Hacking Defined Experts
For those who dare

Friday, August 2, 2013

# ELFs & Other Mystical Creatures

An introduction to reverse engineering in a Linux environment for those Linux binaries

# Overview

- Start with some basics
- Static View
- Dive Deeper
- Now set fire to the rain ;)

# Before We Start

- Get these things:
- http://hackingdefined.org/tools/AugPen13.tar.gz

# Disclaimer

- When you start:

# Disclaimer

- While Reversing:

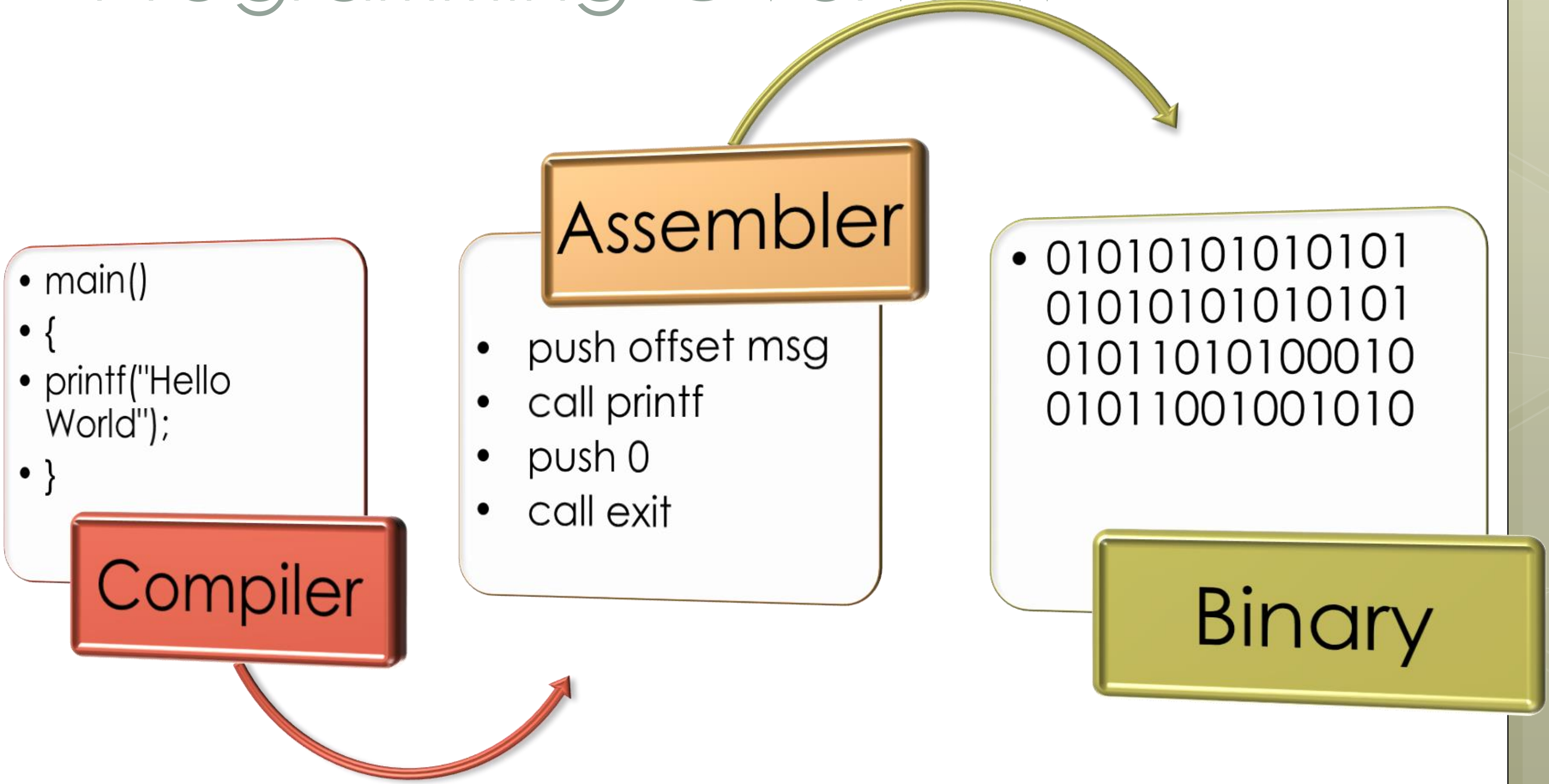
# Disclaimer

- After a small project:
- After a big one:

# What is RE?

- The process of reverse engineering
- Trying to unravel something which is unknown
- Can be implemented on:
  - Program
  - Component
  - Protocol
  - Hardware

# Programming Overview



- main()
- {
- printf("Hello World");
- }

Compiler

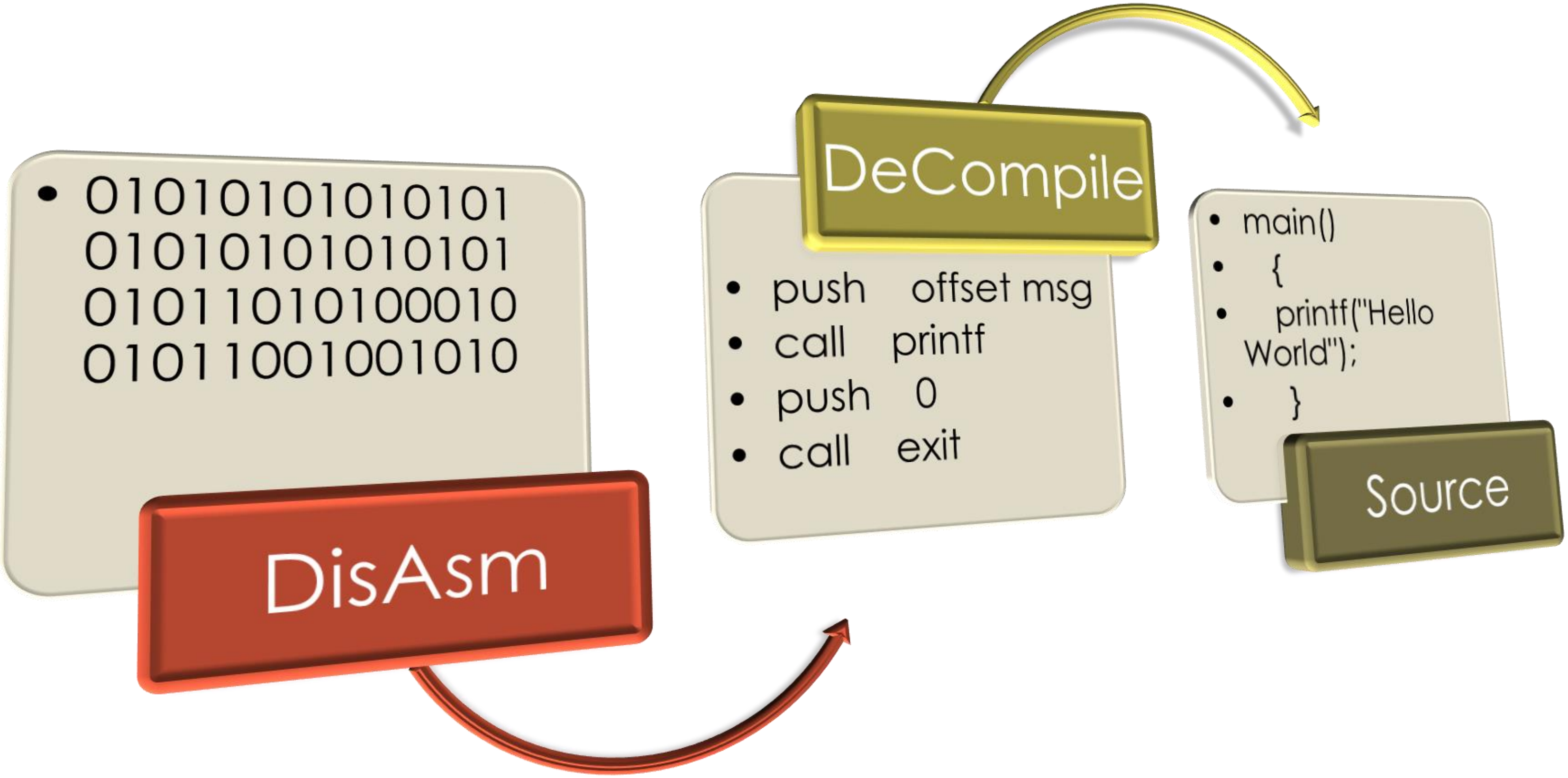
Assembler

- push offset msg
- call printf
- push 0
- call exit

- 01010101010101010101010101010101101010001001011001001010

Binary

# Reversing Overview



- 01010101010101 01010101010101 01011010100010 01011001001010

