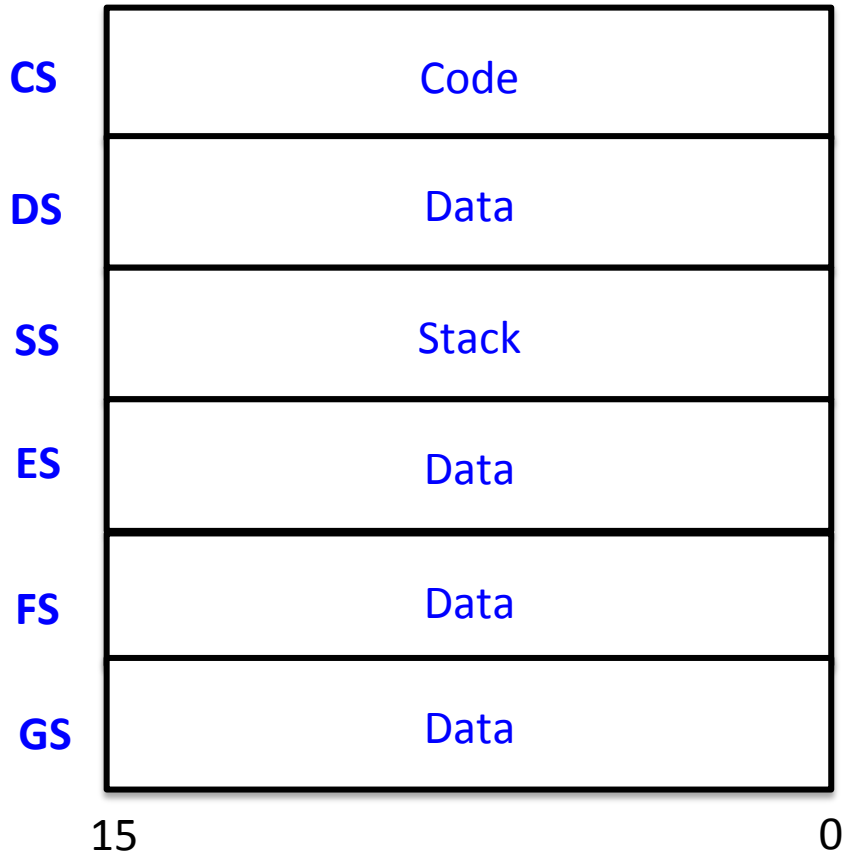
General Purpose Registers



GPR Common Functionality

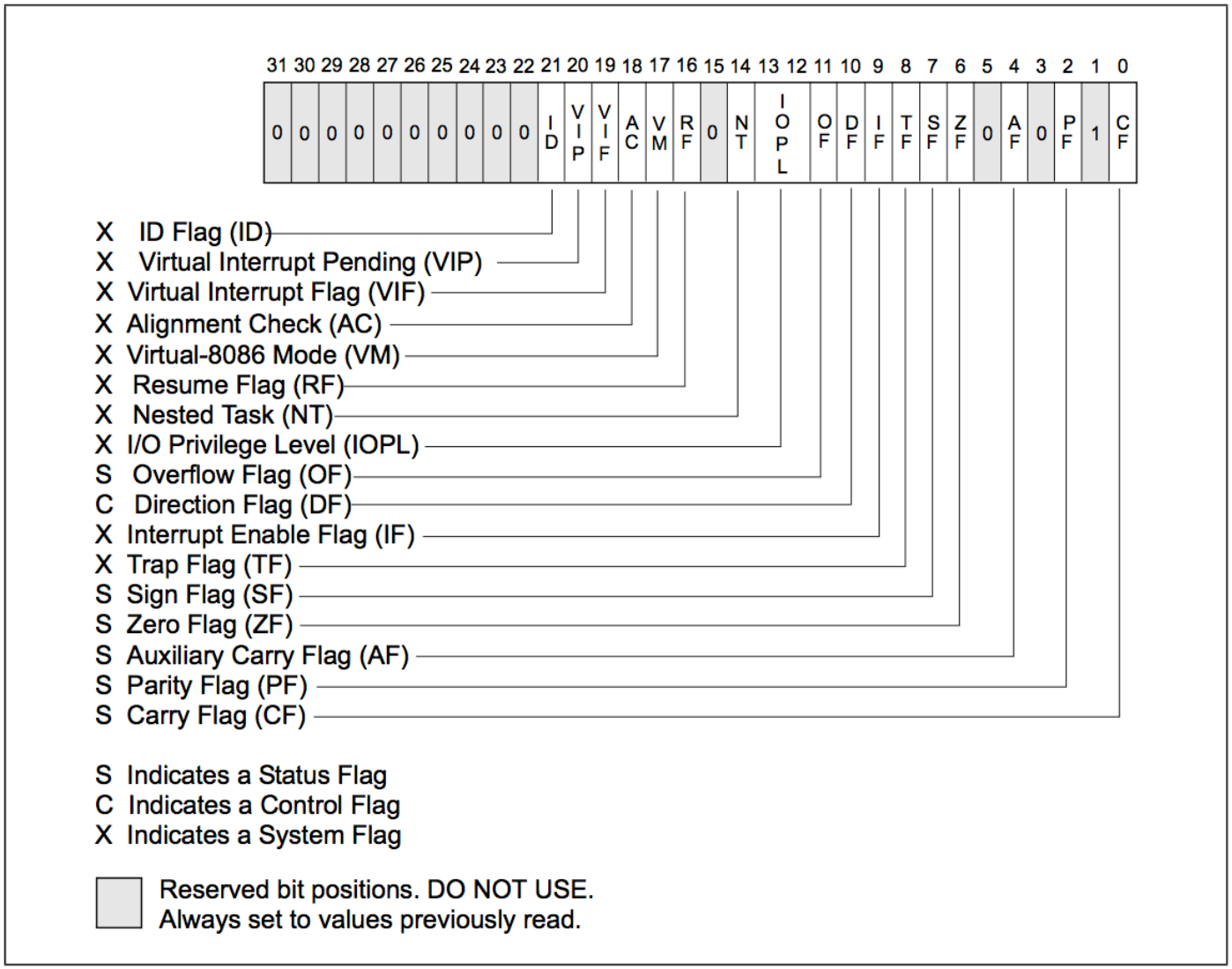
EAX	Accumulator Register – used for storing operands and result data
EBX	Base Register – Pointer to Data
ECX	Counter Register – Loop operations
EDX	Data Register – I/O Pointer
ESI	Data Pointer Registers for memory operations
EDI	
ESP	Stack Pointer Register
EBP	Stack Data Pointer Register

Segment Registers



Usage Depends on memory model – Flat or Segmented?

EFLAGS Register



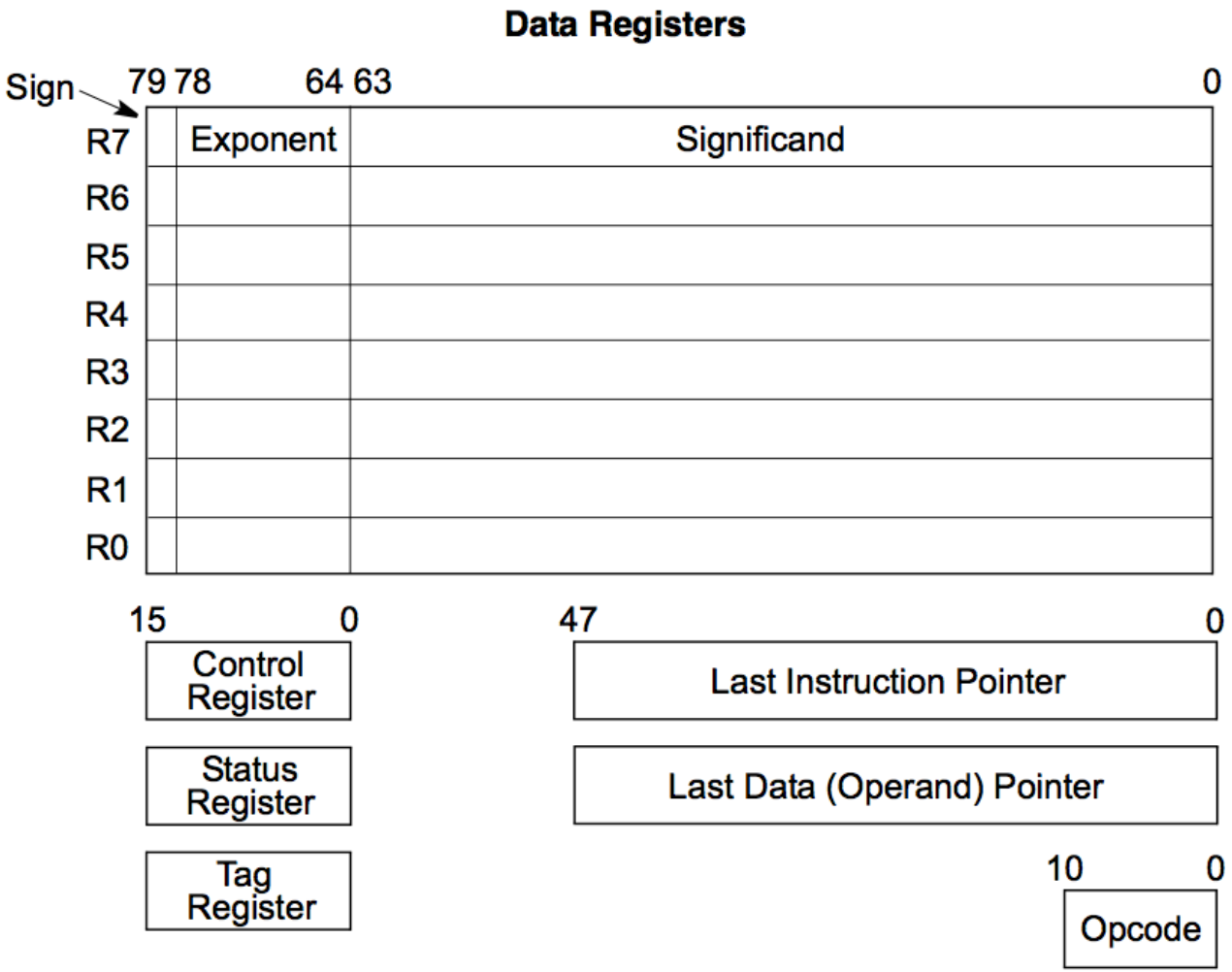
Ref: Intel manual for IA-32

EIP

- Instruction Pointer
- Holy grail for Shellcoding, Exploit Research etc.

Floating Point Unit (FPU) or x87

ST(0) to ST(7)

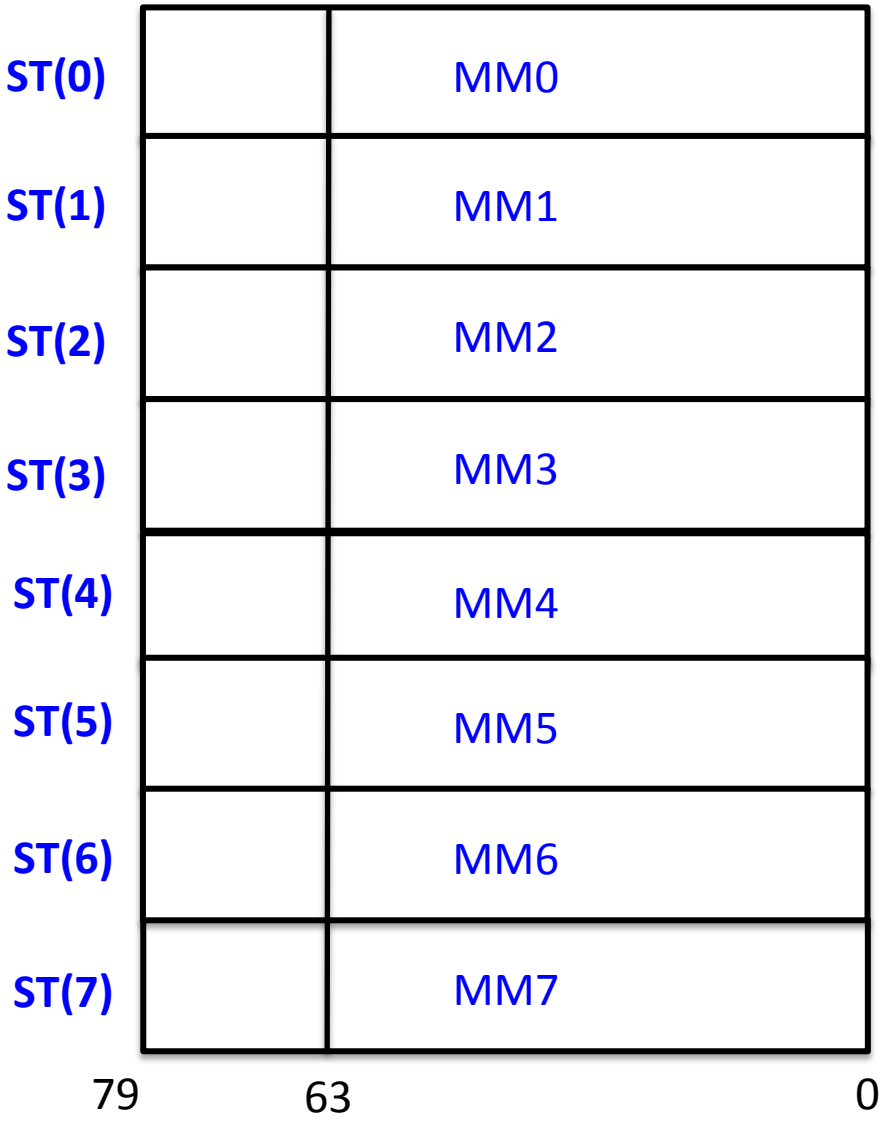


Reference: Intel Manual

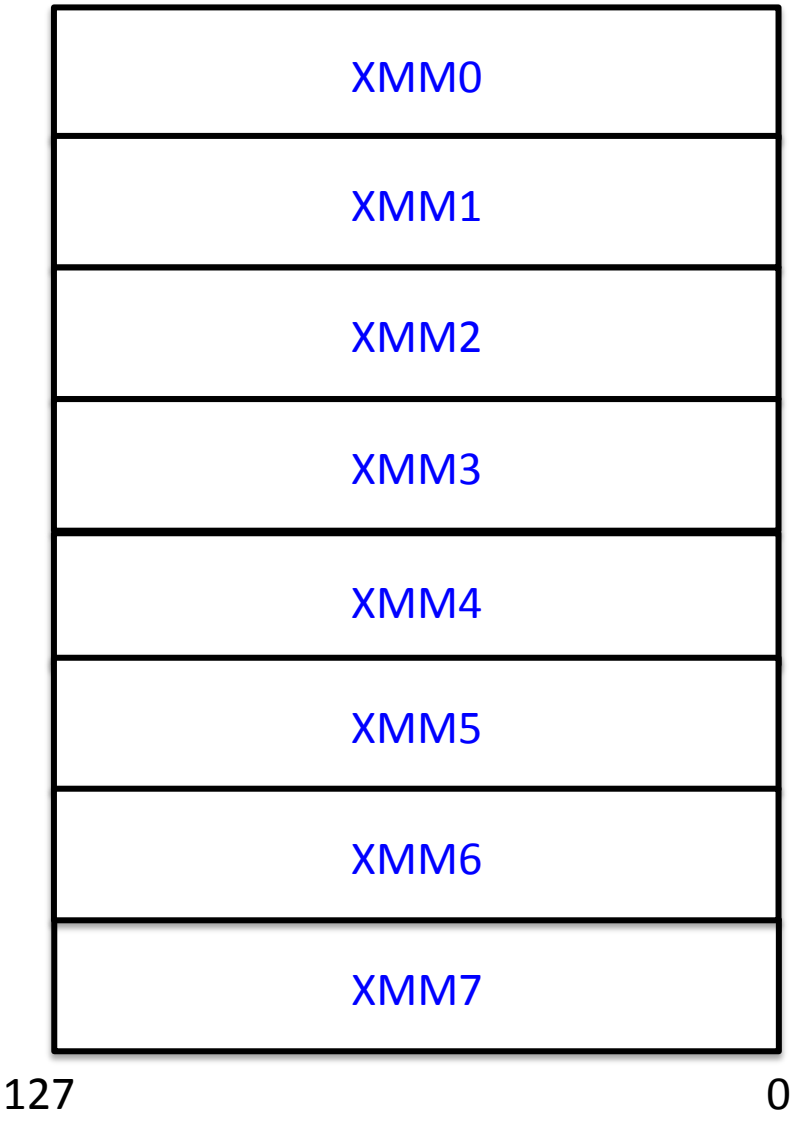
SIMD

- Single Instruction Multiple Data
- Extensions
  - MMX
  - SSE
  - SSE2
  - SSE3
- Uses MMX and XMM Registers

