

Malware Unpacking Workshop



Lilly Chalupowski
August 28, 2019

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

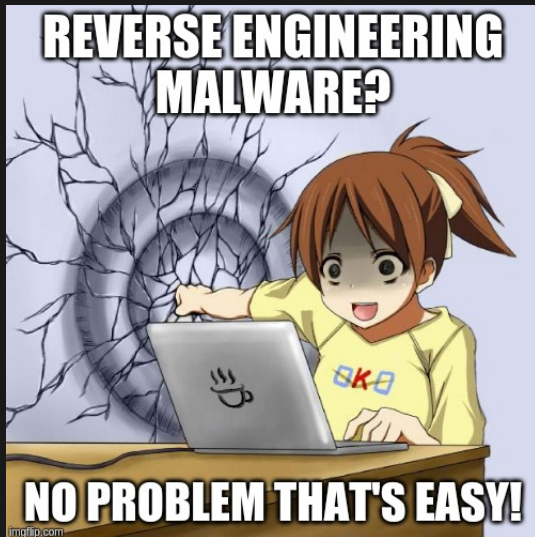
The tools and techniques covered in this presentation can be dangerous and are being shown for educational purposes.

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

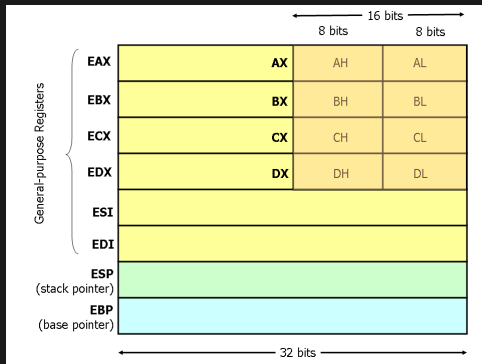
Not this one!



Registers

Not the kind with money in them

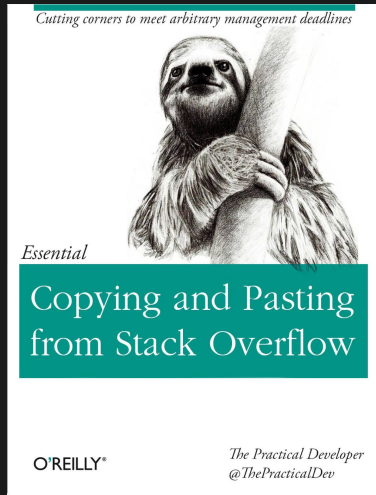
- EAX - Return Value of Functions
- EBX - Base Index (for use with arrays)
- ECX - Counter in Loops
- EDI - Destination Memory Operations
- ESI - Source Memory Operations
- ESP - Stack Pointer
- EBP - Base Frame Pointer



