Malware Unpacking Workshop

Lilly Chalupowski
August 28, 2019

whois lilly.chalupowski

Table: *who.is results*

Name	Lilly Chalupowski
Status	Employed
Creation Date	1986
Expiry	A Long Time from Now (Hopefully)
Registrant Name	GoSecure
Administrative Contact	Travis Barlow
Job	TITAN Malware Research Lead

Agenda
What will we cover?

- Disclaimer
- Reverse Engineering
  - Registers
  - Stack
  - Heap
  - Assembly
  - Calling Conventions
- Tools
  - x64dbg
  - Cutter
  - Radare2
  - Detect it Easy
  - HxD
- Injection Techniques
  - DLL Injection
  - PE Injection
  - Process Hollowing
  - Atom Bombing
- Workshop

Disclaimer
Don't be a Criminal

disclaimer.log

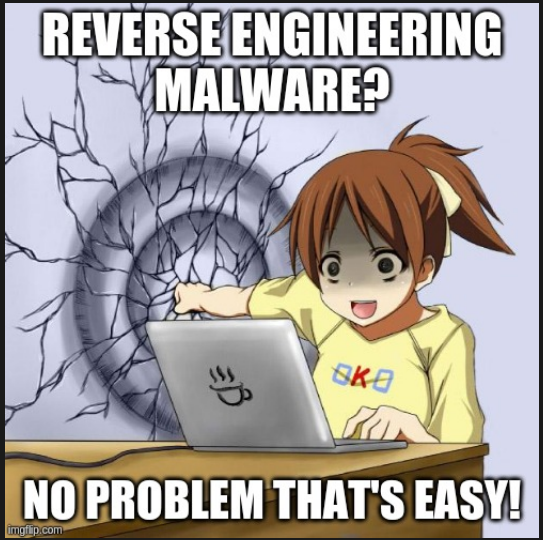
The tools and techniques covered in this presentation can be dangerous and are being shown for educational purposes.

It is a violation of Federal laws to attempt gaining unauthorized access to information, assets or systems belonging to others, or to exceed authorization on systems for which you have not been granted.

Only use these tools with/on systems you own or have written permission from the owner. I (the speaker) do not assume any responsibility and shall not be held liable for any illegal use of these tools.

Reverse Engineering

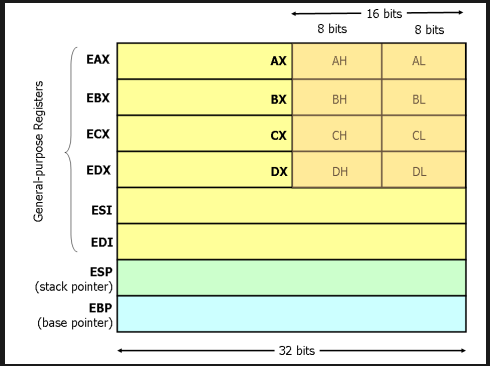
It's easy don't worry!



Registers
Not this one!

Registers
Not the kind with money in them

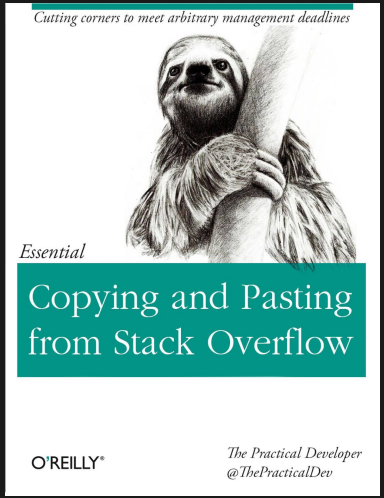
- EAX - Return Value of Functions
- EBX - Base Index (for use with arrays)
- ECX - Counter in Loops
- EDI - Destination Memory Operations
- ESI - Source Memory Operations
- ESP - Stack Pointer
- EBP - Base Frame Pointer