MMX



XMM



Exercise 1.2.1: Lab Setup

- Inspect the General Purpose, Segment, EFLAGS, FPU, MMX, XMM etc. registers on your Ubuntu system

Please Register for this course to receive the solution video for this exercise.
http://SecurityTube-Training.com

SecurityTube Linux Assembly Expert (SLAE^{32})

SecurityTube Linux Assembly Expert

Training: http://www.SecurityTube-Training.com

Community: http://www.SecurityTube.net

Vivek Ramachandran
SWSE, SMFE, SPSE, SGDE, SISE, SLAE^{32} Course Instructor

Exercise 1.2.1: Lab Setup

- Inspect the General Purpose, Segment, EFLAGS, FPU, MMX, XMM etc. registers on your Ubuntu system

SecurityTube Linux Assembly Expert (SLAE^{32})

SecurityTube Linux Assembly Expert

Training: http://www.SecurityTube-Training.com

Community: http://www.SecurityTube.net

Vivek Ramachandran
SWSE, SMFE, SPSE, SGDE, SISE, SLAE^{32} Course Instructor

Notice

This video is part of the **SecurityTube Linux Assembly Expert** course. A limited number of videos in this course will be released on SecurityTube.net

For full access to all theory and exercise videos, PDF slides, Code snippets etc. Please register as a student by visiting:

http://securitytube-training.com

Module 1: 32-Bit ASM on Linux

3. CPU Modes and Memory Management

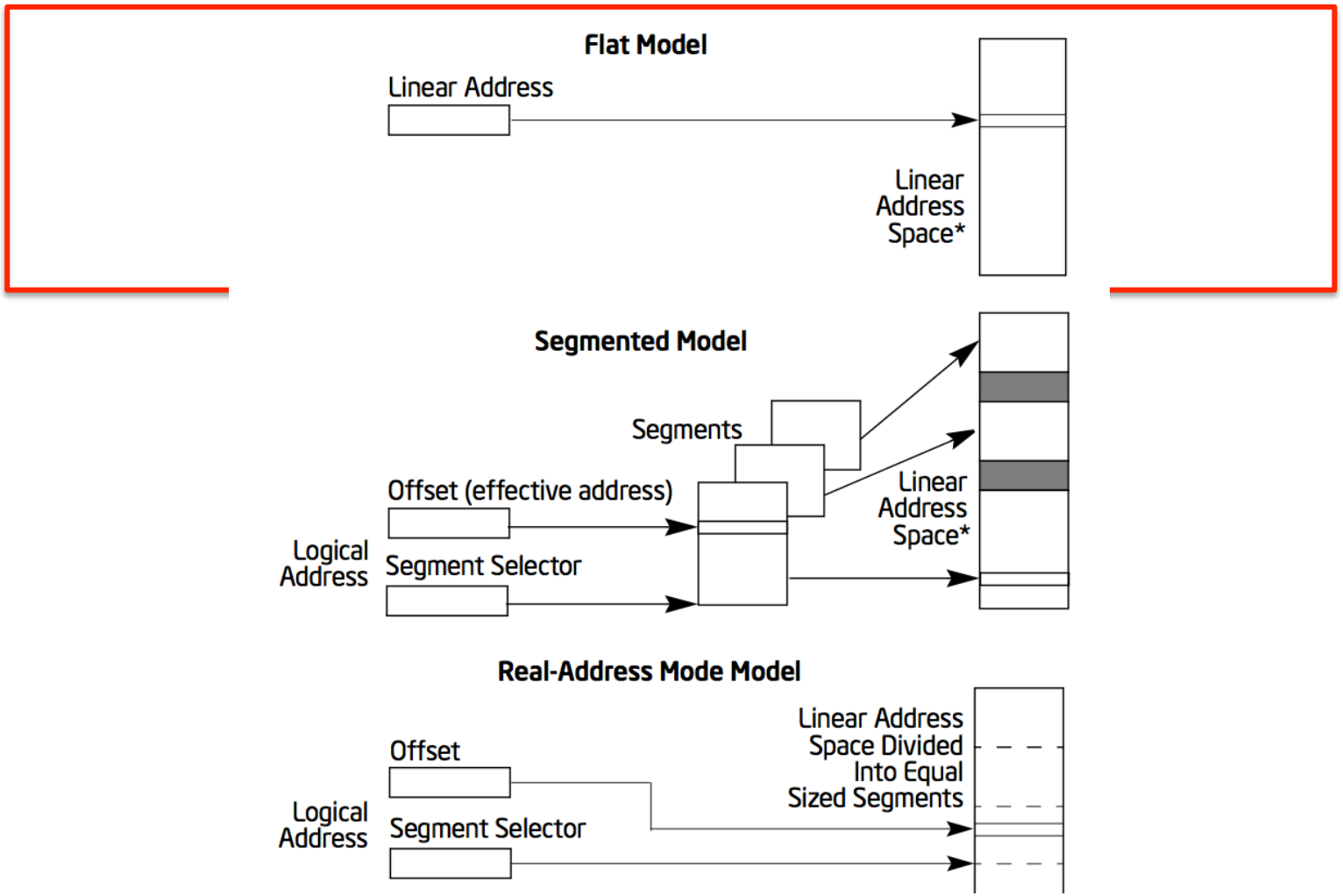
Vivek Ramachandran
SWSE, SMFE, SPSE, SGDE, SISE, SLAE^{32} Course Instructor

http://SecurityTube-Training.com

CPU Modes for IA-32

- Real Mode
  - At power up or reset
  - Can only access 1 MB memory
  - No memory protection
  - Privilege Levels (Kernel vs User space) not possible
- Protected Mode
  - Up to 4GB memory
  - memory protection / privilege level / multi-tasking
  - Supports Virtual-8086 mode
- System Management Mode
  - Used for power management tasks

Memory Models

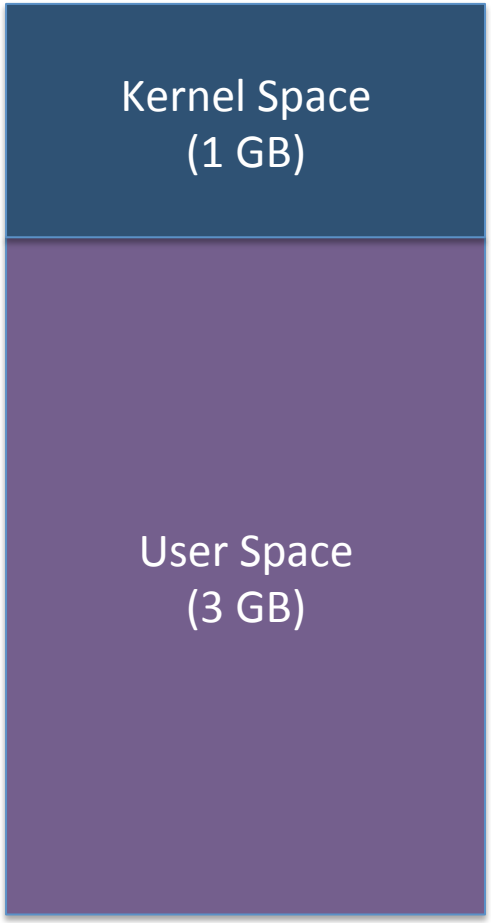


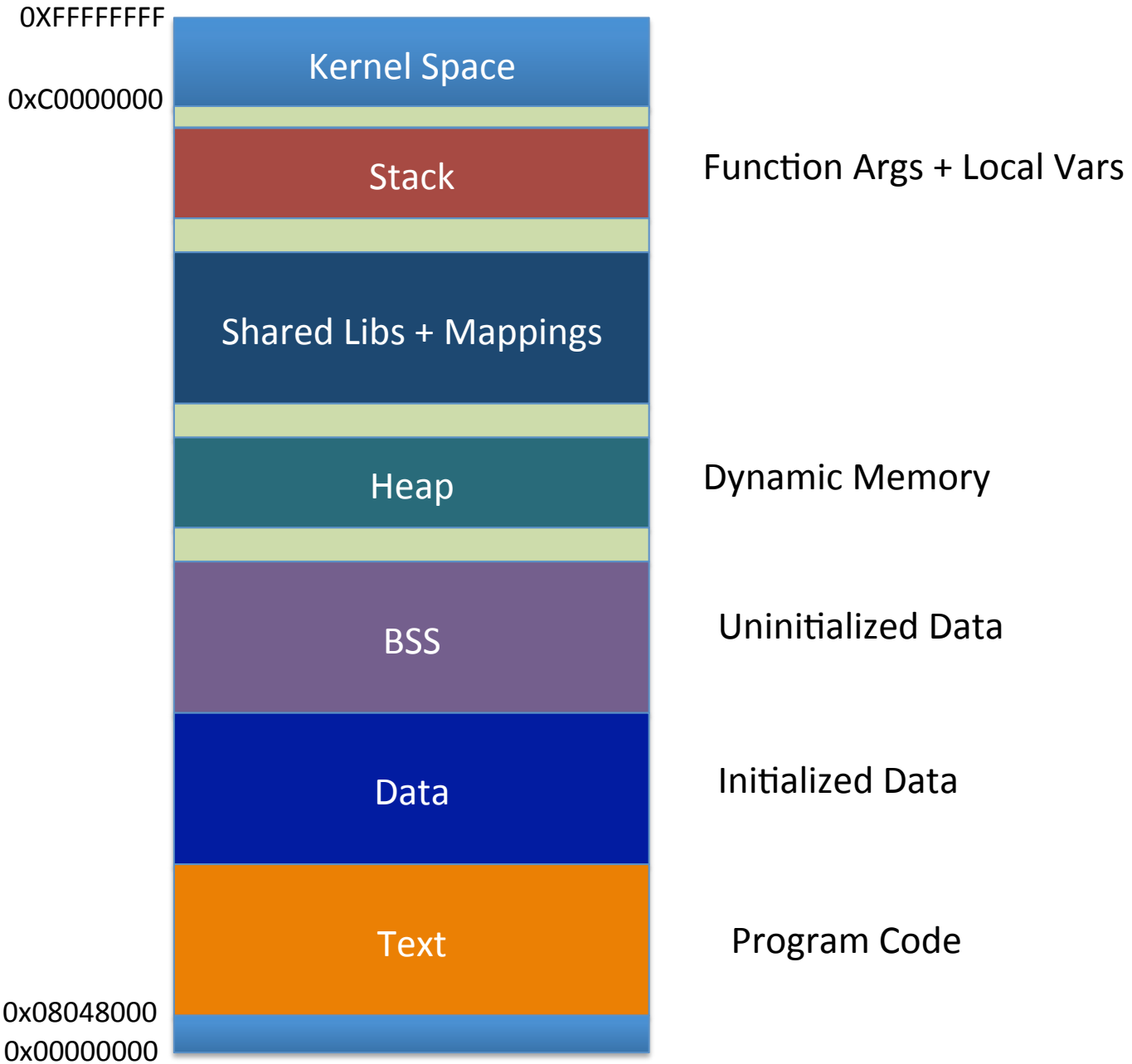
Source: Intel manuals

Linux: CPU Mode and Memory Model

- 32-Bit Linux uses:
  - Protected Mode
  - Flat Memory model
  - 4GB Addressable space => 2^{32}
  - Memory Protection
  - Privilege Levels of Code
  - Segment registers point to segment descriptors
    - GDT / LDT / IDT (Global / Local / Interrupt)

Virtual Memory Model





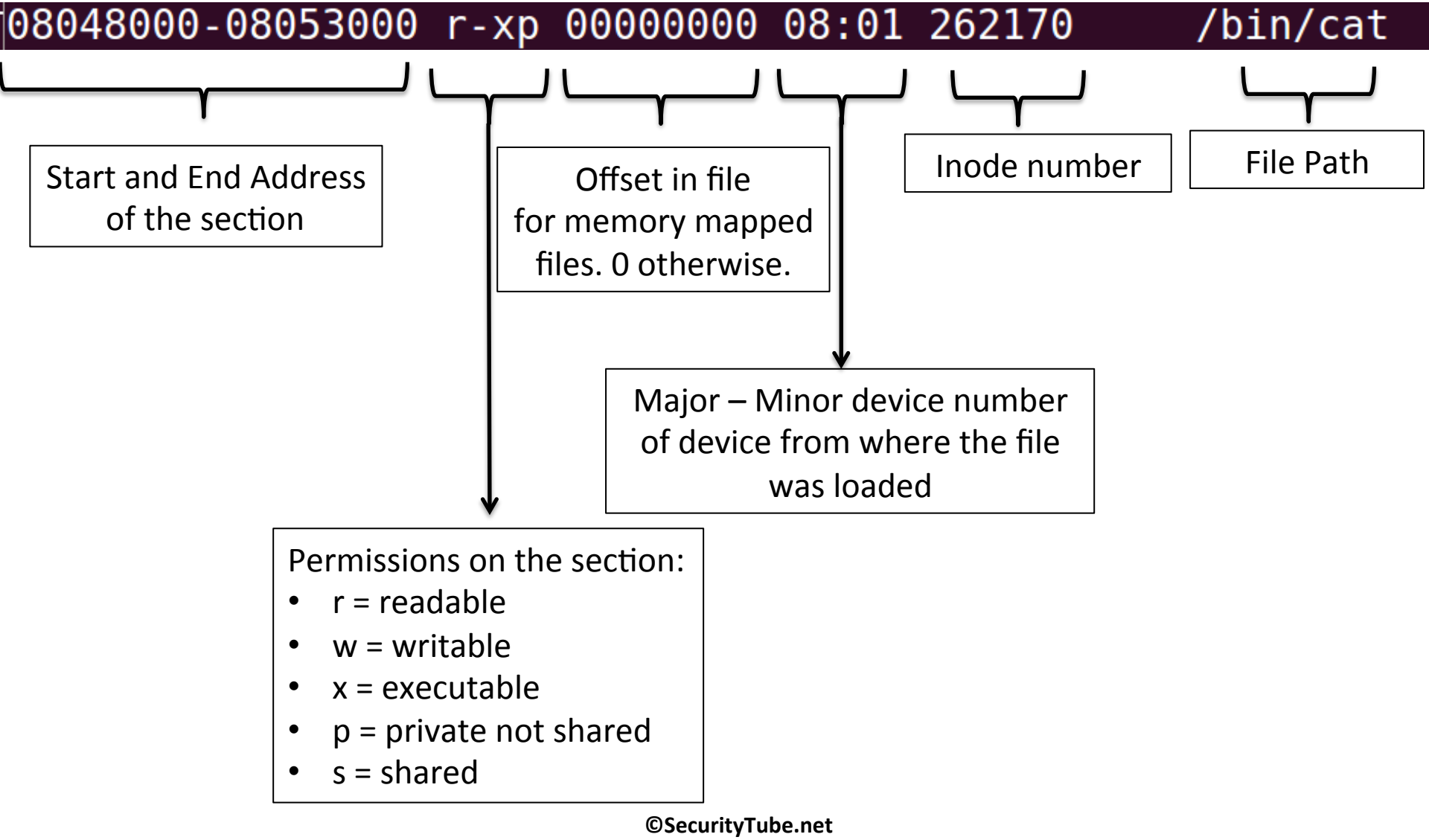
View Process Organization

- /Proc
  - /proc/pid/maps
- pmap
- Attach and view using GDB

cat /proc/pid/maps

```
securitytube@securitytube-VirtualBox:~$ cat /proc/self/maps
08048000-08053000 r-xp 00000000 08:01 262170     /bin/cat
08053000-08054000 r--p 0000a000 08:01 262170     /bin/cat
08054000-08055000 rw-p 0000b000 08:01 262170     /bin/cat
0917e000-0919f000 rw-p 00000000 00:00 0          [heap]
b73cd000-b75cd000 r--p 00000000 08:01 140017     /usr/lib/locale/locale-archive
b75cd000-b75ce000 rw-p 00000000 00:00 0
b75ce000-b776d000 r-xp 00000000 08:01 265843     /lib/i386-linux-gnu/libc-2.15.so
b776d000-b776f000 r--p 0019f000 08:01 265843     /lib/i386-linux-gnu/libc-2.15.so
b776f000-b7770000 rw-p 001a1000 08:01 265843     /lib/i386-linux-gnu/libc-2.15.so
b7770000-b7773000 rw-p 00000000 00:00 0
b7782000-b7783000 r--p 005db000 08:01 140017     /usr/lib/locale/locale-archive
b7783000-b7785000 rw-p 00000000 00:00 0
b7785000-b7786000 r-xp 00000000 00:00 0          [vdso]
b7786000-b77a6000 r-xp 00000000 08:01 265823     /lib/i386-linux-gnu/ld-2.15.so
b77a6000-b77a7000 r--p 0001f000 08:01 265823     /lib/i386-linux-gnu/ld-2.15.so
b77a7000-b77a8000 rw-p 00020000 08:01 265823     /lib/i386-linux-gnu/ld-2.15.so
bfdfc000-bfe1d000 rw-p 00000000 00:00 0          [stack]
securitytube@securitytube-VirtualBox:~$
```

What does all this mean?