DisAsm

- push offset msg
- call printf
- push 0
- call exit

DeCompile

- main()
- {
- printf("Hello World");
- }

Source

# 1/1 Correlation?

| ASM | C |
|---|---|
| `inc  result` | `result++;` |
| `mov  class, 35` | `class =  35;` |
| `and  mask1, 128` | `mask1 &= 128;` |
| `add  marks, 100` | `mark1 += 100;` |

# Translate this

C Source

ASM

```
size = value;

sum += x + y + z;
```

# Translate this

## C Source

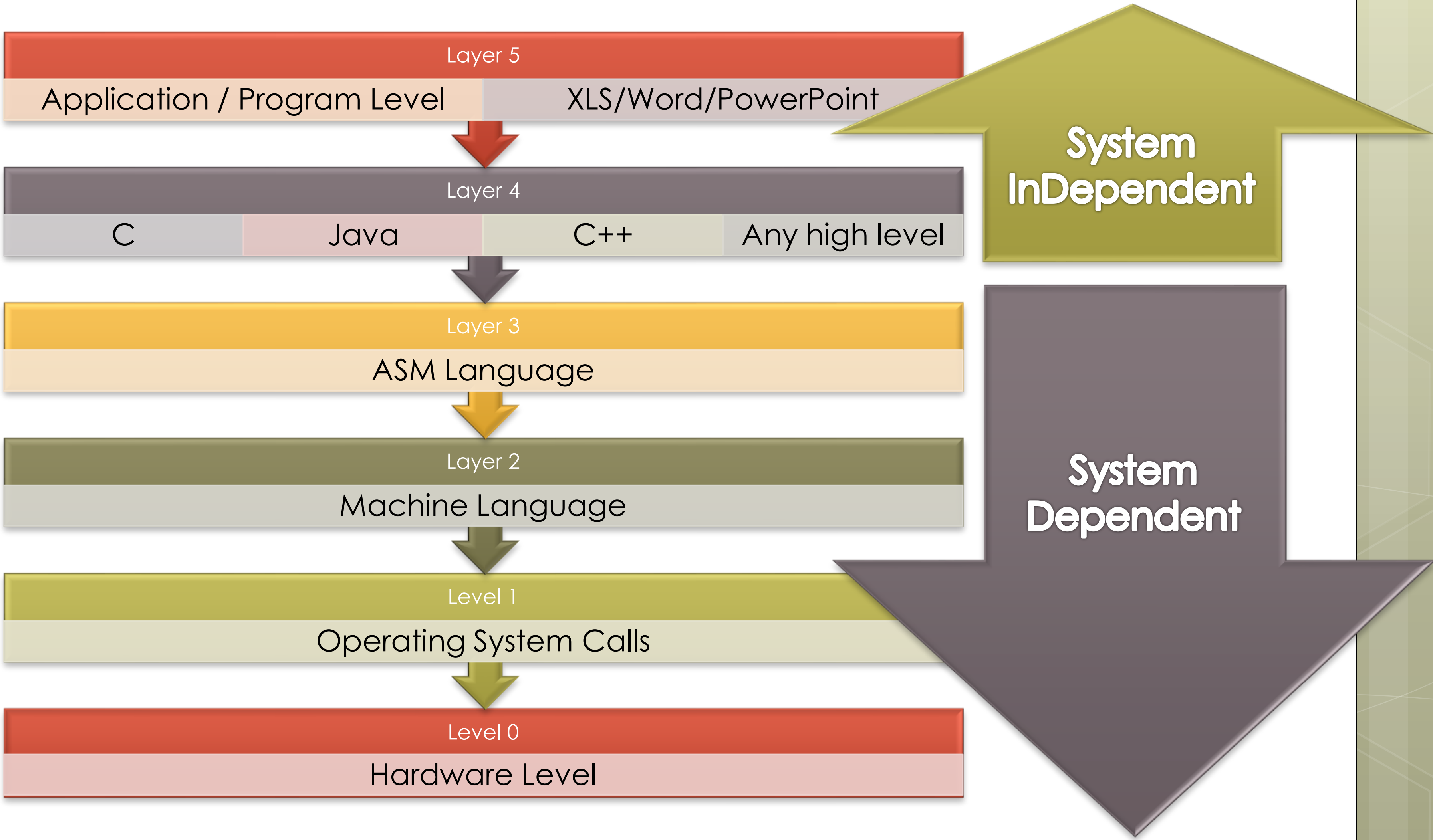
```
size = value;


sum += x + y + z;
```