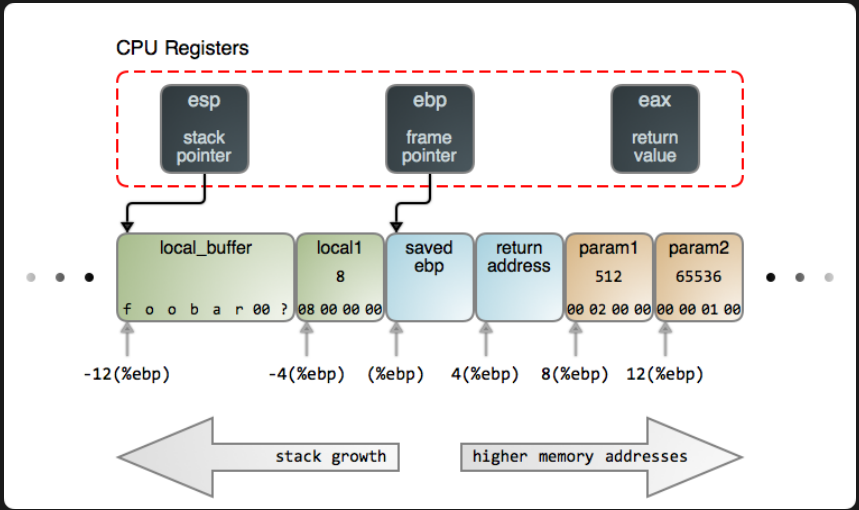


Did You Know: In computer architecture, a processor register is a quickly accessible location available to a computer's central processing unit (CPU).

The Stack

- Last-In First-Out
- Downward Growth
- Function Local Variables
- ESP
- Increment / Decrement = 4
  - Double-Word Aligned




CPU Registers
esp
stack pointer
ebp
frame pointer
eax
return value
local_buffer
f o o b a r 00 ?
local1
8
08 00 00 00
saved ebp
return address
param1
512
00 02 00 00
param2
65536
00 00 01 00
-12(%ebp)
-4(%ebp)
(%ebp)
4(%ebp)
8(%ebp)
12(%ebp)
stack growth
higher memory addresses

Control Flow
Keeping it under control

- Conditionals
  - CMP
  - TEST
  - JMP
  - JCC
- EFLAGS
  - ZF / Zero Flag
  - SF / Sign Flag
  - CF / Cary Flag
  - OF/Overflow Flag

Calling Conventions
Subtitle goes here

- CDECL
  - Arguments Right-to-Left
  - Return Values in EAX
  - Calling Function Cleans the Stack
- STDCALL
  - Used in Windows Win32API
  - Arguments Right-to-Left
  - Return Values in EAX
  - The called function cleans the stack, unlike CDECL
  - Does not support variable arguments
- FASTCALL
  - Uses registers as arguments
  - Useful for shellcode

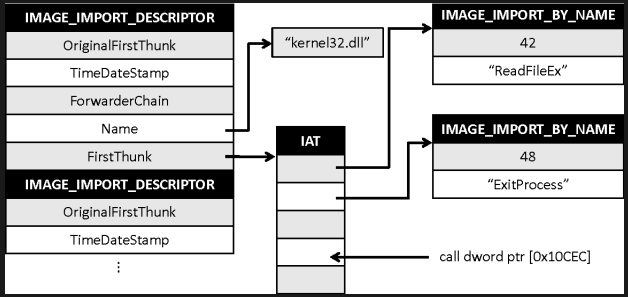
Windows Memory Structure
subtitle

- Stack - Grows up to lower addresses
- Heap - Grows down to higher addresses
- Program Image
- TEB - Thread Environment Block
  - GetLastError()
  - GetVersion()
  - Pointer to the PEB
- PEB - Process Environment Block
  - Image Name
  - Global Context
  - Startup Parameters
  - Image Base Address
  - IAT (Import Address Table)

IAT (Import Address Table) and IDT (Import Lookup Table)

subtitle

- Identical to the IDT (Import Directory Table)
- Binding - The process of where functions are mapped to their virtual addresses overwriting the IAT
- Often the IDT and IAT must be rebuilt when packing and unpacking malware



- Common Instructions
  - MOV
  - XOR
  - PUSH
  - POP

Assembly CDECL (Linux)

subtitle

cdecl.c

```
__cdecl int add_cdecl(int a, int b){
  return a + b;
}
int x = add_cdecl(2, 3);
```

Assembly CDECL (Linux)

subtitle

cdecl.asm

```
_add_cdecl:
  push ebp
  mov ebp, esp
  mov eax, [ebp + 8]  ; get 3 from the stack
  mov edx, [ebp + 12] ; get 2 from the stack
  add eax, edx        ; add values to eax
  pop ebp
  ret
_start:
  push 3              ; second argument
  push 2              ; first argument
  call _add_cdecl
  add esp, 8
```