Process Map within GDB

```
(gdb) info proc mappings
process 2837
Mapped address spaces:

        Start Addr   End Addr       Size     Offset objfile
         0x8048000  0x8124000    0xdc000        0x0 /bin/bash
         0x8124000  0x8125000     0x1000    0xdb000 /bin/bash
         0x8125000  0x812a000     0x5000    0xdc000 /bin/bash
         0x812a000  0x812f000     0x5000        0x0 [heap]
        0xb7e00000 0xb7e01000     0x1000        0x0
        0xb7e01000 0xb7fa0000   0x19f000        0x0 /lib/i386-linux-gnu/libc-2.15.so
        0xb7fa0000 0xb7fa2000     0x2000   0x19f000 /lib/i386-linux-gnu/libc-2.15.so
        0xb7fa2000 0xb7fa3000     0x1000   0x1a1000 /lib/i386-linux-gnu/libc-2.15.so
        0xb7fa3000 0xb7fa7000     0x4000        0x0
        0xb7fa7000 0xb7faa000     0x3000        0x0 /lib/i386-linux-gnu/libdl-2.15.so
        0xb7faa000 0xb7fab000     0x1000     0x2000 /lib/i386-linux-gnu/libdl-2.15.so
        0xb7fab000 0xb7fac000     0x1000     0x3000 /lib/i386-linux-gnu/libdl-2.15.so
        0xb7fac000 0xb7fc8000    0x1c000        0x0 /lib/i386-linux-gnu/libtinfo.so.5.9
        0xb7fc8000 0xb7fca000     0x2000    0x1b000 /lib/i386-linux-gnu/libtinfo.so.5.9
        0xb7fca000 0xb7fcb000     0x1000    0x1d000 /lib/i386-linux-gnu/libtinfo.so.5.9
        0xb7fdb000 0xb7fdd000     0x2000        0x0
        0xb7fdd000 0xb7fde000     0x1000        0x0 [vdso]
        0xb7fde000 0xb7ffe000    0x20000        0x0 /lib/i386-linux-gnu/ld-2.15.so
        0xb7ffe000 0xb7fff000     0x1000    0x1f000 /lib/i386-linux-gnu/ld-2.15.so
        0xb7fff000 0xb8000000     0x1000    0x20000 /lib/i386-linux-gnu/ld-2.15.so
        0xbffdf000 0xc0000000    0x21000        0x0 [stack]
(gdb)
```

SecurityTube Linux Assembly Expert (SLAE^{32})

SecurityTube Linux Assembly Expert

Training: http://www.SecurityTube-Training.com

Community: http://www.SecurityTube.net

Vivek Ramachandran
SWSE, SMFE, SPSE, SGDE, SISE, SLAE^{32} Course Instructor

Notice

This video is part of the **SecurityTube Linux Assembly Expert** course. A limited number of videos in this course will be released on SecurityTube.net

For full access to all theory and exercise videos, PDF slides, Code snippets etc. Please register as a student by visiting:

http://securitytube-training.com

Module 1: 32-Bit ASM on Linux

4. Hello World

Vivek Ramachandran
SWSE, SMFE, SPSE, SGDE, SISE, SLAE^{32} Course Instructor

http://SecurityTube-Training.com

IA-32 Instruction Set

- General Purpose Instructions
- x87 FPU Instructions
- MMX / SSE / SSE2/ SSE3 / SSE4 Instructions
- Other Instruction Set extensions

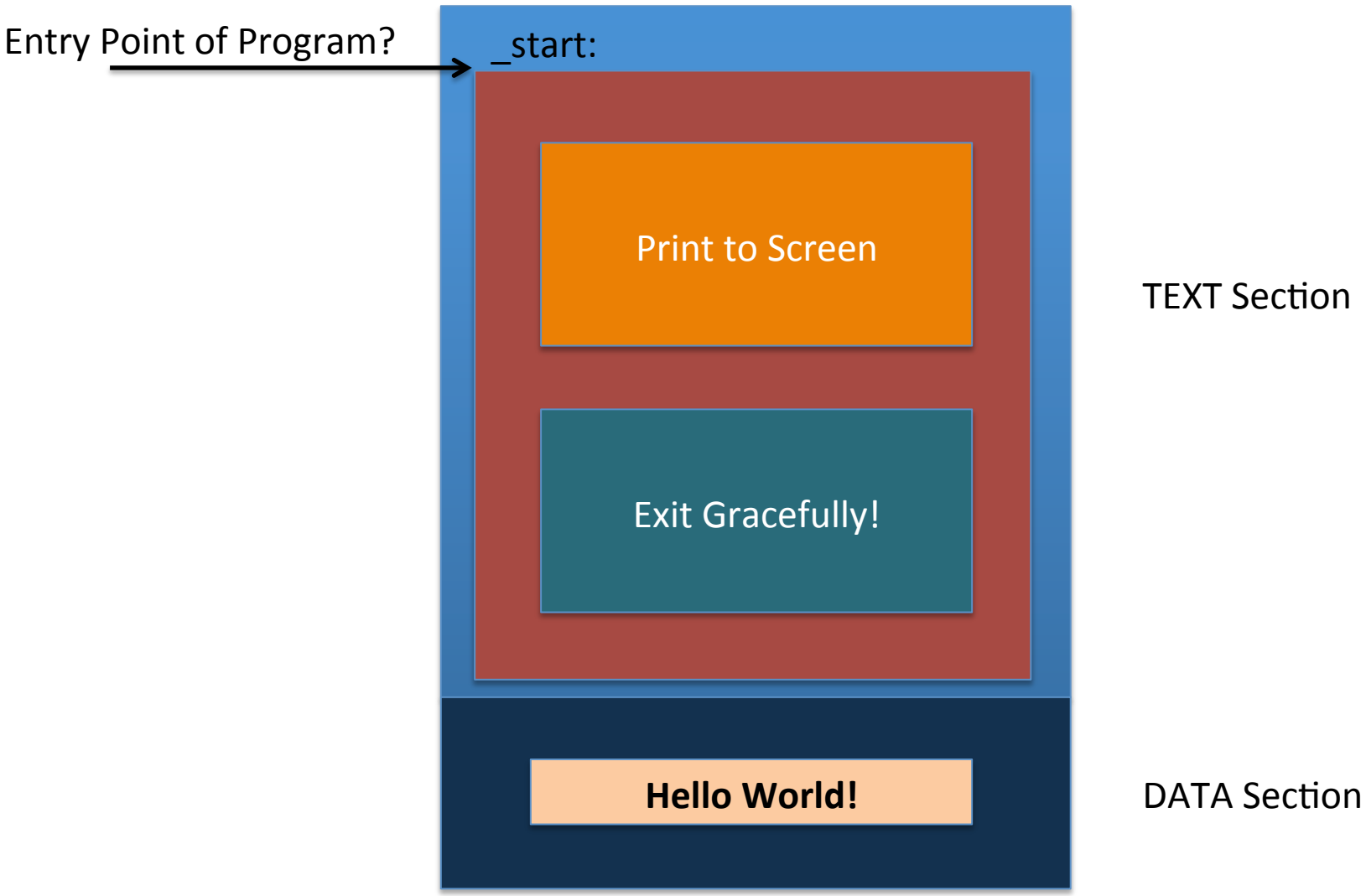
Programming in Assembly

- NASM + LD for assembling and linking
- Executable in ELF format

NASM Documentation:

http://nasm.us/

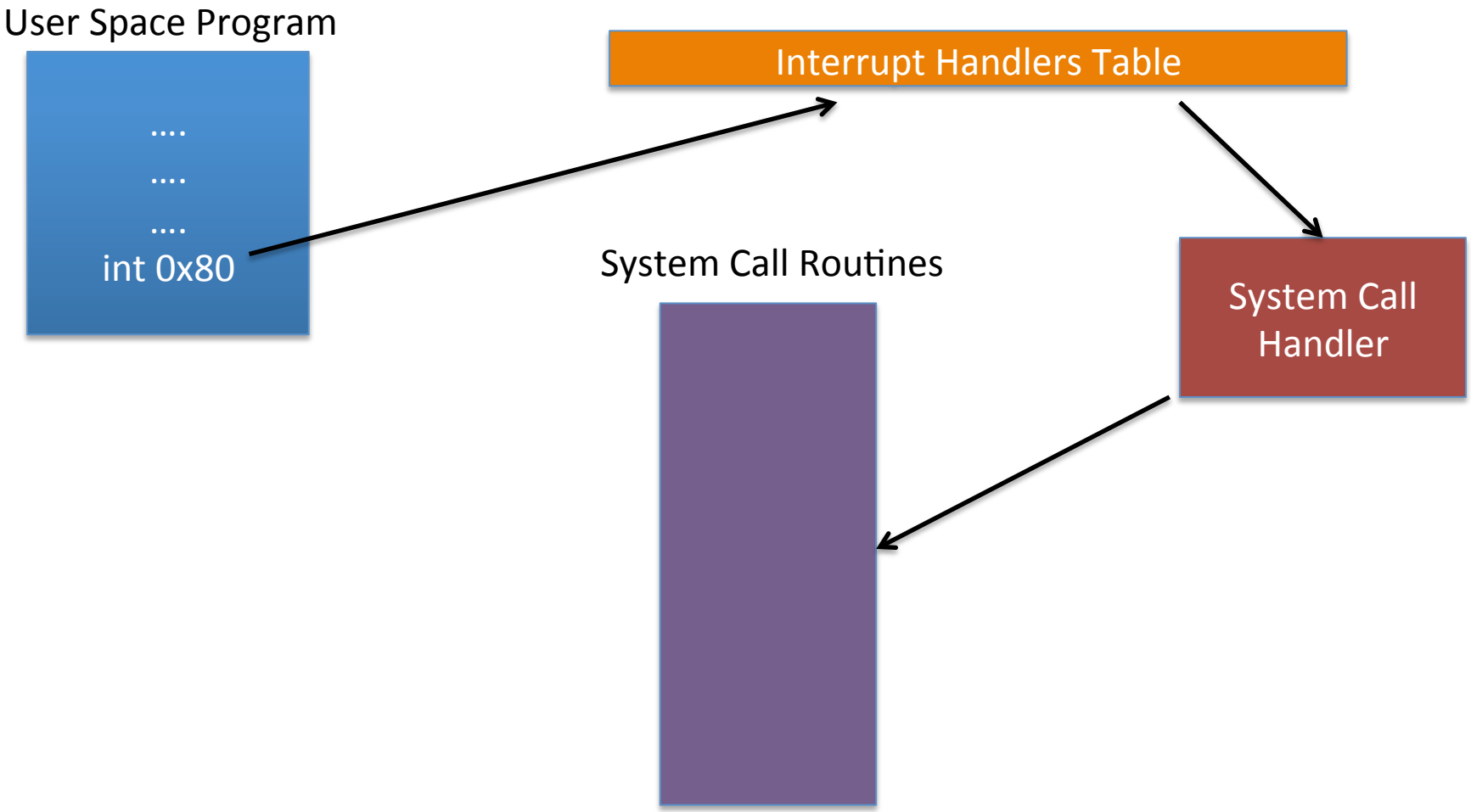
Hello World!



Why System Calls?

- Leverage OS for tasks
- Imagine if you had to write code from scratch to:
  - write to disk
  - print on screen
  - …
- System Calls provide a simple interface for user space programs to the Kernel

How do System calls work?



IA-32 Mechanism to invoke System Call

- int 0x80

- SYSENTER

Modern implementations using VDSO
[Virtual Dynamic Shared Object]

http://articles.manugarg.com/systemcallinlinux2_6.html

Where are these system calls defined?

```
#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
#define __NR_mount               21
#define __NR_umount              22
#define __NR_setuid              23
#define __NR_getuid              24
#define __NR_stime               25
"/usr/include/i386-linux-gnu/asm/unistd_32.h"
```

write()

```
WRITE(2)                                                Linux Programmer's Manual

NAME
       write - write to a file descriptor

SYNOPSIS
       #include <unistd.h>

       ssize_t write(int fd, const void *buf, size_t count);

DESCRIPTION
       write() writes up to count bytes from the buffer pointed buf to th
```

exit

```
_EXIT(2)                                                    Linux Pro

NAME
       _exit, _Exit - terminate the calling process

SYNOPSIS
       #include <unistd.h>

       void _exit(int status);
```

Invoking System Call with 0x80

EAX	**System Call Number**
EBX	1st Argument
ECX	2nd Argument
EDX	3rd Argument
ESI	4th Argument
EDI	5th Argument

Return Value in EAX

Calling Write

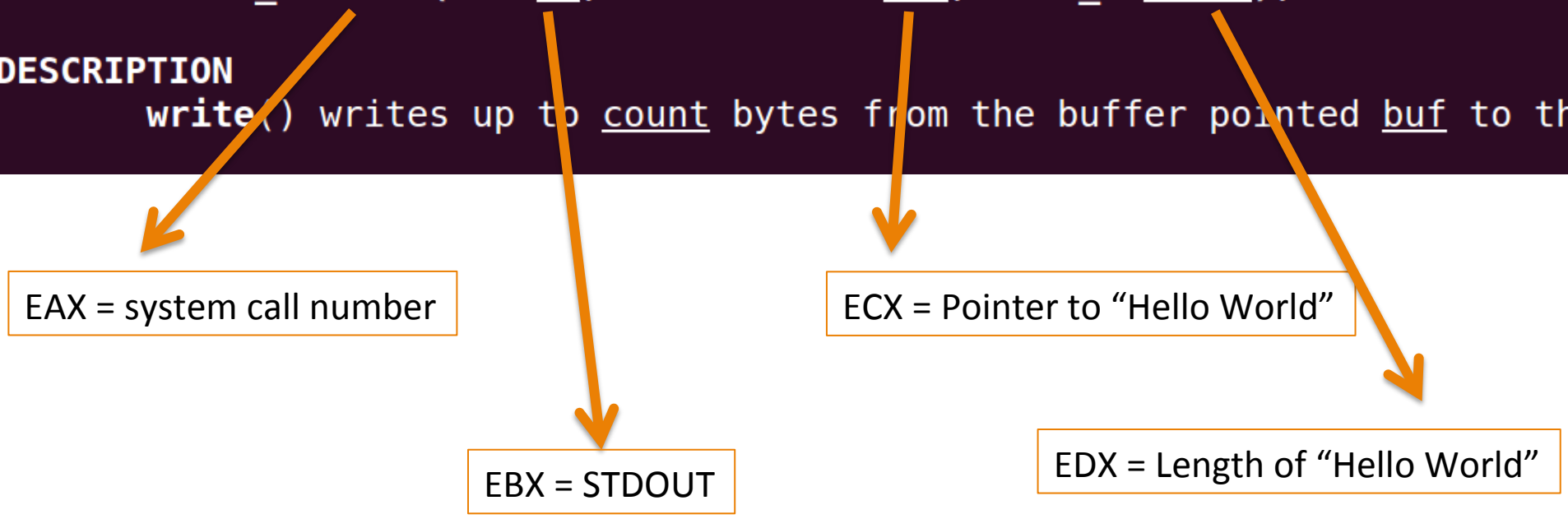
```
WRITE(2)                                                Linux Programmer's Manual

NAME
       write - write to a file descriptor

SYNOPSIS
       #include <unistd.h>

       ssize_t write(int fd, const void *buf, size_t count);

DESCRIPTION
       write() writes up to count bytes from the buffer pointed buf to th
```



Calling Exit

```
_EXIT(2)                                                    Linux Pro

NAME
       _exit, _Exit - terminate the calling process

SYNOPSIS
       #include <unistd.h>

       void _exit(int status);
```

EAX = system call number

EBX = Status Code

Exercise 1.4.1: GDB

- Use GDB to step through the Hello World program and observe:
  - CPU Registers
  - Memory Location
  - …

Please Register for this course to receive the solution video for this exercise.
http://SecurityTube-Training.com

SecurityTube Linux Assembly Expert (SLAE^{32})

SecurityTube Linux Assembly Expert

Training: http://www.SecurityTube-Training.com

Community: http://www.SecurityTube.net

Vivek Ramachandran
SWSE, SMFE, SPSE, SGDE, SISE, SLAE^{32} Course Instructor

Exercise 1.4.1: GDB

- Use GDB to step through the Hello World program and observe:
  - CPU Registers
  - Memory Location
  - …

Please Register for this course to receive the solution video for this exercise.
http://SecurityTube-Training.com

SecurityTube Linux Assembly Expert (SLAE^{32})

SecurityTube Linux Assembly Expert

Training: http://www.SecurityTube-Training.com

Community: http://www.SecurityTube.net

Vivek Ramachandran
SWSE, SMFE, SPSE, SGDE, SISE, SLAE^{32} Course Instructor

Notice

This video is part of the **SecurityTube Linux Assembly Expert** course. A limited number of videos in this course will be released on SecurityTube.net