## ASM

```
mov AX, value
mov size, AX

mov AX, sum
add AX, x
add AX, y
add AX, z
mov sum, AX
```



Layer 5

Application / Program Level | XLS/Word/PowerPoint

Layer 4

C | Java | C++ | Any high level

Layer 3

ASM Language

Layer 2

Machine Language

Level 1

Operating System Calls

Level 0

Hardware Level

System InDependent

System Dependent

# Registers

- EAX :
  - Accumulator : used for performing calculations, and used to store return values from function calls. Basic operations such as add, subtract, compare use this general-purpose register
- EBX :
  - Base : It has no general purpose and can be used to store data.
- ECX :
  - Counter : used for iterations. ECX counts downward.

# Registers

- EDX : data :
  - Extension of the EAX register.
  - It allows for more complex calculations (multiply, divide) by allowing extra data to be stored.
- ESP :
  - Stack pointer
- EBP :
  - Base pointer
- EIP :
  - Instruction pointer

# Memory

- When an application is stared a process is created and virtual memory is assigned to it.
- In a 32 bit process, the address ranges from 0×00000000 to 0xFFFFFFFF
- 0×00000000 -> 0x7FFFFFFF - "user-land"
- 0×80000000 -> 0xFFFFFFFF - "kernel land"

# PEBs & TEBs

- Kernel land memory only accessible by OS.
- When a process is created, a PEB (Process Execution Block) and TEB (Thread Environment Block) are created.