Assembly STDCALL (Windows)

subtitle

stdcall.c

```
__stdcall int add_stdcall(int a, int b){
  return a + b;
}
int x = add_stdcall(2, 3);
```

Assembly STDCALL (Windows)

subtitle

stdcall.asm

```
_add_stdcall:
  push ebp
  mov ebp, esp
  mov eax, [ebp + 8]  ; set eax to 3
  mov edx, [ebp + 12] ; set edx to 2
  add eax, edx
  pop ebp
  ret 8               ; how many bytes to pop
_start:               ; main function
  push 3              ; second argument
  push 2              ; first argument
  call _add_stdcall
```

Assembly FASTCALL

subtitle

cdecl.c

```
__fastcall int add_fastcall(int a, int b){
  return a + b;
}
int x = add_fastcall(2, 3);
```

Assembly FASTCALL

subtitle

fastcall.asm

```
_add_fastcall:
  push ebp
  mov ebp, esp
  add eax, edx          ; add and save result in eax
  pop ebp
  ret
_start:
  mov eax, 2            ; first argument
  mov edx, 3            ; second argument
  call _add_fastcall
```

Guess the Calling Convention

Hello World Intel Syntax

hello.asm

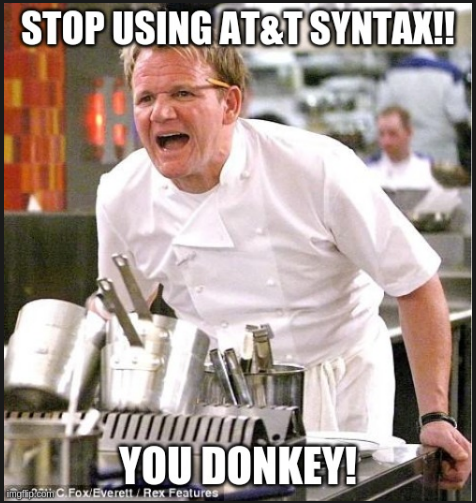
```
section     .text                   ; the code section
global      _start                  ; tell linker entrypoint
_start:
  mov     edx,len                   ; message length
  mov     ecx,msg                   ; message to write
  mov     ebx,1                     ; file descriptor stdout
  mov     eax,4                     ; syscall number for write
  int     0x80                      ; linux x86 interrupt
  mov     eax,1                     ; syscall number for exit
  int     0x80                      ; linux x86 interrupt
section     .data                   ; the data section
  msg     db  'Hello, world!',0x0   ; null terminated string
  len     equ \$ - msg              ; message length
```

terminal

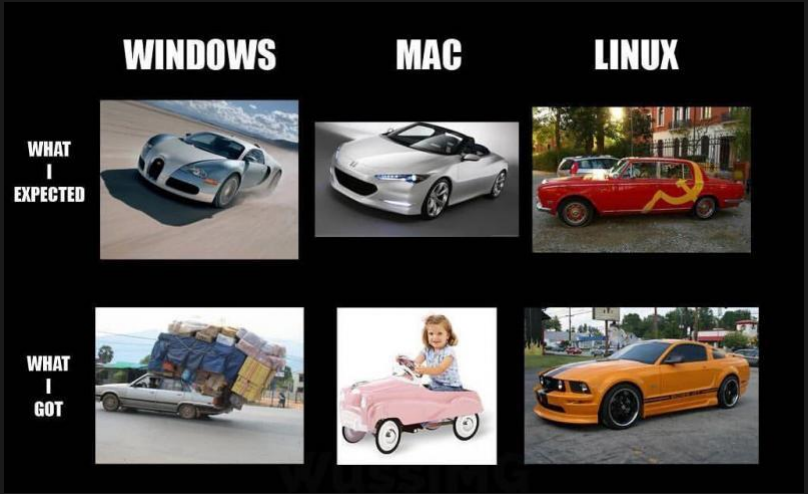
```
malware@work ~$ nasm -f elf32 -o hello.o hello.asm
malware@work ~$ ld -m elf_i386 -o hello hello.o
malware@work ~$ ./hello
Hello, World!
malware@work ~$
```

Assembly Flavors

I know you were thinking it!



Tools of the Trade



x64dbg

subtitle