For full access to all theory and exercise videos, PDF slides, Code snippets etc. Please register as a student by visiting:

http://securitytube-training.com

Module 1: 32-Bit ASM on Linux

5. Data Types

Vivek Ramachandran
SWSE, SMFE, SPSE, SGDE, SISE, SLAE^{32} Course Instructor

http://SecurityTube-Training.com

Fundamental Data Types

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

Source: IA-32 Manual

Signed and Unsigned

Unsigned Double Word

31 0

Signed Double Word

31 30 0

Source: IA-32 Manual

NASM …

- Case Sensitive syntax

- Accessing memory reference with []
  - message db 0xAA, 0xBB, 0xCC, 0xDD
  - mov eax, message ← moves address into eax
  - move eax, [message] ← moves value into eax

Defining Initialized Data in NASM

```
db    0x55                ; just the byte 0x55
db    0x55,0x56,0x57      ; three bytes in succession
db    'a',0x55            ; character constants are OK
db    'hello',13,10,'$'   ; so are string constants
dw    0x1234              ; 0x34 0x12
dw    'a'                 ; 0x61 0x00 (it's just a number)
dw    'ab'                ; 0x61 0x62 (character constant)
dw    'abc'               ; 0x61 0x62 0x63 0x00 (string)
dd    0x12345678          ; 0x78 0x56 0x34 0x12
dd    1.234567e20         ; floating-point constant
dq    0x123456789abcdef0  ; eight byte constant
dq    1.234567e20         ; double-precision float
dt    1.234567e20         ; extended-precision float
```

Source: NASM Manual Pg. 30

Declare Uninitialized Data

```
buffer:         resb    64              ; reserve 64 bytes
wordvar:        resw    1               ; reserve a word
```

Source: NASM Manual Pg. 30

Special Tokens

- $ - evaluates to the current line

- $$ - evaluates to the beginning of current section

Source: NASM Manual Pg. 37

EQU and TIMES

```
message         db      'hello, world'
msglen          equ     $-message
```

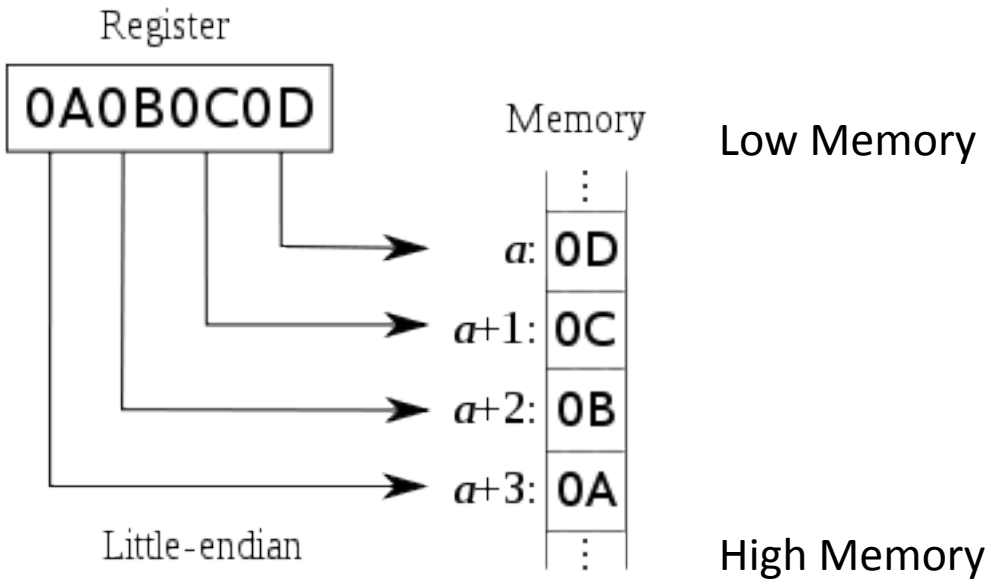
Data:

```
zerobuf:        times 64 db 0
```

Instruction:

```
times 100 movsb
```

IA-32 uses Little Endian format



Source: Wikipedia http://en.wikipedia.org/wiki/Endianness

GDB

SecurityTube GNU Debugger Expert

http://www.securitytube.net/tags/sgde

SecurityTube Linux Assembly Expert (SLAE^{32})

SecurityTube Linux Assembly Expert

Training: http://www.SecurityTube-Training.com

Community: http://www.SecurityTube.net

Vivek Ramachandran
SWSE, SMFE, SPSE, SGDE, SISE, SLAE^{32} Course Instructor

Notice

This video is part of the **SecurityTube Linux Assembly Expert** course. A limited number of videos in this course will be released on SecurityTube.net

For full access to all theory and exercise videos, PDF slides, Code snippets etc. Please register as a student by visiting:

http://securitytube-training.com

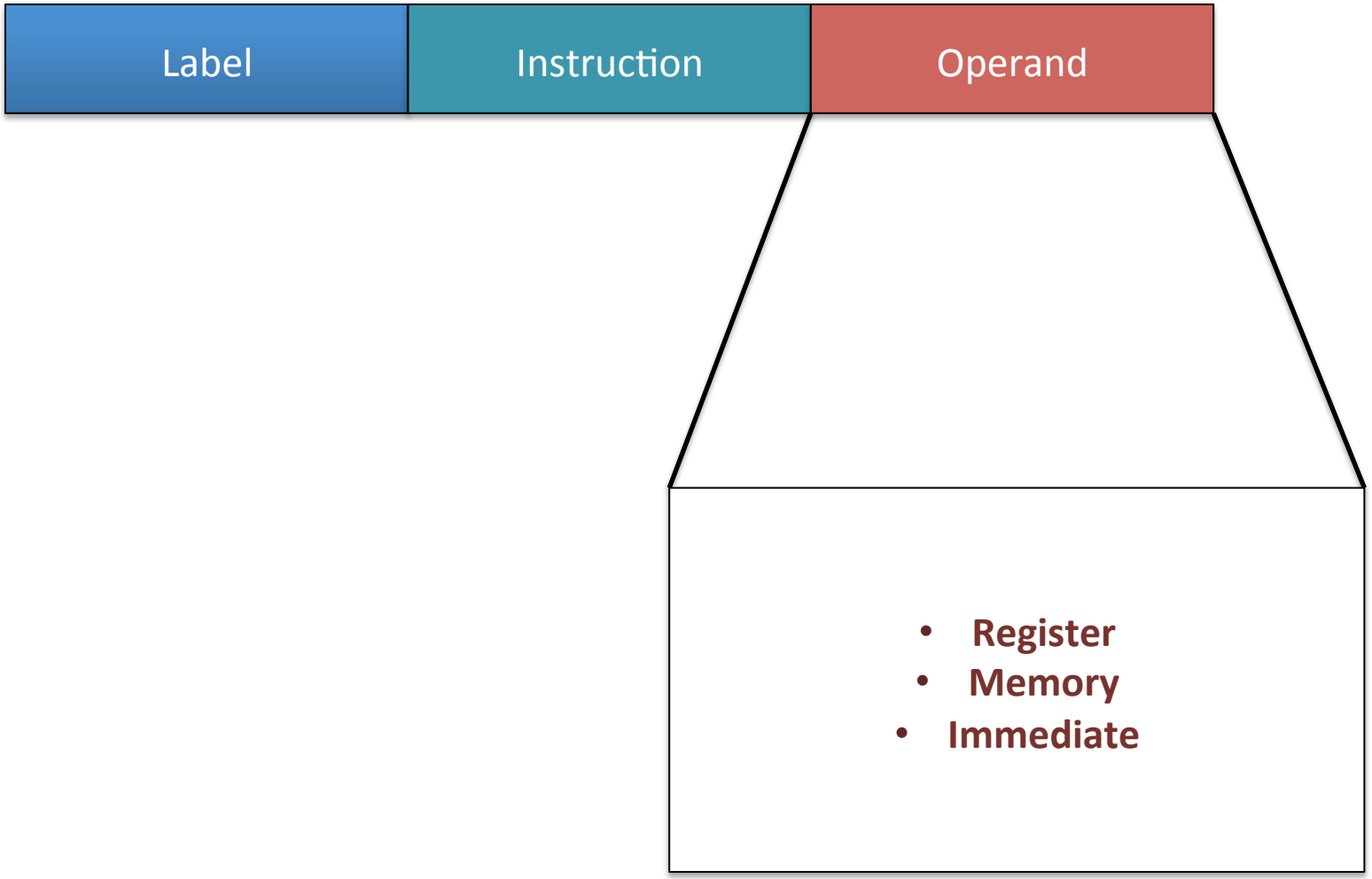
Module 1: 32-Bit ASM on Linux

6. Moving Data

Vivek Ramachandran
SWSE, SMFE, SPSE, SGDE, SISE, SLAE^{32} Course Instructor

http://SecurityTube-Training.com

Instruction



MOV

- Most common instruction in ASM

- Allowed Directions
  - Between Registers
  - Memory to Register and Register to Memory
  - Immediate Data to Register
  - Immediate Data to Memory

LEA

- Load Effective Address – load pointer values
- LEA EAX, [label]

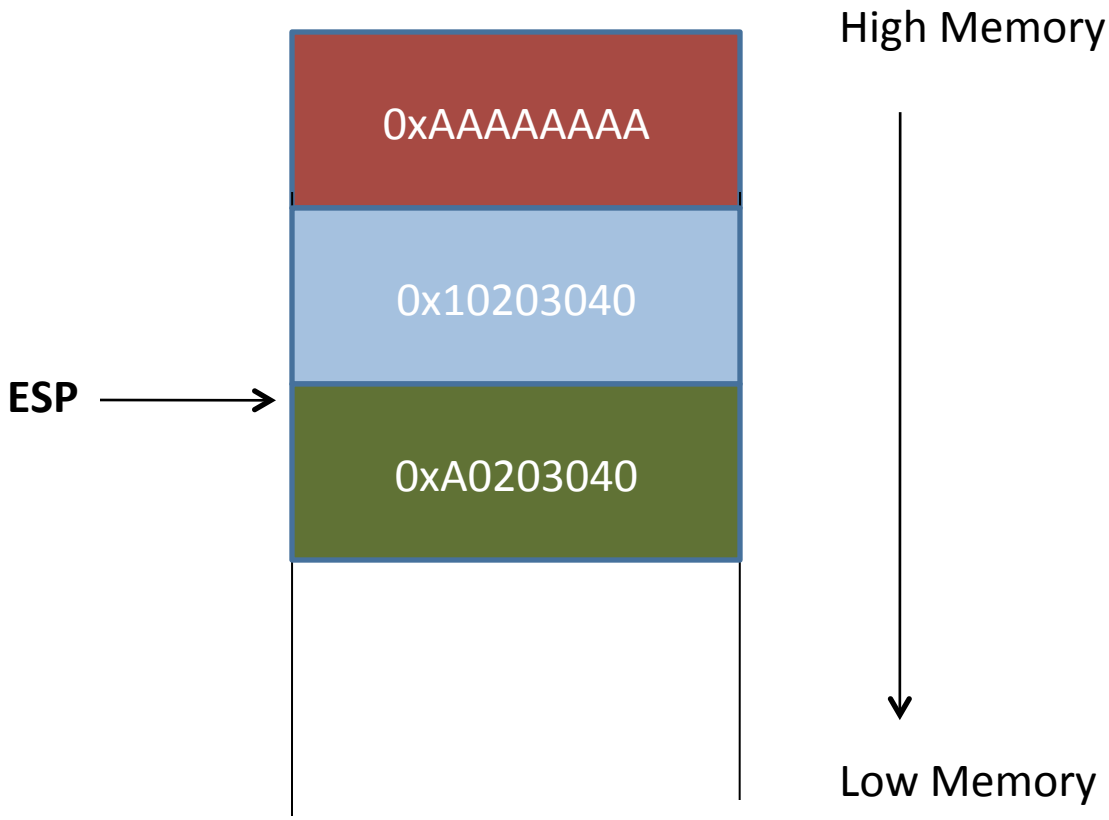
XCHG

- Exchanges (swaps) Values
- XCHG Register, Register
- XCHG Register, Memory

Stack

- Used by processes and threads to store temporary data
  - local variables
  - return addresses
- Stack is a Last-in-First-out (LIFO) data structure

Stack is a LIFO



PUSH – Pushes a value onto the Stack

POP – Removes the topmost value from the Stack

ESP – Should point to top of Stack

Exercise 1.6.1

- Use the PUSH / POP / … stack instructions in your program

- Use GDB to examine the stack using ESP and track the changes as the instructions run

Please Register for this course to receive the solution video for this exercise.
http://SecurityTube-Training.com

SecurityTube Linux Assembly Expert (SLAE^{32})

SecurityTube Linux Assembly Expert

Training: http://www.SecurityTube-Training.com

Community: http://www.SecurityTube.net

Vivek Ramachandran
SWSE, SMFE, SPSE, SGDE, SISE, SLAE^{32} Course Instructor

Notice

This video is part of the **SecurityTube Linux Assembly Expert** course. A limited number of videos in this course will be released on SecurityTube.net

For full access to all theory and exercise videos, PDF slides, Code snippets etc. Please register as a student by visiting:

http://securitytube-training.com

Exercise 1.6.1

- Use the PUSH / POP / … stack instructions in your program

- Use GDB to examine the stack using ESP and track the changes as the instructions run

Please Register for this course to receive the solution video for this exercise.
http://SecurityTube-Training.com

SecurityTube Linux Assembly Expert (SLAE^{32})

SecurityTube Linux Assembly Expert

Training: http://www.SecurityTube-Training.com

Community: http://www.SecurityTube.net

Vivek Ramachandran
SWSE, SMFE, SPSE, SGDE, SISE, SLAE^{32} Course Instructor

Notice

This video is part of the **SecurityTube Linux Assembly Expert** course. A limited number of videos in this course will be released on SecurityTube.net

For full access to all theory and exercise videos, PDF slides, Code snippets etc. Please register as a student by visiting:

http://securitytube-training.com

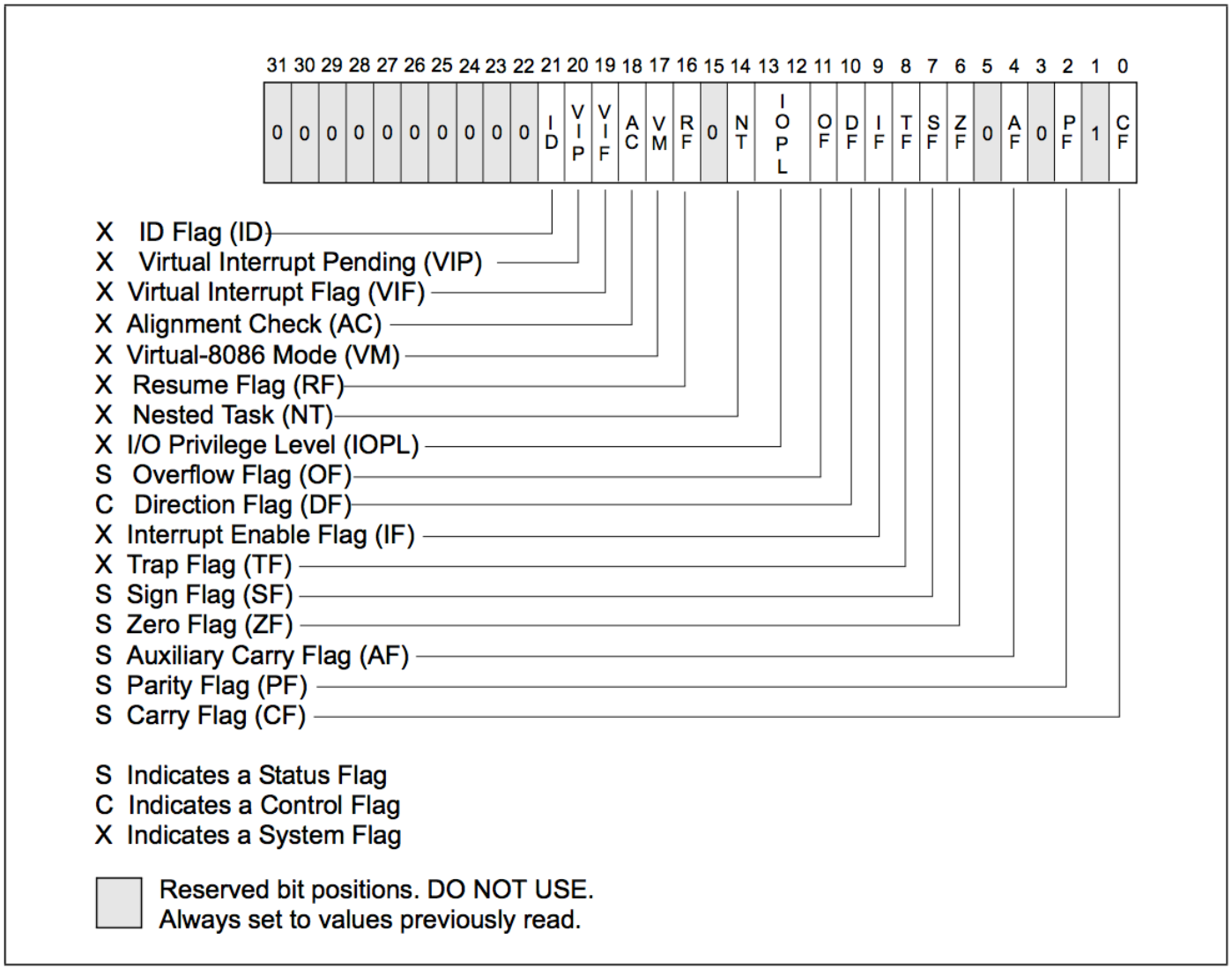
Module 1: 32-Bit ASM on Linux

7. Arithmetic Instructions

Vivek Ramachandran
SWSE, SMFE, SPSE, SGDE, SISE, SLAE^{32} Course Instructor

http://SecurityTube-Training.com

EFLAGS Register



Ref: Intel manual for IA-32

Arithmetic Instructions

- ADD destination, source
- ADC destination, source (plus carry flag)
- SUB and SBB
- INC and DEC

