

A black and white illustration of a character wearing a fedora hat and a mask, holding a briefcase. The character has red glowing eyes and a serious expression. The briefcase is open, revealing a dark interior.

**Finders Keepers (KPot Stealers)**

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

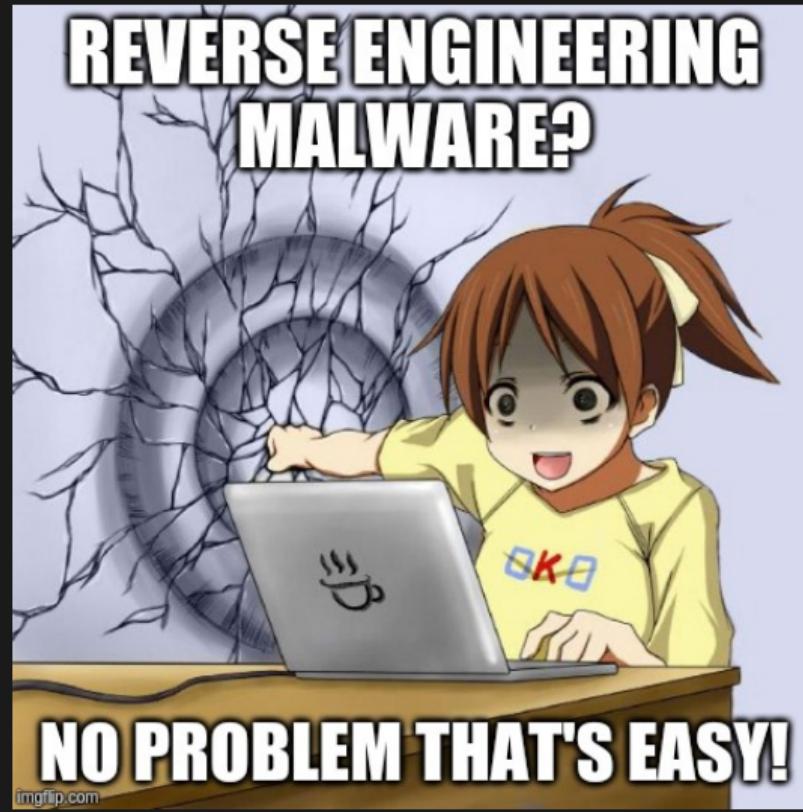
I (the speaker) do not assume any responsibility and shall not be held liable for anyone who infects their machine with the malware supplied for this workshop.

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

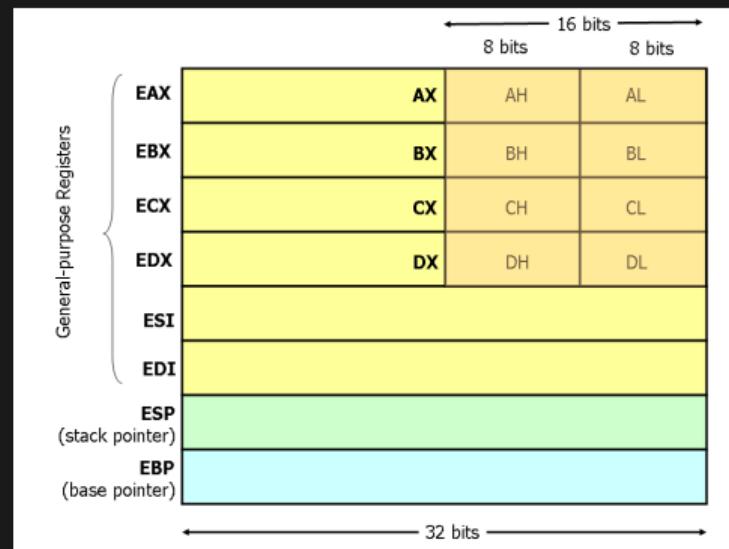
We choose to reverse engineer! We choose to reverse engineer... We choose to reverse engineer and do the other things, not because they are easy, but because they are hard; because that goal will serve to organize and measure the best of our energies and skills, because that challenge is one that we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