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

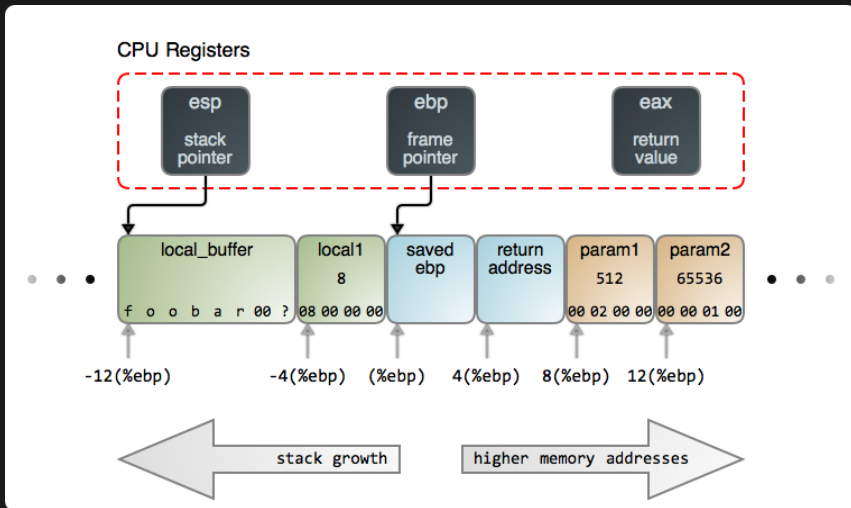
The Stack

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



Stack

The stack



Control Flow

Keeping it under control

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



Calling Conventions

Subtitle goes here

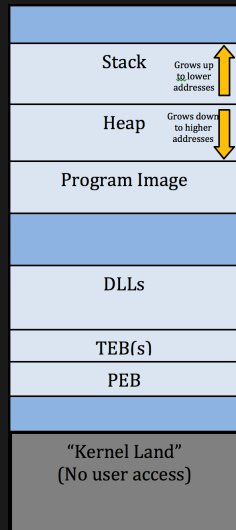
- CDECL
 - Arguments Right-to-Left
 - Return Values in EAX
 - Calling Function Cleans the Stack
- STDCALL
 - Used in Windows Win32API
 - Arguments Right-to-Left
 - Return Values in EAX
 - The called function cleans the stack, unlike CDECL
 - Does not support variable arguments
- FASTCALL
 - Uses registers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



Windows Memory Structure

subtitle

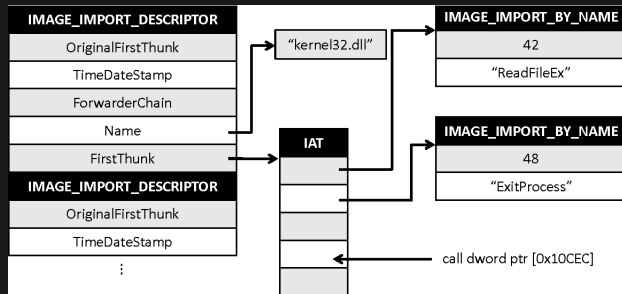
- Stack - Grows up to lower addresses
- Heap - Grows down to higher addresses
- Program Image
- TEB - Thread Environment Block
 - GetLastError()
 - GetVersion()
 - Pointer to the PEB
- PEB - Process Environment Block
 - Image Name
 - Global Context
 - Startup Parameters
 - Image Base Address
 - IAT (Import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



Assembly

Instructions

- 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
```

Assembler and Linking

subtitle

terminal

```
malware@work ~$ nasm -f elf32 -o hello.o hello.asm
```

```
malware@work ~$ ld -m elf_i386 -o hello hello.o
```

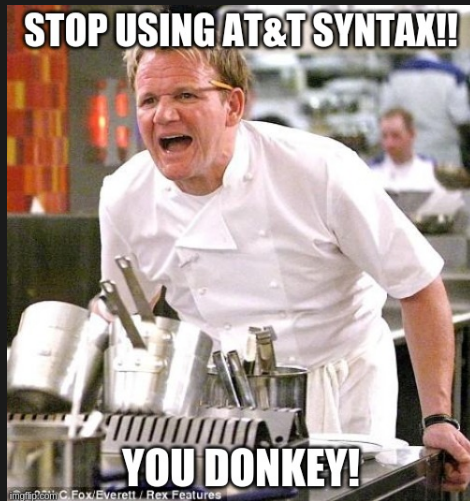
```
malware@work ~$ ./hello
```

Hello, World!