Exercise 1.7.1

- Multiply and Divide Instructions
- GDB to trace execution and register values

Please Register for this course to receive the solution video for this exercise.
http://SecurityTube-Training.com

SecurityTube Linux Assembly Expert (SLAE^{32})

SecurityTube Linux Assembly Expert

Training: http://www.SecurityTube-Training.com

Community: http://www.SecurityTube.net

Vivek Ramachandran
SWSE, SMFE, SPSE, SGDE, SISE, SLAE^{32} Course Instructor

Notice

This video is part of the **SecurityTube Linux Assembly Expert** course. A limited number of videos in this course will be released on SecurityTube.net

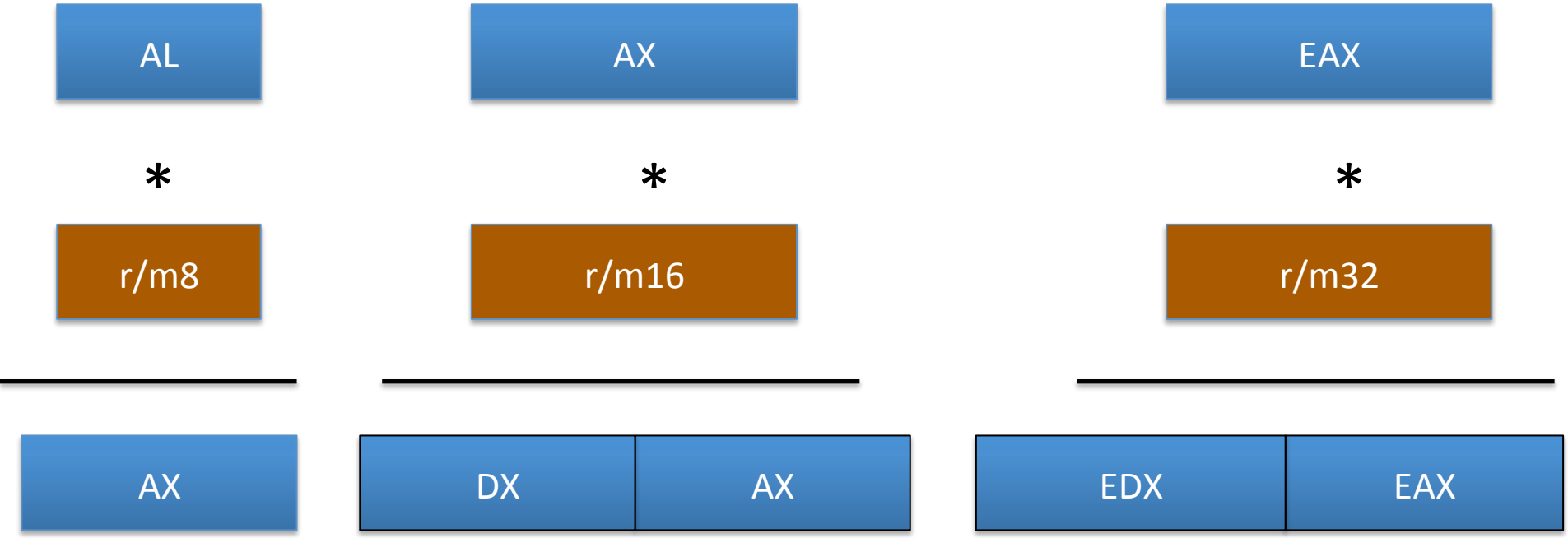
For full access to all theory and exercise videos, PDF slides, Code snippets etc. Please register as a student by visiting:

http://securitytube-training.com

Exercise 1.7.1

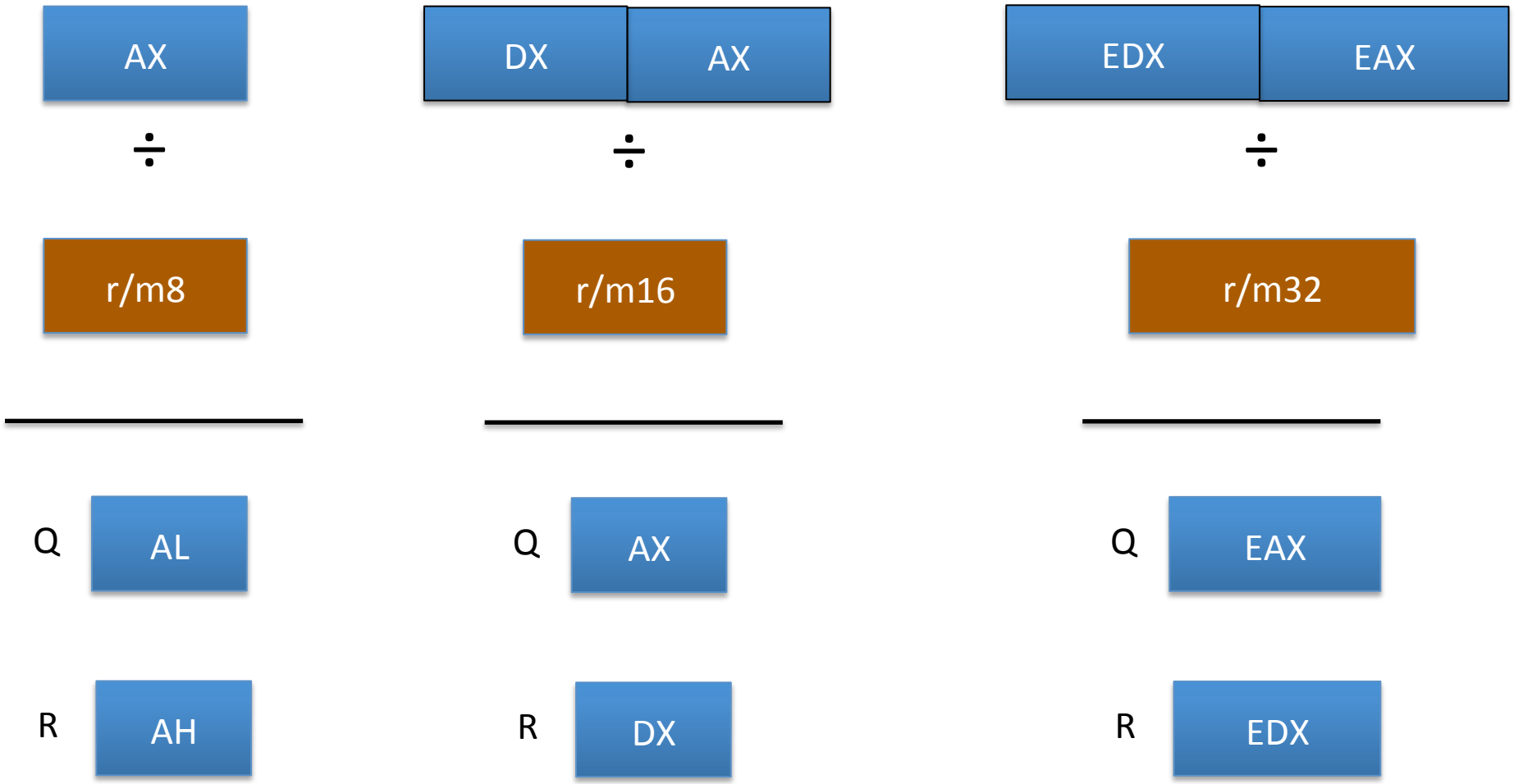
- Multiply and Divide Instructions
- GDB to trace execution and register values

Unsigned Multiply (MUL)



OF = 1 and CF = 1 if upper half of result is non-zero

Unsigned Divide (DIV)



Signed Arithmetic

- IMUL
- IDIV

SecurityTube Linux Assembly Expert (SLAE^{32})

SecurityTube Linux Assembly Expert

Training: http://www.SecurityTube-Training.com

Community: http://www.SecurityTube.net

Vivek Ramachandran
SWSE, SMFE, SPSE, SGDE, SISE, SLAE^{32} Course Instructor

Notice

This video is part of the **SecurityTube Linux Assembly Expert** course. A limited number of videos in this course will be released on SecurityTube.net

For full access to all theory and exercise videos, PDF slides, Code snippets etc. Please register as a student by visiting:

http://securitytube-training.com

Module 1: 32-Bit ASM on Linux

8. Logical Instructions

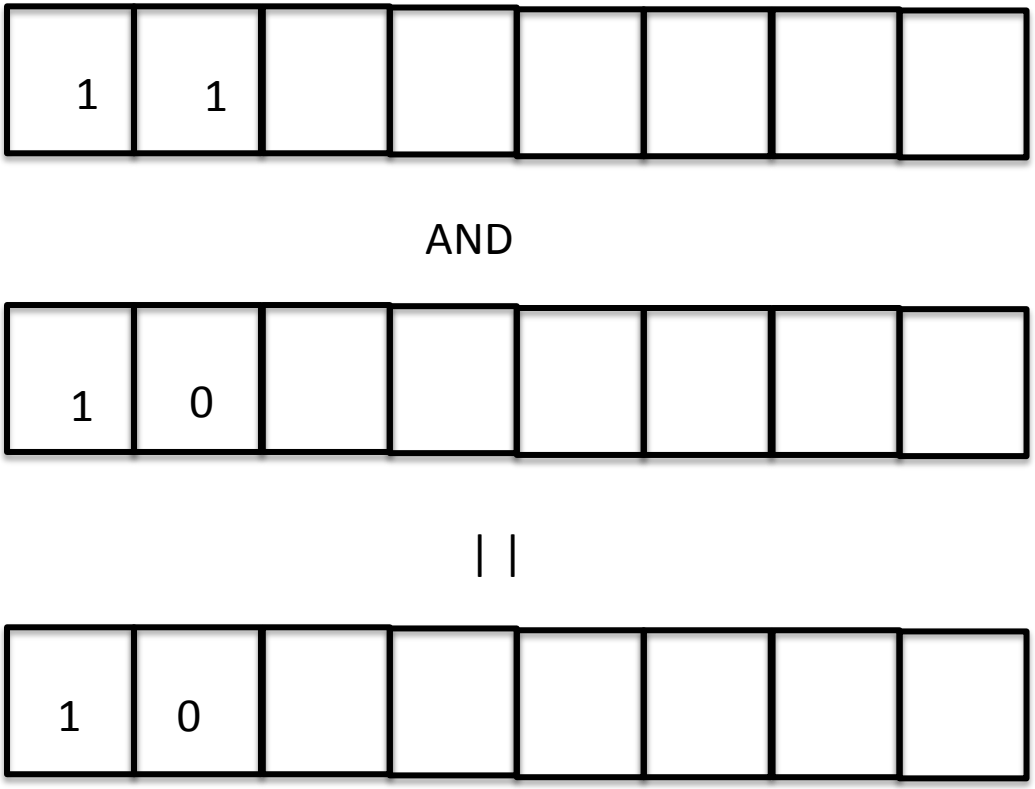
Vivek Ramachandran
SWSE, SMFE, SPSE, SGDE, SISE, SLAE^{32} Course Instructor

http://SecurityTube-Training.com

Logical Operations

- AND r/m , r/m/imm (8, 16, 32 bits)
- OR
- XOR
- NOT

Bitwise Operation



Other Instructions

- SAR – Shift Arithmetic Right
- SHR - Shift Logical Right
- ROR
- ROL
- …

SecurityTube Linux Assembly Expert (SLAE^{32})

SecurityTube Linux Assembly Expert

Training: http://www.SecurityTube-Training.com

Community: http://www.SecurityTube.net

Vivek Ramachandran
SWSE, SMFE, SPSE, SGDE, SISE, SLAE^{32} Course Instructor

Notice

This video is part of the **SecurityTube Linux Assembly Expert** course. A limited number of videos in this course will be released on SecurityTube.net

For full access to all theory and exercise videos, PDF slides, Code snippets etc. Please register as a student by visiting:

http://securitytube-training.com

Module 1: 32-Bit ASM on Linux

9. Control Instructions

Vivek Ramachandran
SWSE, SMFE, SPSE, SGDE, SISE, SLAE^{32} Course Instructor

http://SecurityTube-Training.com

Control Instructions

- Controls the flow of the program
- Based on “events” e.g. calculation led to 0
- Uses flags to determine decision
- Branching
  - Unconditional – JMP
  - Conditional - Jxx

JMP

- Unconditional
  - compare it with the GOTO statement in C
- Types:
  - Near Jump: Current Code Segment
    - Short: -128 to +127 from current position
  - Far Jump: In another Segment

Jxx

- Jxx – Conditional
  - JZ, JNZ, JA, JAE, JC, JNC etc.
  - uses flags
- Cannot be used for Far Jumps
  - JNZ label1
  - JMP Far_Label
  - lable1:
- Intel Manual is the best reference

Exercise 1.9.1

- Investigate the use of the following instructions – LOOP, LOOPZ, LOOPNZ etc.

- Use GDB to trace through the execution

Please Register for this course to receive the solution video for this exercise.
http://SecurityTube-Training.com

SecurityTube Linux Assembly Expert (SLAE^{32})

SecurityTube Linux Assembly Expert

Training: http://www.SecurityTube-Training.com

Community: http://www.SecurityTube.net

Vivek Ramachandran
SWSE, SMFE, SPSE, SGDE, SISE, SLAE^{32} Course Instructor

Notice

This video is part of the **SecurityTube Linux Assembly Expert** course. A limited number of videos in this course will be released on SecurityTube.net

For full access to all theory and exercise videos, PDF slides, Code snippets etc. Please register as a student by visiting:

http://securitytube-training.com

Exercise 1.9.1

- Investigate the use of the following instructions – LOOP, LOOPE, LOOPNE etc.