# Tools of the Trade

- Disassemblers
- Packers
- Crypters
- Debuggers
- Hex Editors
- Many many more

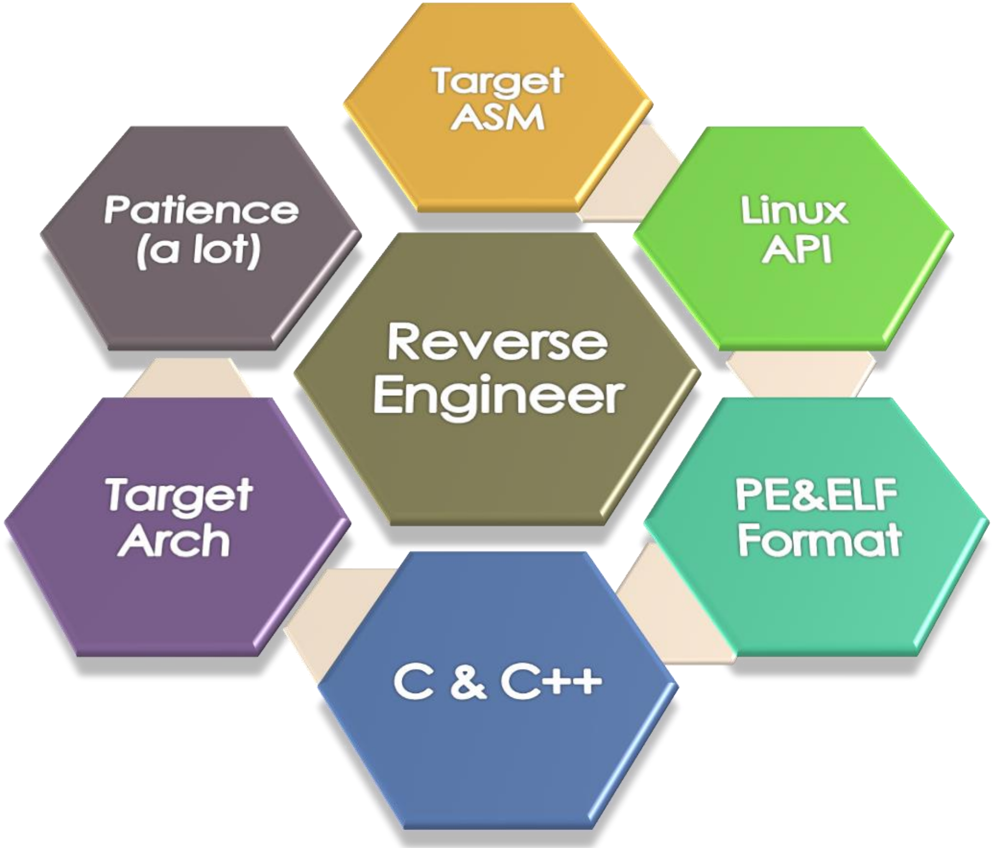
# Memory Segments

- Code Segment
  - Contains instruction sets for the CPU.
  - The EIP keeps track of the next instruction.
- Data Segment
  - Variables
  - Dynamic buffers
- Stack Segment
  - Passing Data & Arguments to functions
  - Space for Variables
  - Start (bottom of stack) from end of virtual memory and grows down (lower address)

# What we are going to do

- http://hackingdefined.org/tools/AugPen13.tar.gz
- Extract the file you’ve downloaded
- Install the ida_pro
- **`sudo apt-get install readelf elfdump objdump`**