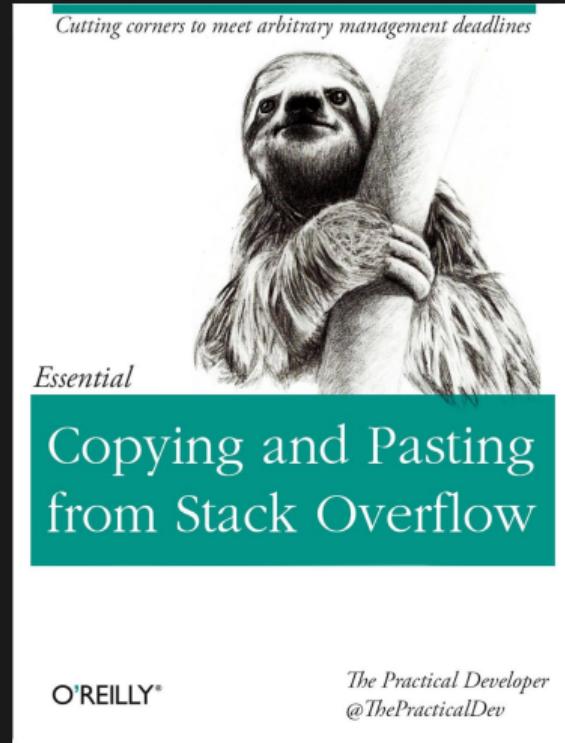
Did You Know: In computer architecture, a processor register is a quickly accessible location available to a computer's central processing unit (CPU).