- Use GDB to trace through the execution

SecurityTube Linux Assembly Expert (SLAE^{32})

SecurityTube Linux Assembly Expert

Training: http://www.SecurityTube-Training.com

Community: http://www.SecurityTube.net

Vivek Ramachandran
SWSE, SMFE, SPSE, SGDE, SISE, SLAE^{32} Course Instructor

Notice

This video is part of the **SecurityTube Linux Assembly Expert** course. A limited number of videos in this course will be released on SecurityTube.net

For full access to all theory and exercise videos, PDF slides, Code snippets etc. Please register as a student by visiting:

http://securitytube-training.com

Module 1: 32-Bit ASM on Linux

10. Procedures

Vivek Ramachandran
SWSE, SMFE, SPSE, SGDE, SISE, SLAE^{32} Course Instructor

http://SecurityTube-Training.com

Procedure

- Set of operations grouped together
- Called often from different places in the code
- CALL Procedure_Name
- In NASM procedures are defined using Labels

Format of a Procedure

ProcedureName:

… code…

… code …

… code …

RET

Arguments to a Procedure

- Passed via Registers
- Passed on the stack
- Passed as data structures in memory referenced by registers / or on stack

Saving and Restoring State

- Saving / Restoring Registers
  - PUSHAD / POPAD
- Saving / Restoring Flags
  - PUSHFD / POPFD
- Saving / Restoring
  - ENTER / LEAVE + RET

Exercise 1.10.1

- Write a program which saves registers and flags before calling a procedure
  - it should also save / restore the frame pointer

Please Register for this course to receive the solution video for this exercise.
http://SecurityTube-Training.com

SecurityTube Linux Assembly Expert (SLAE^{32})

SecurityTube Linux Assembly Expert

Training: http://www.SecurityTube-Training.com

Community: http://www.SecurityTube.net

Vivek Ramachandran
SWSE, SMFE, SPSE, SGDE, SISE, SLAE^{32} Course Instructor

Notice

This video is part of the **SecurityTube Linux Assembly Expert** course. A limited number of videos in this course will be released on SecurityTube.net

For full access to all theory and exercise videos, PDF slides, Code snippets etc. Please register as a student by visiting:

http://securitytube-training.com

Exercise 1.10.1

- Write a program which saves registers and flags before calling a procedure
  - it should also save / restore the frame pointer

Prologue and Epilogue

- Wikipedia:

https://en.wikipedia.org/wiki/Function_prologue

- Space is reserved for storing local variables

SecurityTube Linux Assembly Expert (SLAE^{32})

SecurityTube Linux Assembly Expert

Training: http://www.SecurityTube-Training.com

Community: http://www.SecurityTube.net

Vivek Ramachandran
SWSE, SMFE, SPSE, SGDE, SISE, SLAE^{32} Course Instructor

Notice

This video is part of the **SecurityTube Linux Assembly Expert** course. A limited number of videos in this course will be released on SecurityTube.net

For full access to all theory and exercise videos, PDF slides, Code snippets etc. Please register as a student by visiting:

http://securitytube-training.com

Module 1: 32-Bit ASM on Linux

11. Strings

Vivek Ramachandran
SWSE, SMFE, SPSE, SGDE, SISE, SLAE^{32} Course Instructor

http://SecurityTube-Training.com

String Instructions

- MOVS (MOVS / MOVSW / MOVSD)
- CMPS - Compares
- SCAS - Subtracts
- LODS - Loads

ESI and EDI registers are typically used with DF

SecurityTube Linux Assembly Expert (SLAE^{32})

SecurityTube Linux Assembly Expert

Training: http://www.SecurityTube-Training.com

Community: http://www.SecurityTube.net

Vivek Ramachandran
SWSE, SMFE, SPSE, SGDE, SISE, SLAE^{32} Course Instructor

Notice

This video is part of the **SecurityTube Linux Assembly Expert** course. A limited number of videos in this course will be released on SecurityTube.net

For full access to all theory and exercise videos, PDF slides, Code snippets etc. Please register as a student by visiting:

http://securitytube-training.com

Module 1: 32-Bit ASM on Linux

11. Libc and NASM

Vivek Ramachandran
SWSE, SMFE, SPSE, SGDE, SISE, SLAE^{32} Course Instructor

http://SecurityTube-Training.com

Syscalls are good but …

- Too low level at times
- Standard C Library – libc has tons of useful functions
- Calling Libc functions from Assembly

Things to Remember

- Define all libc functions you want use with extern
- All arguments in reverse order on stack
  - CALL function(a,b,c,d)
  - push d, push c, push b, push a
- Adjust the stack after calling libc functions
- Link with GCC rather than LD – use main instead of _start

SecurityTube Linux Assembly Expert (SLAE^{32})

SecurityTube Linux Assembly Expert

Training: http://www.SecurityTube-Training.com

Community: http://www.SecurityTube.net

Vivek Ramachandran
SWSE, SMFE, SPSE, SGDE, SISE, SLAE^{32} Course Instructor

Notice

This video is part of the **SecurityTube Linux Assembly Expert** course. A limited number of videos in this course will be released on SecurityTube.net

For full access to all theory and exercise videos, PDF slides, Code snippets etc. Please register as a student by visiting:

http://securitytube-training.com

Module 2: Introduction to Shellcoding

1. Shellcoding Basics

Vivek Ramachandran
SWSE, SMFE, SPSE, SGDE, SISE, SLAE^{32} Course Instructor

http://SecurityTube-Training.com

What is Shellcode?

- Machine code with a 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 separate compiling required

How is Shellcode delivered?

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

Shellcode Resources

- http://www.shell-storm.org/
- http://exploit-db.com
- http://www.projectshellcode.com/

Testing Shellcode

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

unsigned char code[] = \
"SHELLCODE ";


main()
{

        printf("Shellcode Length:  %d\n", strlen(code));

        int (*ret)() = (int(*)())code;

        ret();

}


~
```

SecurityTube Linux Assembly Expert (SLAE^{32})

SecurityTube Linux Assembly Expert

Training: http://www.SecurityTube-Training.com

Community: http://www.SecurityTube.net

Vivek Ramachandran
SWSE, SMFE, SPSE, SGDE, SISE, SLAE^{32} Course Instructor

Notice

This video is part of the **SecurityTube Linux Assembly Expert** course. A limited number of videos in this course will be released on SecurityTube.net

For full access to all theory and exercise videos, PDF slides, Code snippets etc. Please register as a student by visiting:

http://securitytube-training.com

Module 2: Introduction to Shellcoding

2. Exit Shellcode

Vivek Ramachandran
SWSE, SMFE, SPSE, SGDE, SISE, SLAE^{32} Course Instructor

http://SecurityTube-Training.com

Exit Shellcode

- Use of EAX and EBX registers with the syscall number and exit code leads to 0x00 in shellcode
- Change instructions to avoid 0x00
- Change instruction to make shellcode more compact

Objdump to Shellcode

- Command line FU

- http://www.commandlinefu.com/commands/view/6051/get-all-shellcode-on-binary-file-from-objdump

SecurityTube Linux Assembly Expert (SLAE^{32})

SecurityTube Linux Assembly Expert

Training: http://www.SecurityTube-Training.com

Community: http://www.SecurityTube.net

Vivek Ramachandran
SWSE, SMFE, SPSE, SGDE, SISE, SLAE^{32} Course Instructor

Notice

This video is part of the **SecurityTube Linux Assembly Expert** course. A limited number of videos in this course will be released on SecurityTube.net

For full access to all theory and exercise videos, PDF slides, Code snippets etc. Please register as a student by visiting:

http://securitytube-training.com

Module 2: Introduction to Shellcoding

3. HelloWorld Shellcode using JMP-CALL-POP

Vivek Ramachandran
SWSE, SMFE, SPSE, SGDE, SISE, SLAE^{32} Course Instructor

http://SecurityTube-Training.com

Modifying Hello World

- Replace all 0x00 opcode instructions
- No hardcoded addresses
  - dynamically figure out address of “Hello World” string

JMP-CALL-POP

JMP short Call_shellcode:

shellcode:

pop ecx

....

...

...

Call_shellcode:

call shellcode:

HelloWorld db “Hello World!”

SecurityTube Linux Assembly Expert (SLAE^{32})

SecurityTube Linux Assembly Expert

Training: http://www.SecurityTube-Training.com

Community: http://www.SecurityTube.net

Vivek Ramachandran
SWSE, SMFE, SPSE, SGDE, SISE, SLAE^{32} Course Instructor

Notice

This video is part of the **SecurityTube Linux Assembly Expert** course. A limited number of videos in this course will be released on SecurityTube.net

For full access to all theory and exercise videos, PDF slides, Code snippets etc. Please register as a student by visiting:

http://securitytube-training.com

Module 2: Introduction to Shellcoding

4. HelloWorld Shellcode using Stack

Vivek Ramachandran
SWSE, SMFE, SPSE, SGDE, SISE, SLAE^{32} Course Instructor

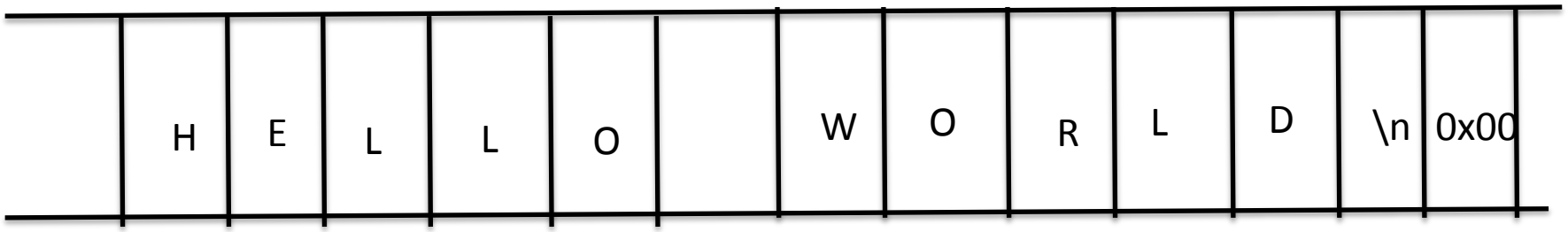
http://SecurityTube-Training.com

Using the Stack

- PUSH the value of the “Hello World” string on the stack
- Get a reference using ESP
- String needs to be pushed in reverse as stack grows from High to Low memory

Stack grows from High memory to Low memory

High Memory



Low Memory

SecurityTube Linux Assembly Expert (SLAE^{32})

SecurityTube Linux Assembly Expert

Training: http://www.SecurityTube-Training.com

Community: http://www.SecurityTube.net

Vivek Ramachandran
SWSE, SMFE, SPSE, SGDE, SISE, SLAE^{32} Course Instructor

Notice

This video is part of the **SecurityTube Linux Assembly Expert** course. A limited number of videos in this course will be released on SecurityTube.net

For full access to all theory and exercise videos, PDF slides, Code snippets etc. Please register as a student by visiting:

http://securitytube-training.com

Module 2: Introduction to Shellcoding

5. Execve Shellcode JMP-CALL-POP Method

Vivek Ramachandran
SWSE, SMFE, SPSE, SGDE, SISE, SLAE^{32} Course Instructor

http://SecurityTube-Training.com

Execute a new program

- execute a new program from within the shellcode
- “/bin/bash” to get a shell
- common technique to get a command prompt from an exploited process

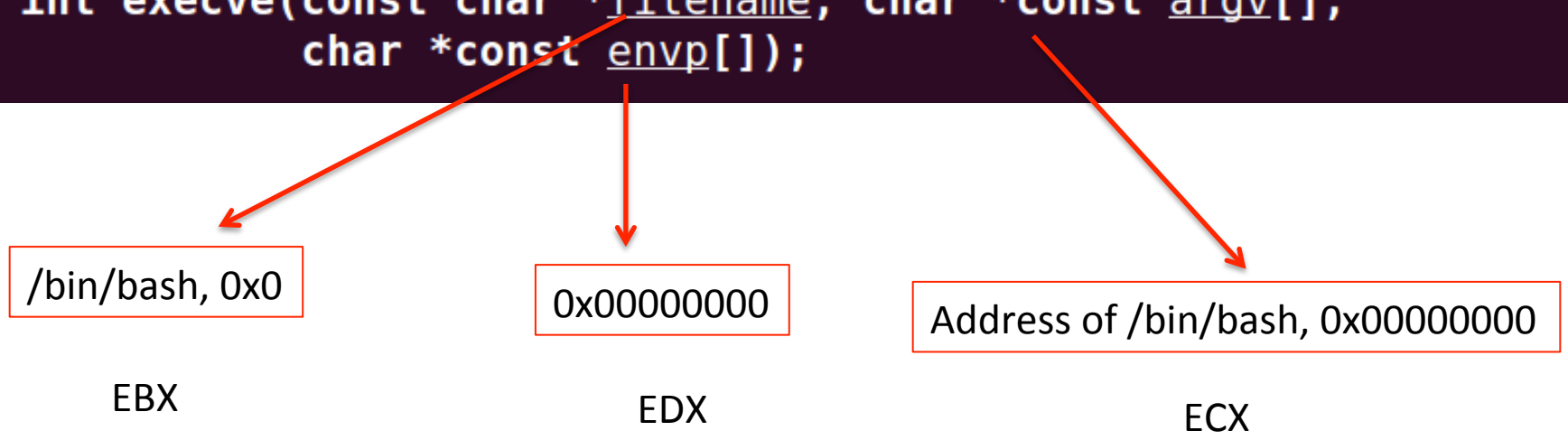
Execve