```
malware@work ~$
```

Assembly Flavors

I know you were thinking it!



Tools of the Trade

WINDOWS

**WHAT
I
EXPECTED**



MAC



LINUX



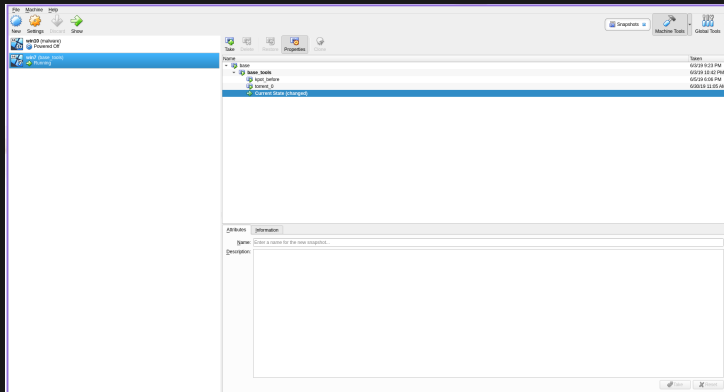
**WHAT
I
GOT**



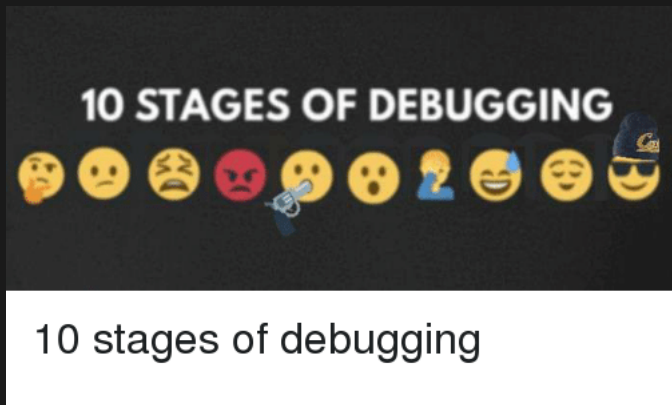
VirtualBox

Purpose

- Snapshots
- Security Layer
- Multiple Systems



- Resolving APIs
- Dumping Memory
- Modify Control Flow
- Identify Key Behaviors



The screenshot displays the x64dbg application interface with several key components highlighted by red boxes and labels:

- Disassembly:** The central pane shows the assembly code of the `sofacy-packed.1211C63` function. It includes instructions like `push rdi`, `movzx esi, eax`, `call sofacy-packed.1212A60`, and `jmp sofacy-packed.121301E`.
- Registers:** The right-hand pane displays the current state of CPU registers, including `EAX`, `ECX`, `EDX`, `ESP`, `EBP`, `ESI`, `EDI`, and `EIP`.