# Registers

reverse\_engineering: 0x01

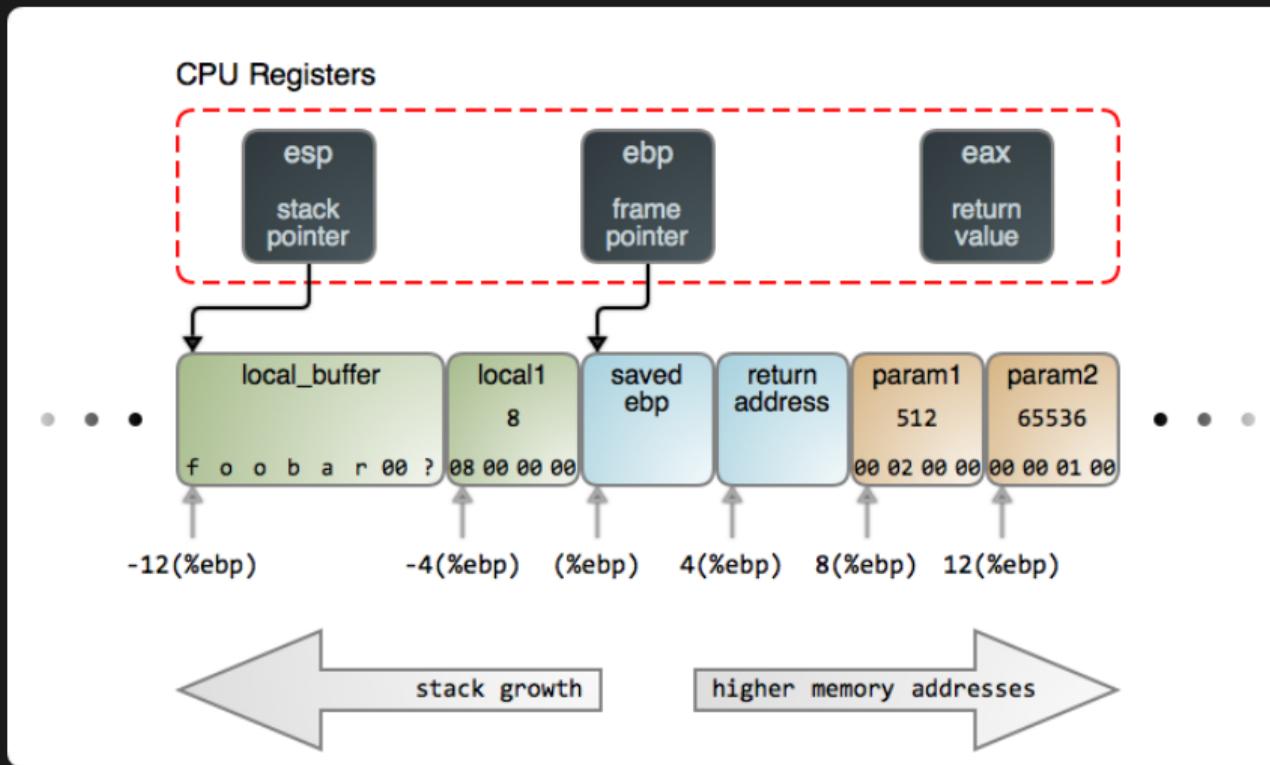


- Last-In First-Out
- Downward Growth
- Function Local Variables
- ESP
- Increment / Decrement = 4
  - Double-Word Aligned



# 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



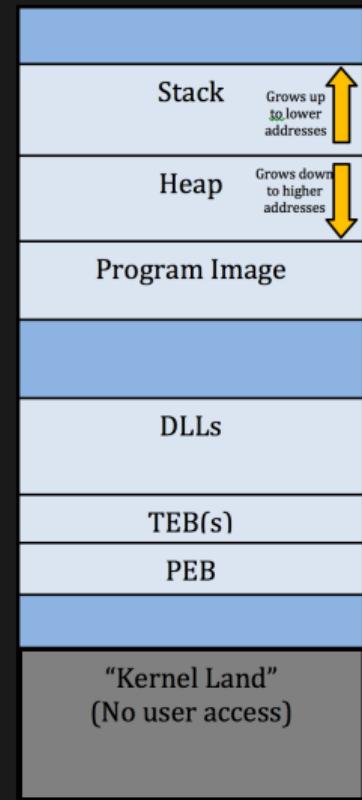
# Calling Conventions

reverse\_engineering: 0x05

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

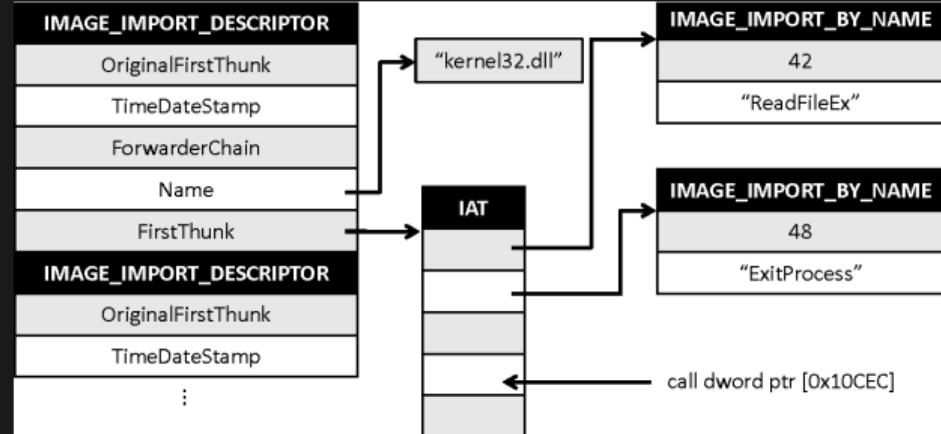


- Stack - Grows up to lower addresses
- Heap - Grows down to higher addresses
- Program Image
- TEB - Thread Environment Block
  - GetLastError()
  - GetVersion()
  - Pointer to the PEB
- PEB - Process Environment Block
  - Image Name
  - Global Context
  - Startup Parameters
  - Image Base Address
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