```
EXECVE(2)                          Linux Programmer's Manual

NAME
       execve - execute program

SYNOPSIS
       #include <unistd.h>

       int execve(const char *filename, char *const argv[],
                  char *const envp[]);
```

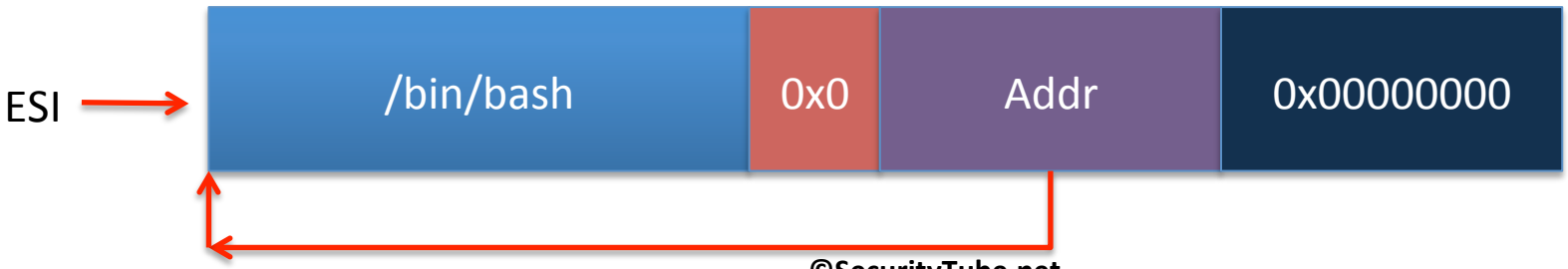


We cannot have NULLs in the Shellcode

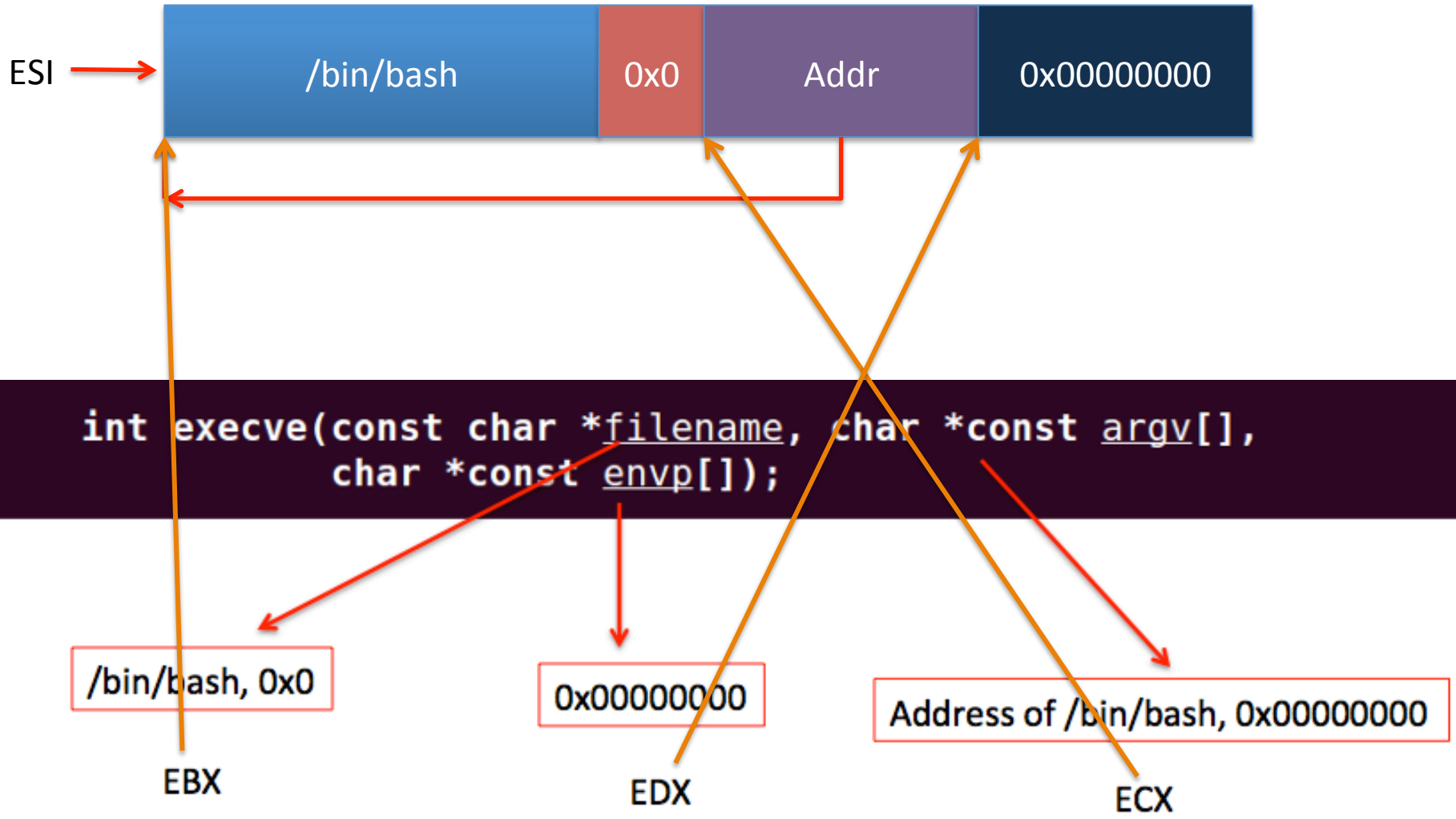
Approach

Initial String

1. Use JMP-CALL-POP to find the address of the string
2. Convert “A” to 0x0
3. Convert “BBBB” to address of “/bin/bash”
4. Convert “CCCC” to 0x00000000



Loading the Registers



Is there a need for exit()

- execve does not return if successful

- there is no need for exit() to be called

SecurityTube Linux Assembly Expert (SLAE^{32})

SecurityTube Linux Assembly Expert

Training: http://www.SecurityTube-Training.com

Community: http://www.SecurityTube.net

Vivek Ramachandran
SWSE, SMFE, SPSE, SGDE, SISE, SLAE^{32} Course Instructor

Notice

This video is part of the **SecurityTube Linux Assembly Expert** course. A limited number of videos in this course will be released on SecurityTube.net

For full access to all theory and exercise videos, PDF slides, Code snippets etc. Please register as a student by visiting:

http://securitytube-training.com

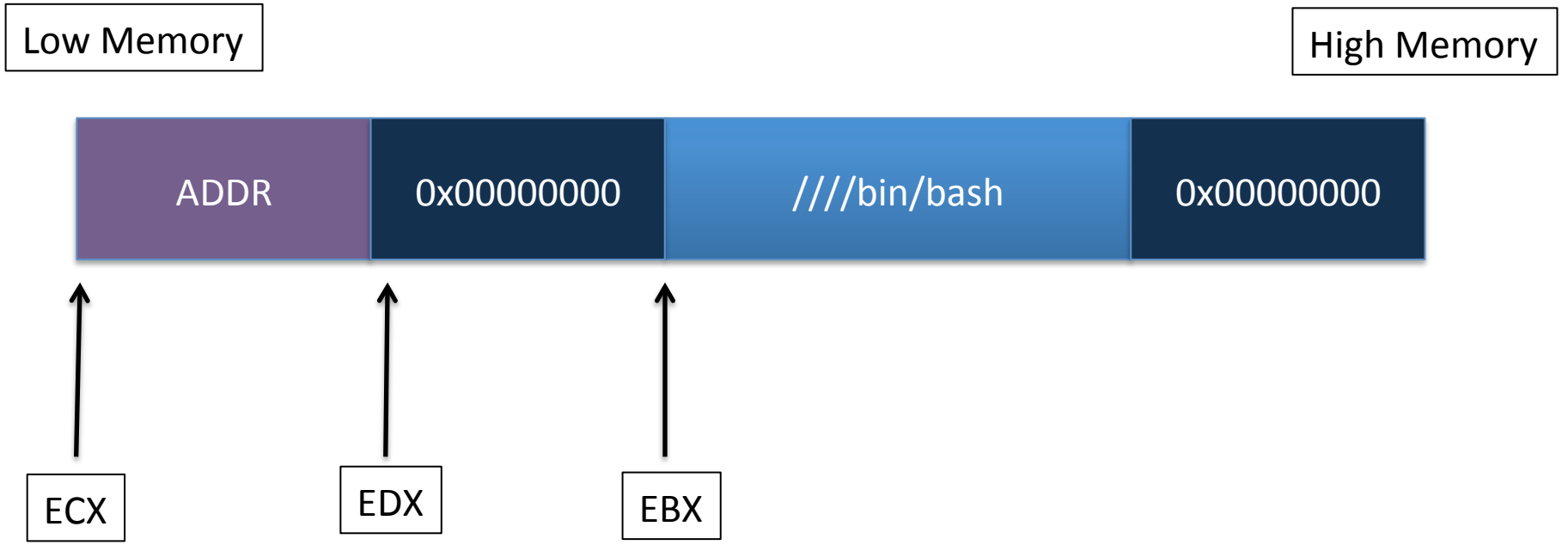
Module 2: Introduction to Shellcoding

6. Execve Shellcode Stack Method

Vivek Ramachandran
SWSE, SMFE, SPSE, SGDE, SISE, SLAE^{32} Course Instructor

http://SecurityTube-Training.com

Stack Push



SecurityTube Linux Assembly Expert (SLAE^{32})

SecurityTube Linux Assembly Expert

Training: http://www.SecurityTube-Training.com

Community: http://www.SecurityTube.net

Vivek Ramachandran
SWSE, SMFE, SPSE, SGDE, SISE, SLAE^{32} Course Instructor

Notice

This video is part of the **SecurityTube Linux Assembly Expert** course. A limited number of videos in this course will be released on SecurityTube.net

For full access to all theory and exercise videos, PDF slides, Code snippets etc. Please register as a student by visiting:

http://securitytube-training.com

Module 2: Introduction to Shellcoding

7. XOR Encoder and Decoder

Vivek Ramachandran
SWSE, SMFE, SPSE, SGDE, SISE, SLAE^{32} Course Instructor

http://SecurityTube-Training.com

XOR

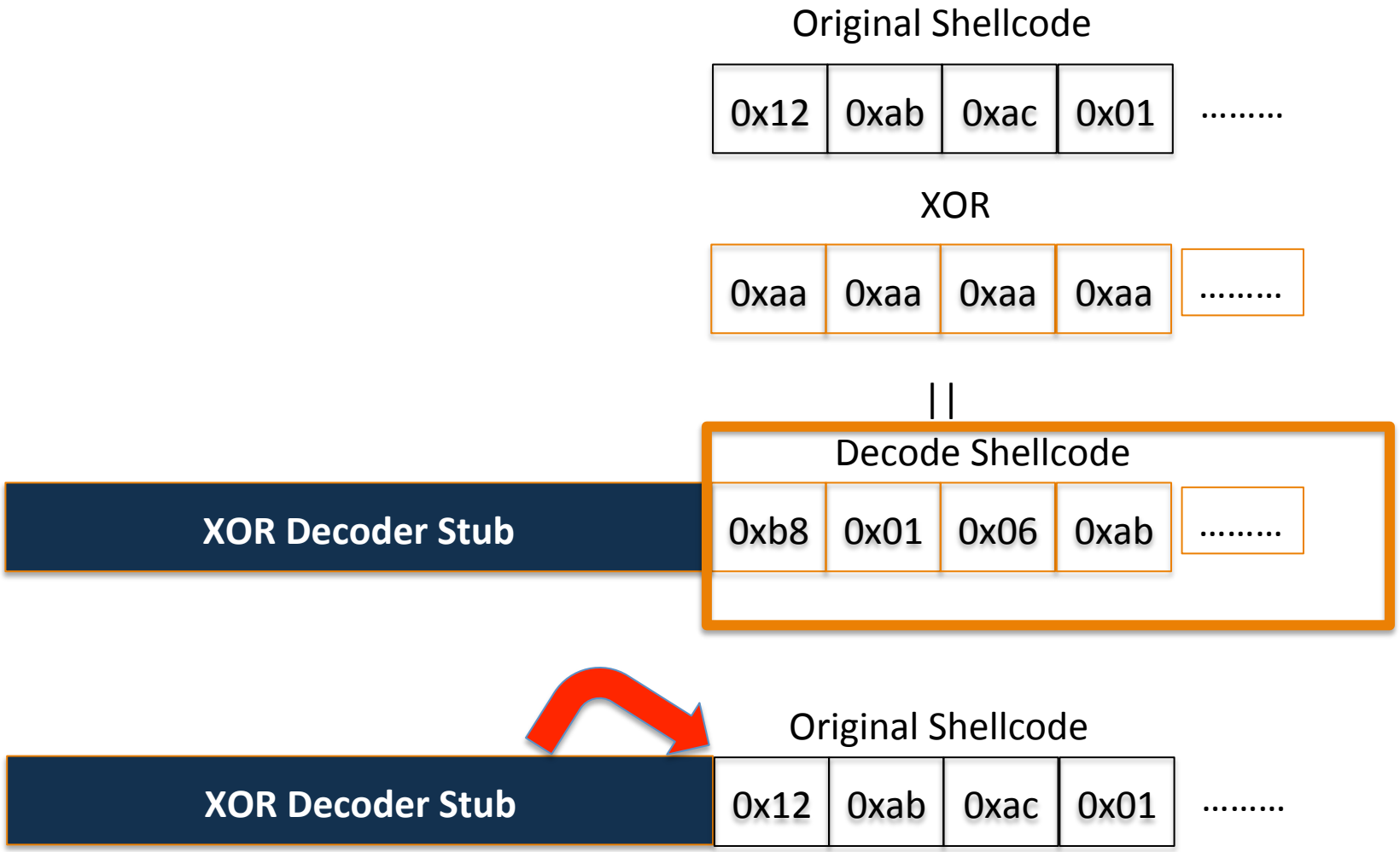
A	B	A xor B
0	0	0
1	1	0
1	0	1
0	1	1

Interesting: (A xor B) xor B = A

What does this mean for us?

- Select an encoder byte e.g. 0xAA
- XOR every byte of Shellcode with 0xAA
- Write a decoder stub which will XOR the encoded shellcode bytes with 0xAA and recover original shellcode
- Stub then passes control to decoded shellcode

Too much text can kill a concept ☺



SecurityTube Linux Assembly Expert (SLAE^{32})

SecurityTube Linux Assembly Expert

Training: http://www.SecurityTube-Training.com

Community: http://www.SecurityTube.net

Vivek Ramachandran
SWSE, SMFE, SPSE, SGDE, SISE, SLAE^{32} Course Instructor

Notice

This video is part of the **SecurityTube Linux Assembly Expert** course. A limited number of videos in this course will be released on SecurityTube.net

For full access to all theory and exercise videos, PDF slides, Code snippets etc. Please register as a student by visiting:

http://securitytube-training.com

Module 2: Introduction to Shellcoding

8. Using Metasploit's Encoders

Vivek Ramachandran
SWSE, SMFE, SPSE, SGDE, SISE, SLAE^{32} Course Instructor

http://SecurityTube-Training.com

Metasploit Payloads

```
securitytube@securitytube-VirtualBox:~/SLAE/Shellcode/Metasploit$ sudo msfpayload -l | grep "linux/x86"
    linux/x86/adduser                                Create a new user with UID 0
    linux/x86/chmod                                  Runs chmod on specified file with specified mode
    linux/x86/exec                                   Execute an arbitrary command
    linux/x86/meterpreter/bind_ipv6_tcp              Listen for a connection over IPv6, Staged meterpreter server
    linux/x86/meterpreter/bind_nonx_tcp              Listen for a connection, Staged meterpreter server
    linux/x86/meterpreter/bind_tcp                   Listen for a connection, Staged meterpreter server
    linux/x86/meterpreter/find_tag                   Use an established connection, Staged meterpreter server
    linux/x86/meterpreter/reverse_ipv6_tcp           Connect back to attacker over IPv6, Staged meterpreter server
    linux/x86/meterpreter/reverse_nonx_tcp           Connect back to the attacker, Staged meterpreter server
    linux/x86/meterpreter/reverse_tcp                Connect back to the attacker, Staged meterpreter server
    linux/x86/metsvc_bind_tcp                        Stub payload for interacting with a Meterpreter Service
    linux/x86/metsvc_reverse_tcp                     Stub payload for interacting with a Meterpreter Service
    linux/x86/read_file                              Read up to 4096 bytes from the local file system and write it
back out to the specified file descriptor
    linux/x86/shell/bind_ipv6_tcp                    Listen for a connection over IPv6, Spawn a command shell (stag
ed)
    linux/x86/shell/bind_nonx_tcp                    Listen for a connection, Spawn a command shell (staged)
    linux/x86/shell/bind_tcp                         Listen for a connection, Spawn a command shell (staged)
    linux/x86/shell/find_tag                         Use an established connection, Spawn a command shell (staged)
    linux/x86/shell/reverse_ipv6_tcp                 Connect back to attacker over IPv6, Spawn a command shell (sta
ged)
    linux/x86/shell/reverse_nonx_tcp                 Connect back to the attacker, Spawn a command shell (staged)
    linux/x86/shell/reverse_tcp                      Connect back to the attacker, Spawn a command shell (staged)
    linux/x86/shell_bind_ipv6_tcp                    Listen for a connection over IPv6 and spawn a command shell
    linux/x86/shell_bind_tcp                         Listen for a connection and spawn a command shell
    linux/x86/shell_find_port                        Spawn a shell on an established connection
    linux/x86/shell_find_tag                         Spawn a shell on an established connection (proxy/nat safe)
    linux/x86/shell_reverse_tcp                      Connect back to attacker and spawn a command shell
    linux/x86/shell_reverse_tcp2                     Connect back to attacker and spawn a command shell
securitytube@securitytube-VirtualBox:~/SLAE/Shellcode/Metasploit$
```

Metasploit Encoders

Name	Rank	Description
cmd/generic_sh	good	Generic Shell Variable Substitution Command Encoder
cmd/ifs	low	Generic ${IFS} Substitution Command Encoder
cmd/printf_php_mq	manual	printf(1) via PHP magic_quotes Utility Command Encoder
generic/none	normal	The "none" Encoder
mipsbe/longxor	normal	XOR Encoder
mipsle/longxor	normal	XOR Encoder
php/base64	great	PHP Base64 Encoder
ppc/longxor	normal	PPC LongXOR Encoder
ppc/longxor_tag	normal	PPC LongXOR Encoder
sparc/longxor_tag	normal	SPARC DWORD XOR Encoder
x64/xor	normal	XOR Encoder
x86/alpha_mixed	low	Alpha2 Alphanumeric Mixedcase Encoder
x86/alpha_upper	low	Alpha2 Alphanumeric Uppercase Encoder
x86/avoid_underscore_tolower	manual	Avoid underscore/tolower
x86/avoid_utf8_tolower	manual	Avoid UTF8/tolower
x86/call4_dword_xor	normal	Call+4 Dword XOR Encoder
x86/context_cpuid	manual	CPUID-based Context Keyed Payload Encoder
x86/context_stat	manual	stat(2)-based Context Keyed Payload Encoder
x86/context_time	manual	time(2)-based Context Keyed Payload Encoder
x86/countdown	normal	Single-byte XOR Countdown Encoder
x86/fnstenv_mov	normal	Variable-length Fnstenv/mov Dword XOR Encoder
x86/jmp_call_additive	normal	Jump/Call XOR Additive Feedback Encoder
x86/nonalpha	low	Non-Alpha Encoder
x86/nonupper	low	Non-Upper Encoder
x86/shikata_ga_nai	excellent	Polymorphic XOR Additive Feedback Encoder
x86/single_static_bit	manual	Single Static Bit
x86/unicode_mixed	manual	Alpha2 Alphanumeric Unicode Mixedcase Encoder
x86/unicode_upper	manual	Alpha2 Alphanumeric Unicode Uppercase Encoder

Leverage Metasploit

- Create our own Shellcode

- Use Msfencode to encode it
  - use encoded shellcode
  - dump into binary and execute

AV and IDS Evasion

- Almost all Encoders in Metasploit are well known and documented
- Might not be useful to create evasion
  - Shikata_ga_nai works at times
- Need for custom encoders

Custom Encoder

- Easy to write a custom encoder and bypass AV etc. till the time the technique is not disclosed
- Very difficult to write an encoder which has a public technique and evade AV
- Encoder Stub is the one which is generally fingerprinted

SecurityTube Linux Assembly Expert (SLAE^{32})

SecurityTube Linux Assembly Expert

Training: http://www.SecurityTube-Training.com

Community: http://www.SecurityTube.net

Vivek Ramachandran
SWSE, SMFE, SPSE, SGDE, SISE, SLAE^{32} Course Instructor

Notice

This video is part of the **SecurityTube Linux Assembly Expert** course. A limited number of videos in this course will be released on SecurityTube.net

For full access to all theory and exercise videos, PDF slides, Code snippets etc. Please register as a student by visiting:

http://securitytube-training.com

Module 2: Introduction to Shellcoding

9. Simple NOT encoder

Vivek Ramachandran
SWSE, SMFE, SPSE, SGDE, SISE, SLAE^{32} Course Instructor

http://SecurityTube-Training.com

NOT Encoder

- Transform every byte in your shellcode using NOT
- Decoder will NOT the encoded byte to get the original shellcode byte
- Pass control to shellcode after all bytes decoded

SecurityTube Linux Assembly Expert (SLAE^{32})

SecurityTube Linux Assembly Expert

Training: http://www.SecurityTube-Training.com

Community: http://www.SecurityTube.net

Vivek Ramachandran
SWSE, SMFE, SPSE, SGDE, SISE, SLAE^{32} Course Instructor

Notice

This video is part of the **SecurityTube Linux Assembly Expert** course. A limited number of videos in this course will be released on SecurityTube.net

For full access to all theory and exercise videos, PDF slides, Code snippets etc. Please register as a student by visiting:

http://securitytube-training.com

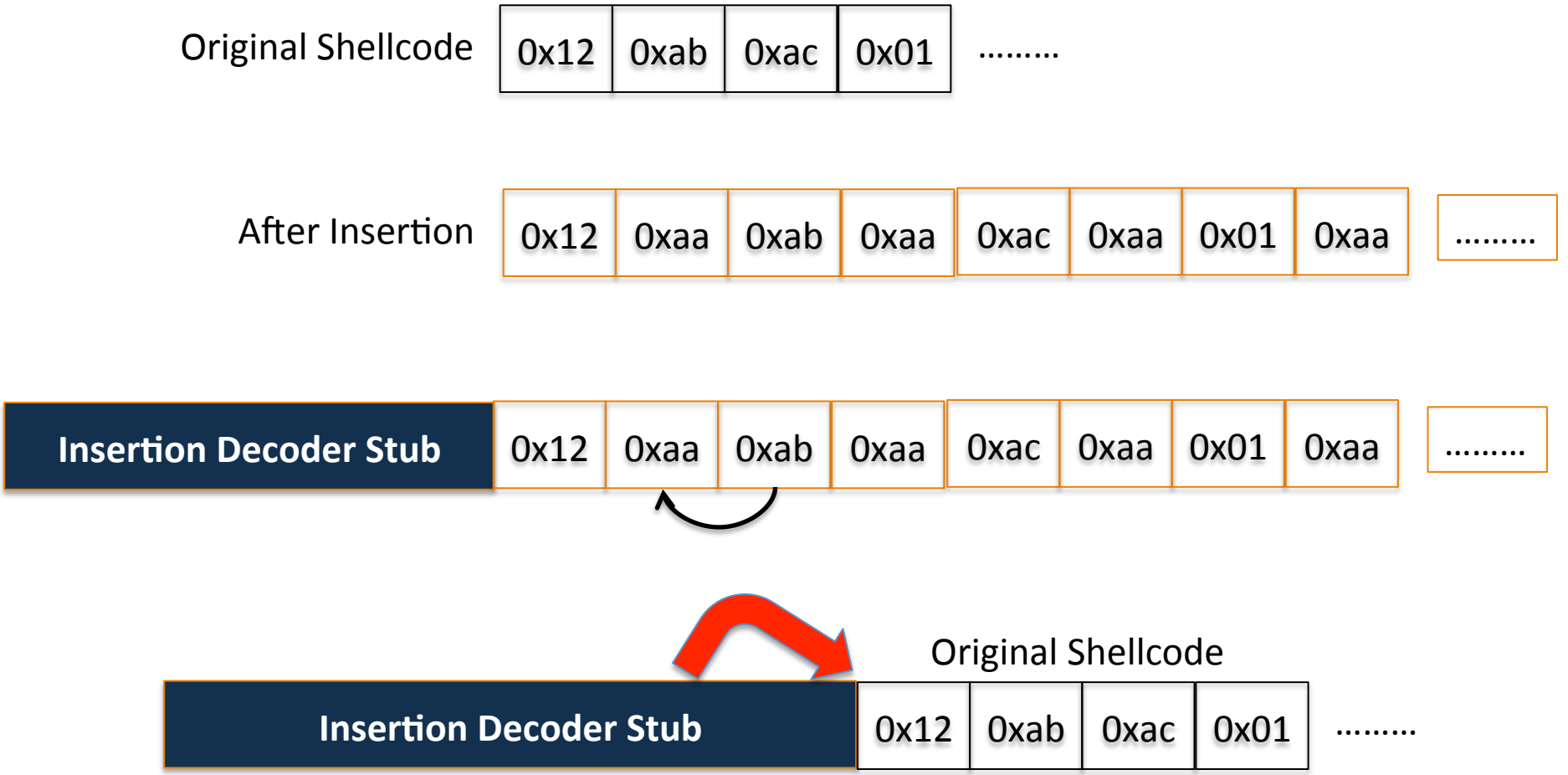
Module 2: Introduction to Shellcoding

10. Insertion Encoder

Vivek Ramachandran
SWSE, SMFE, SPSE, SGDE, SISE, SLAE^{32} Course Instructor

http://SecurityTube-Training.com

Insertion Encoder?



SecurityTube Linux Assembly Expert (SLAE^{32})

SecurityTube Linux Assembly Expert

Training: http://www.SecurityTube-Training.com

Community: http://www.SecurityTube.net

Vivek Ramachandran
SWSE, SMFE, SPSE, SGDE, SISE, SLAE^{32} Course Instructor

Notice

This video is part of the **SecurityTube Linux Assembly Expert** course. A limited number of videos in this course will be released on SecurityTube.net

For full access to all theory and exercise videos, PDF slides, Code snippets etc. Please register as a student by visiting:

http://securitytube-training.com

Module 2: Introduction to Shellcoding

11. XOR Decoder using MMX

Vivek Ramachandran
SWSE, SMFE, SPSE, SGDE, SISE, SLAE^{32} Course Instructor

http://SecurityTube-Training.com

Ton of instructions!

- FPU
- MMX
- SSE
- SSE2

Advantages

- Existing “popular” shellcodes hardly use them
- Probably detection rates lesser by AV and other analysis tools
- Easy to replicate existing functionality using these extensions

MMX based XOR Decoder

- SIMD – Single instruction multiple data
- Registers MM0 to MM7
- Can load 8 bytes qword
- Moving Data – movq
- XOR'ing Data – pxor
- Key Difference from the previous XOR decoder
  - Operates over 8 bytes at the same time

Using the FPU for GetPC

FSTENV GetPC

[todo:References]

A way to retrieve the value of EIP on x86 systems is to use the x87 `FSTENV` instruction to store the state of the x87 floating point chip after issuing a `FLDZ` instruction. The structure store in memory by `FSTENV` will then contain the address of the `FLDZ` instruction at offset 0x0C of the structure:

```
$+0     D9EE          FLDZ                    ; Floating point stores $+0 in its environment
$+2     D974E4 F4     FSTENV SS:[ESP-0xC]     ; Save environment at ESP-0xC; now [ESP] = $+0
$+6     59            POP ECX                 ; ECX = $+0
```

Source: http://skypher.com/wiki/index.php/Hacking/Shellcode/GetPC

FSTENV stores control, status and tag word, **instruction pointer**, data pointer and last opcode

SecurityTube Linux Assembly Expert (SLAE^{32})

SecurityTube Linux Assembly Expert

Training: http://www.SecurityTube-Training.com

Community: http://www.SecurityTube.net

Vivek Ramachandran
SWSE, SMFE, SPSE, SGDE, SISE, SLAE^{32} Course Instructor

Notice

This video is part of the **SecurityTube Linux Assembly Expert** course. A limited number of videos in this course will be released on SecurityTube.net

For full access to all theory and exercise videos, PDF slides, Code snippets etc. Please register as a student by visiting:

http://securitytube-training.com

Module 2: Introduction to Shellcoding

12. Polymorphism

Vivek Ramachandran
SWSE, SMFE, SPSE, SGDE, SISE, SLAE^{32} Course Instructor

http://SecurityTube-Training.com

Easy Fingerprinting of Basic Shellcode

- AV and IDS can use the shellcode as a pattern to search
- Easy to fingerprint
- Detection simple

Encoding and Encryption

- Original shellcode protected
- Decoder / Decryptor Stub however small, is prone to fingerprinting
- Back to square 1 ☺

Imagine IF

- We could make our shellcode look different everytime we create it
- Functionality remains the same
- Semantically equivalent instructions

Detection is now MUCH, MUCH Difficult

Enter Polymorphism

Origins in the Virus World

```
----[ 3.1 - Back in 1992...

In 1992, Dark Avenger invented a revolutionary technique he called
polymorphism. What is it ? It simply consist of ciphering the code of the
virus and generate a decipher routine which is different at each time, so
that the whole virus is different at each time and can't be scanned !

Very good polymorphic engines have appeared : the TridenT Polymorphic
Engine (TPE), Dark Angel Mutation Engine (DAME).

As a consequence, antivirus makers developped new heuristic techniques
such as spectrum analysis, code emulators, ...
```

Source: http://www.phrack.org/issues.html?issue=61&id=9#article

Basic Principle of Create Polymorphic Shellcode

- Replace instructions with equivalent functionality ones
- Add garbage instructions which don't change functionality in any way "NOP Equivalents"

Polymorphic Engines

- ADMutate:
  - http://www.ktwo.ca/readme.html
  - http://www.youtube.com/watch?v=XMt9ExL9I00
- CLET
  - http://www.phrack.org/issues.html?issue=61&id=9#article
- VX Heavens Mirror
  - http://download.adamas.ai/dlbase/Stuff/VX%20Heavens%20Library/static/vdat/mainmenu.htm

SecurityTube Linux Assembly Expert (SLAE^{32})

SecurityTube Linux Assembly Expert

Training: http://www.SecurityTube-Training.com

Community: http://www.SecurityTube.net

Vivek Ramachandran
SWSE, SMFE, SPSE, SGDE, SISE, SLAE^{32} Course Instructor

Notice

This video is part of the **SecurityTube Linux Assembly Expert** course. A limited number of videos in this course will be released on SecurityTube.net

For full access to all theory and exercise videos, PDF slides, Code snippets etc. Please register as a student by visiting:

http://securitytube-training.com

Module 2: Introduction to Shellcoding

13. Analyzing 3rd Party Shellcode

Vivek Ramachandran
SWSE, SMFE, SPSE, SGDE, SISE, SLAE32 Course Instructor

http://SecurityTube-Training.com

Should I run this?

```
"\x6A\x7F\x5A\x54\x59\x31\xDB\x6A\x03\x58\xCD\x80\x51\xC3"
```

Analyzing Shellcode

- Use GDB

- Use Ndisasm

- Libemu – Shellcode emulation
  http://libemu.carnivore.it/

Staged Shellcode

- Divided into 2 stages:
  - First stage is small and loads the second stage
    - from input
    - from a file / over a network
  - First stage passes control to the second stage
- Useful when very less space to run shellcode in an exploit

Case Study: Analyzing Staged Shellcode

```
/*
 * (linux/x86) stagger that reads second stage shellcode (127 bytes maximum) from stdin - 14 bytes
 * _fkz / twitter: @_fkz
 *
 * sc = "\x6A\x7F\x5A\x54\x59\x31\xDB\x6A\x03\x58\xCD\x80\x51\xC3"
 *
 * Example of use:
 * (echo -ne "\xseconde stage shellcode\x"; cat) | ./stager
 */

 char shellcode[] =

                "\x6A\x7F"              //      push    byte    +0x7F
                "\x5A"                  //      pop             edx
                "\x54"                  //      push    esp
                "\x59"                  //      pop             esp
                "\x31\xDB"              //      xor             ebx,ebx
                "\x6A\x03"              //      push    byte    +0x3
                "\x58"                  //      pop             eax
                "\xCD\x80"              //      int             0x80
                "\x51"                  //      push    ecx
                "\xC3";                 //      ret
```

Source: http://www.shell-storm.org/shellcode/files/shellcode-824.php

Network based second stage loading

- Bind TCP (staged)
- Reverse TCP (staged)
- Meterpreter is staged ☺

SecurityTube Linux Assembly Expert (SLAE^{32})

SecurityTube Linux Assembly Expert

Training: http://www.SecurityTube-Training.com

Community: http://www.SecurityTube.net

Vivek Ramachandran
SWSE, SMFE, SPSE, SGDE, SISE, SLAE^{32} Course Instructor

Notice

This video is part of the **SecurityTube Linux Assembly Expert** course. A limited number of videos in this course will be released on SecurityTube.net

For full access to all theory and exercise videos, PDF slides, Code snippets etc. Please register as a student by visiting:

http://securitytube-training.com

Module 2: Introduction to Shellcoding

14. Analyzing Shell_Bind_TCP and Shell_Reverse_TCP with Libemu

Vivek Ramachandran
SWSE, SMFE, SPSE, SGDE, SISE, SLAE^{32} Course Instructor

http://SecurityTube-Training.com

Shell_XXX_TCP

- Binding a Shell is the most common way to get command line access to a compromised system
- Bind Shell (Listening on victim's port)
- Reverse Shell (Connects back to Attacker's port)

Libemu

http://libemu.carnivore.it/

SecurityTube Linux Assembly Expert (SLAE^{32})

SecurityTube Linux Assembly Expert

Training: http://www.SecurityTube-Training.com

Community: http://www.SecurityTube.net

Vivek Ramachandran
SWSE, SMFE, SPSE, SGDE, SISE, SLAE^{32} Course Instructor

Notice

This video is part of the **SecurityTube Linux Assembly Expert** course. A limited number of videos in this course will be released on SecurityTube.net

For full access to all theory and exercise videos, PDF slides, Code snippets etc. Please register as a student by visiting:

http://securitytube-training.com

Module 2: Introduction to Shellcoding

14. Crypters

Vivek Ramachandran
SWSE, SMFE, SPSE, SGDE, SISE, SLAE^{32} Course Instructor

http://SecurityTube-Training.com

Crypters

- Encrypt Executable / Shellcode
- Decrypt at runtime and run
- For powerful crypto techniques like RC4, AES etc. a lot of assembly code
- Shellcode size too large to be useful

RC4

- Symmetric Stream Cipher
- 2 Step process:
  - Key Scheduling Algorithm
  - Pseudo Random Number Generation
- Full Details: http://en.wikipedia.org/wiki/RC4

Writing an RC4 Shellcode Crypter in C

- Encryption Phase:
  - For a given key, encrypts shellcode
- Decryption Phase:
  - For the same key, decrypts shellcode
  - Executes it

Chaining Methods

- Create Shellcode
- Encode with XOR
- Encrypt with Crypter (needs to be the last)

RC4 in Assembly

- https://thunked.org/programming/rc4-in-assembly-t23.html
- http://youritguy.wordpress.com/2010/06/13/adler-32-and-rc4-in-inline-assembly/
- http://nayuki.eigenstate.org/page/rc4-cipher-in-x86-assembly

Hyperion

- PE Cryptor
- http://www.exploit-db.com/wp-content/themes/exploit/docs/18849.pdf
- Encrypted (AES) with weak key
- Key bruteforced at runtime

SecurityTube Linux Assembly Expert (SLAE^{32})

SecurityTube Linux Assembly Expert

Training: http://www.SecurityTube-Training.com

Community: http://www.SecurityTube.net

Vivek Ramachandran
SWSE, SMFE, SPSE, SGDE, SISE, SLAE^{32} Course Instructor

Notice

This video is part of the **SecurityTube Linux Assembly Expert** course. A limited number of videos in this course will be released on SecurityTube.net

For full access to all theory and exercise videos, PDF slides, Code snippets etc. Please register as a student by visiting:

http://securitytube-training.com

Module 3: Certification Exam

The Grand Finale

Vivek Ramachandran
SWSE, SMFE, SPSE, SGDE, SISE, SLAE^{32} Course Instructor

http://SecurityTube-Training.com

Exam Format

- 7 Assignments of varying difficulty
- Post solutions to your personal blog
  - wordpress.com, Blogger or your own domain
- Store code in a Github account

Assignment #1

- Create a Shell_Bind_TCP shellcode
  - Binds to a port
  - Execs Shell on incoming connection
- Port number should be easily configurable

Assignment #2

- Create a Shell_Reverse_TCP shellcode
  - Reverse connects to configured IP and Port
  - Execs shell on successful connection
- IP and Port should be easily configurable

Assignment #3

- Study about the Egg Hunter shellcode
- Create a working demo of the Egghunter
- Should be configurable for different payloads

Assignment #4

- Create a custom encoding scheme like the "Insertion Encoder" we showed you

- PoC with using execve-stack as the shellcode to encode with your schema and execute

Assignment #5

- Take up at least 3 shellcode samples created using Msfpayload for linux/x86
- Use GDB/Ndisasm/Libemu to dissect the functionality of the shellcode
- Present your analysis

Assignment #6

- Take up 3 shellcodes from Shell-Storm and create polymorphic versions of them to beat pattern matching
- The polymorphic versions cannot be larger 150% of the existing shellcode
- Bonus points for making it shorter in length than original

Assignment #7

- Create a custom crypter like the one shown in the “crypters” video
- Free to use any existing encryption schema
- Can use any programming language

Blog post must mention

This blog post has been created for completing the requirements of the SecurityTube Linux Assembly Expert certification:

http://securitytube-training.com/online-courses/securitytube-linux-assembly-expert/

Student ID: SLAE-XXXXX

Evaluation Criteria

- Originality of Shellcode
- Quality of Explanation – detailed and insightful
- Each Assignment carries 10 marks
- Certification Criteria: > 50 out of 70 marks

Extra Points ☺

- Posting additional new shellcodes beyond the assignments (10 points)

- Shellcode submitted to and accepted by:

  – Shell-Storm.org
  – Exploit-db.com

  (10 points)

- Community Interaction (5 points)
  – Chatter on Twitter, Facebook
  – Comments on Blog posts

Submission Format

- Email to feedback@binarysecuritysolutions.com
- Subject: SLAE Exam Blog Posts
- Email contains:
  - Links to all 7 blog posts
  - Link to Gitgub account where code is stored
  - Link to Shell-Storm / Exploit-db submissions
  - Link to Twitter / Facebook if posted there
- Around 5 working days for result