- EFLAGS:** The right-hand pane shows the status of the EFLAGS register, including bits like `OF`, `DF`, `IF`, `OF`, `DF`, `IF`, `OF`, `DF`, `IF`, `OF`, `DF`, `IF`.
- Heap:** The bottom-left pane shows the memory layout of the heap, with addresses ranging from `774C0000` to `774C0000`.
- Stack:** The bottom-right pane shows the memory layout of the stack, with addresses ranging from `774C0000` to `774C0000`.

At the bottom of the interface, the command line shows the current process and command:

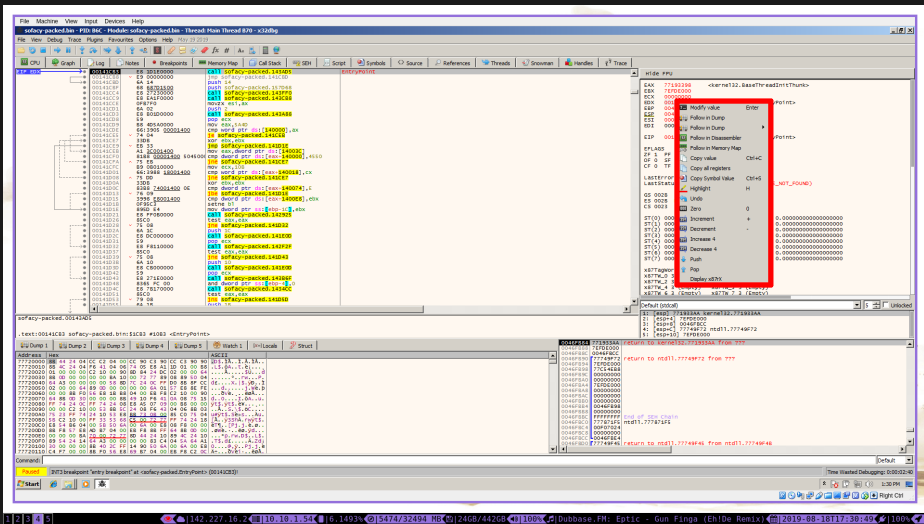
```
3rb3rus at 03d53c in /home/c3rb3rus
^A scrot
```

The screenshot displays the x64dbg debugger interface with the following components:

- File Menu:** Includes options like File, Machine, View, Input, Devices, Help.
- Run Menu:** A red box highlights the 'Run' menu, which contains:
 - Run (F5)
 - Run until selection (F4)
 - Pause (F12)
 - Restart (Ctrl+F2)
 - Close (Alt+F2)
 - Step into (F7)
 - Step over (F8)
 - Execute till return (Ctrl+F9)
 - Run to user code (Alt+F9)