# What’s Needed?

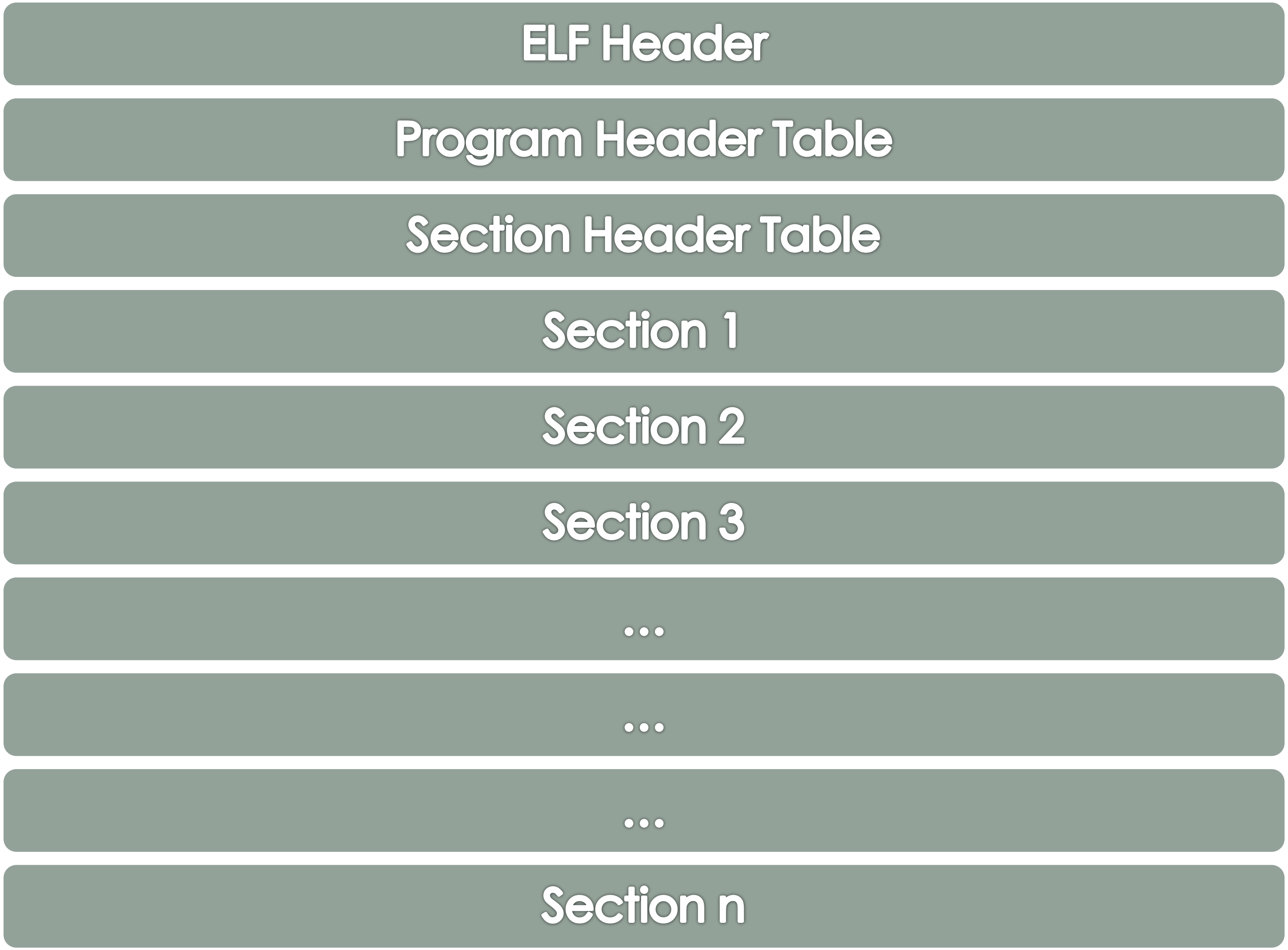


# Let's talk about ELFs

- They have pointy ears
- The live in Rivendell which looks like this:

- They look like this:

00:00:57
01:10



ELF Header

Program Header Table

Section Header Table

Section 1

Section 2

Section 3

…

…

…

Section n

# Read 'em!

- Let's use
- `readelf -e new`
- All your headers are belong to us

# Code Sections

- Let’s use
- `readelf -s new`
- All your functions are belong to us

# String Section

- Let’s have  a look at a not-properly-compiled code with:
- `readelf -p .strtab new`

# Quick Little Thing

- Frames are important
- When calling a function the following instructions are called:

```
push    %ebp         //save *frame
mov     %esp, %ebp   //set new *frame
sub     $80, %esp    //allocate 80
push    %esi         //save
push    %edi         //save
push    %ebx         //save
```

# Quick Little Thing

- Exiting a function:

```
mov     %edi, %eax          // Return
pop     %ebx                // Restore
pop     %esi                // Restore
pop     %edi                // Restore
leave
ret
```

FOR TEH LULZ

# More Stuff

- HackingDefined.org – Codes, Articles, Guides
- Suggestions/Comments box
- Intro to Reversing on Corelan - Nice

Friday, August 2, 2013

# Thank you

ELFs and other cool creatures