 - Advanced
- CPU:** Shows the current CPU state, including registers like EAX, ECX, EDI, etc.
- Disassembly:** Displays the assembly code being executed, with instructions like 'CALL', 'PUSH', 'MOV', 'CMP', etc.
- Memory Dump:** Shows a dump of memory at address 00141C83, with a search for 'entryPoint'.
- Registers:** A list of registers and their values, including EAX, ECX, EDI, etc.
- Stack:** A view of the stack memory, showing addresses and data.
- Command Line:** A field for entering commands, currently showing '3072 breakpoint "entry breakpoint" at <sofacy-packed.entryPoint> (00141C83)'.

x64dbg

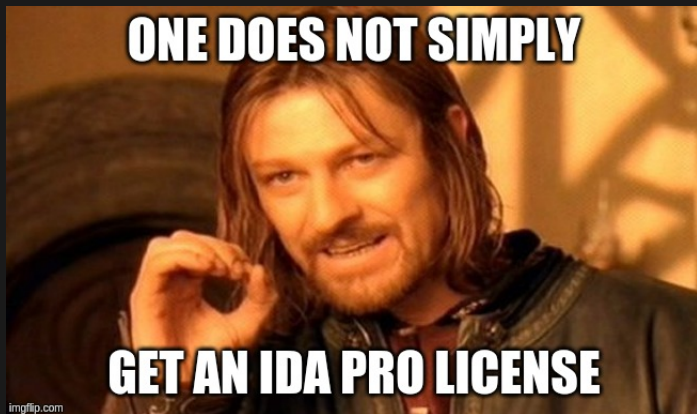
Context Menus



Cutter

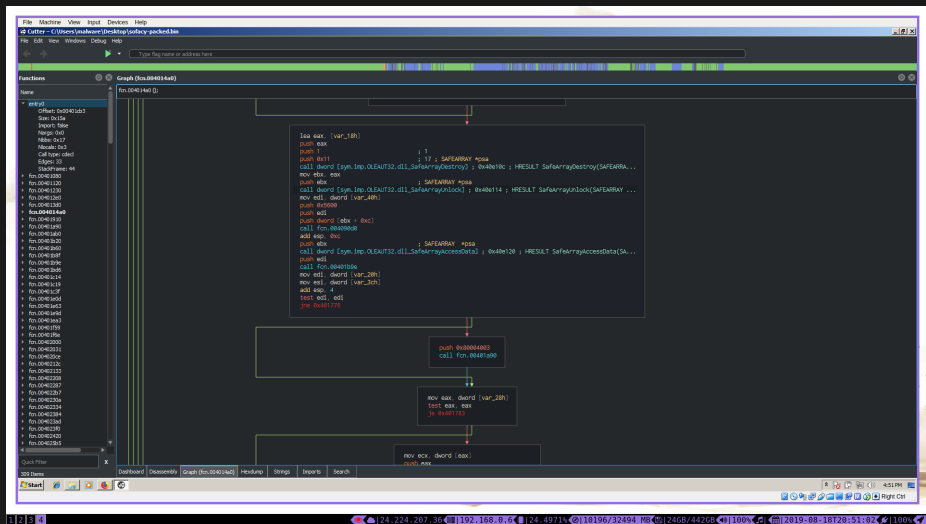
Purpose

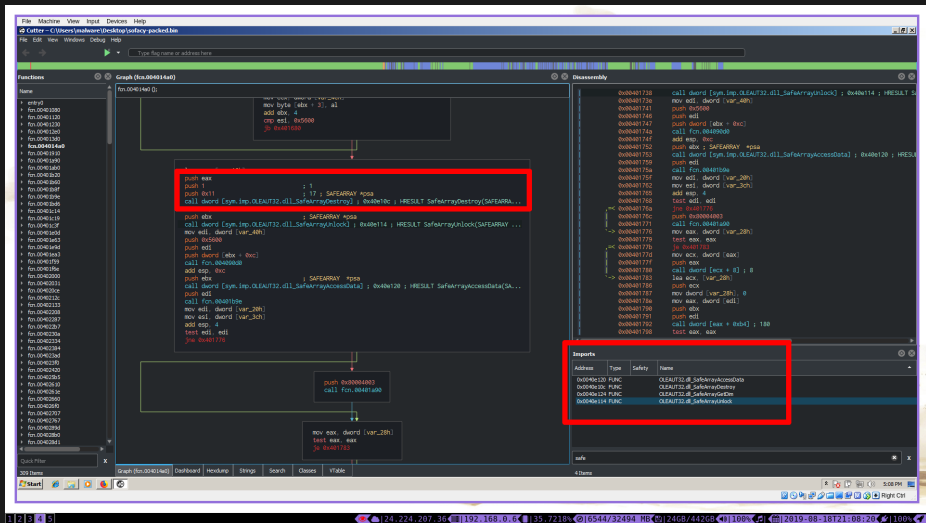
- Markup Reverse Engineered Code
- Control Flow Navigation
- Pseudo Code



Cutter

Graph View





Radare2

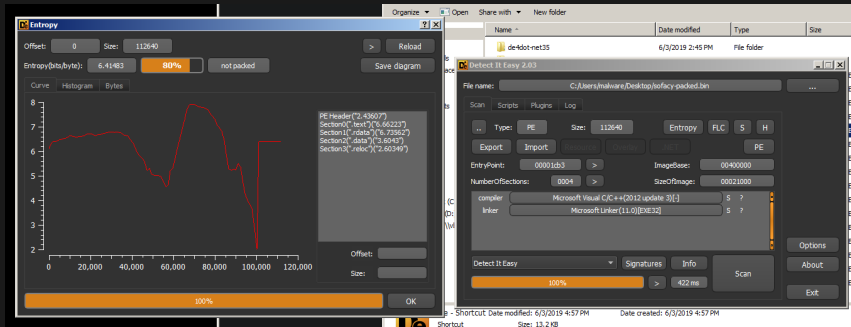
Backend of Cutter

```
[0x0003960]> !screenfetch
-./oyddmdhs+:+
-odnMMMMMMMMMMNmhys+/-
-yNMMMMMMMMMMNNmndhy+-
-oeMMMMMMMMMMNNadnnnddhhyy/-
omMMMMMMMMMMNNhyyochndddhhd+
-ydMMMMMMMMMMNdhss+so/-mddhhhdms+
cyhdnNMMMMMMNNyooymddddhhhhhyhWd
:cyhhdNMMMMMMNNmddddhhhhhyyeh
..+sydNMMMMMMNNmddddhhhhhhmMay
/+MMMMMMNNmddddhhhhmMhs+
-cNMMMMMMNNmdddddhdmMhs+
-NMMMMMMNNmdddddmMhs+/-
/WMWMMMMMMNNmdddmMhds+
+MMMMMMNNmdddmMhds+/-
yMMMMMMNNmdddmMhs+/-
/MMMMMMNNmdddmMhs+/-
/ohdmddhs+++/-
-./!!!!/:-
[0x0003960]> pd 16
;-- section..text:
(fcn) main 678
int main (int argc, char **argv, char **envp);
bp: 0 (vars 0, args 0)
sp: 9 (vars 9, args 0)
rg: 2 (vars 0, args 2)
; DATA XREF from entry0 (0x530d)
0x0003960 4157      push r15
0x0003962 4156      push r14
0x0003964 4155      push r13
0x0003966 4154      push r12
0x0003968 55        push rbp
0x0003969 53        push rbx
0x000396a 89fd      mov ebp, edi
0x000396c 4889f3    mov rbx, rsi
0x000396f 4883ec48  sub rsp, 0x48
0x0003973 488b3e    mov rdi, qword [rsi]
0x0003976 e875dc0800 call fcn.000115f0
0x000397b 488d35603301 lea rsi, [0x00016ce2]
0x0003982 bf06000000 mov edi, 6
0x0003987 e824feffff call sym.imp.setlocale
0x000398c 488d35003401 lea rsi, str_usr_share_locale
0x0003993 488d3d6f3401 lea rdi, [0x00016e09]
; [13] -r-x section size 72107 named .text
; argc
; argv
; "H"
; argv
; const char *locale
; int category
; char *setlocale(int category, const char *locale)
; 0x16e23; "/usr/share/locale"; char *dirname
; "coreutils"; char *domainname
[0x0003960]> px 32
; offset - 01 2 3 4 5 6 7 8 9 A B C D E F 0123456789ABCDEF
0x0003960 4157 4156 4155 4154 5553 89fd 4889 f348 AMAVAUATUS..H..H
0x0003970 83ec 4848 8b3e e875 dc08 0048 8d35 6033 ..HH.>.u..H.5`3
[0x0003960]> !scrt
```

Detect it Easy

Purpose

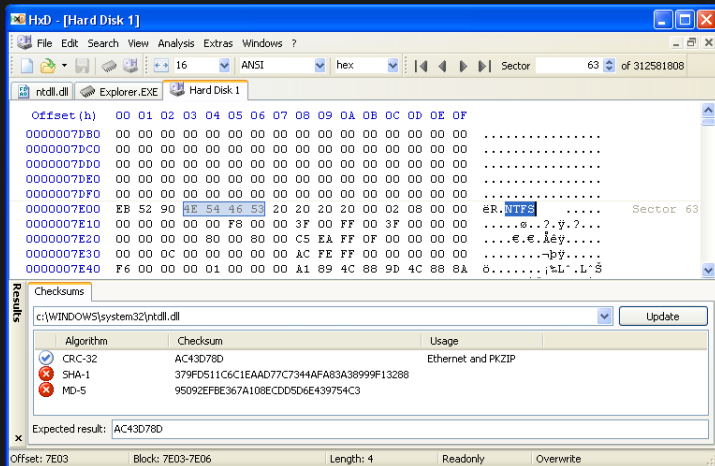
- Type
- Packer
- Linker
- Entropy



HxD

Purpose

- Modify Dumps
- Read Memory
- Determine File Type



Useful Linux Commads

subtitle

terminal

```
malware@work ~$ file sample.bin
```

```
sample.bin: PE32 executable (GUI) Intel 80386, for MS Windows
```

```
malware@work ~$ exiftool sample.bin > metadata.log
```

```
malware@work ~$ hexdump -C -n 128 sample.bin | less
```

```
malware@work ~$ VBoxManage list vms
```

```
"win10" {53014b4f-4c94-49b0-9036-818b84a192c9}
```

```
"win7" {942cde2e-6a84-4edc-b98a-d7326b4662ee}
```

```
malware@work ~$ VBoxManage startvm win7
```

```
malware@work ~$
```