reverse\_engineering: 0x0f

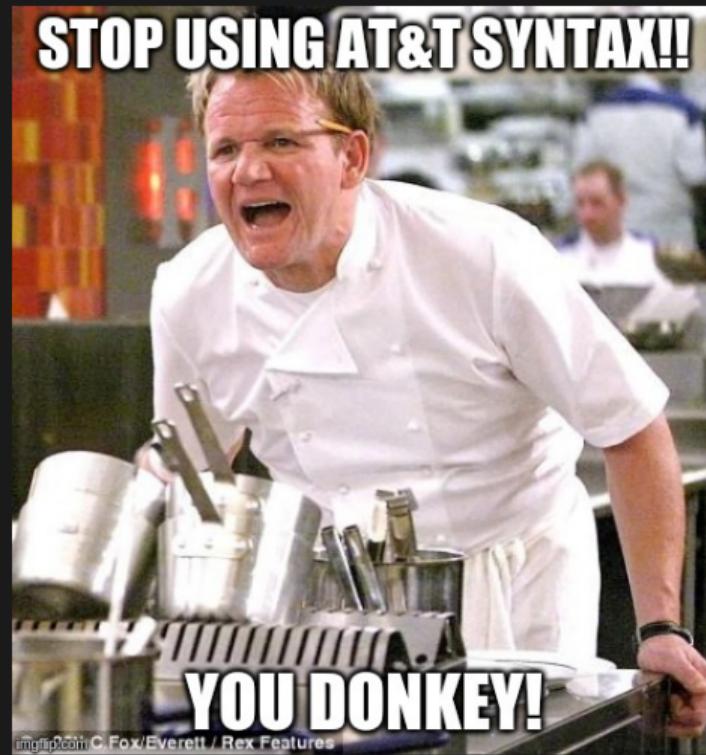
## hello.asm

```
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 ~$
```

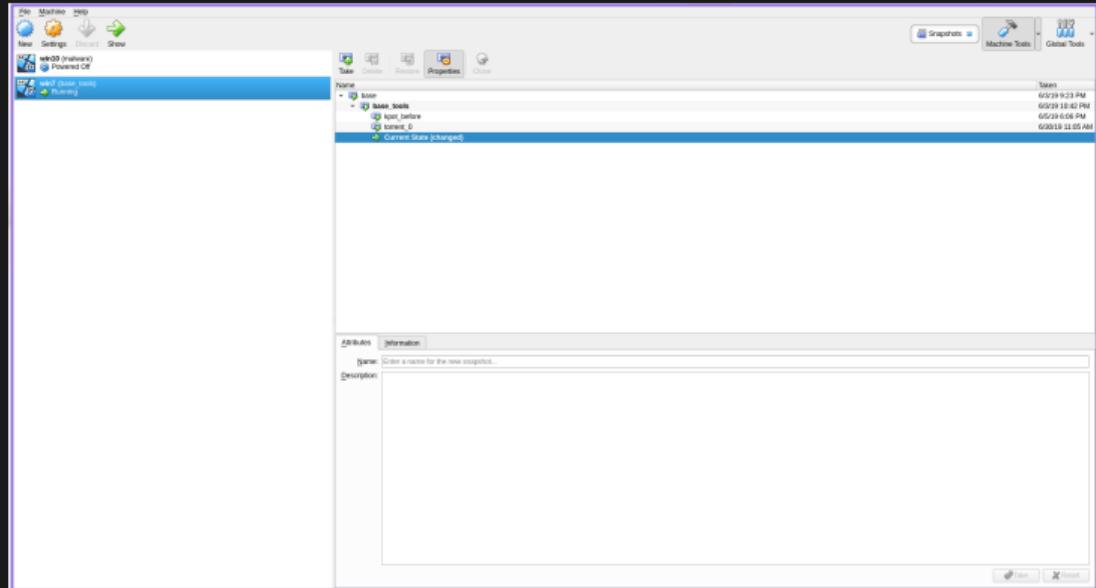


imgtip.com C. Fox/Everett / Rex Features



tools\_of\_the\_trade: 0x00

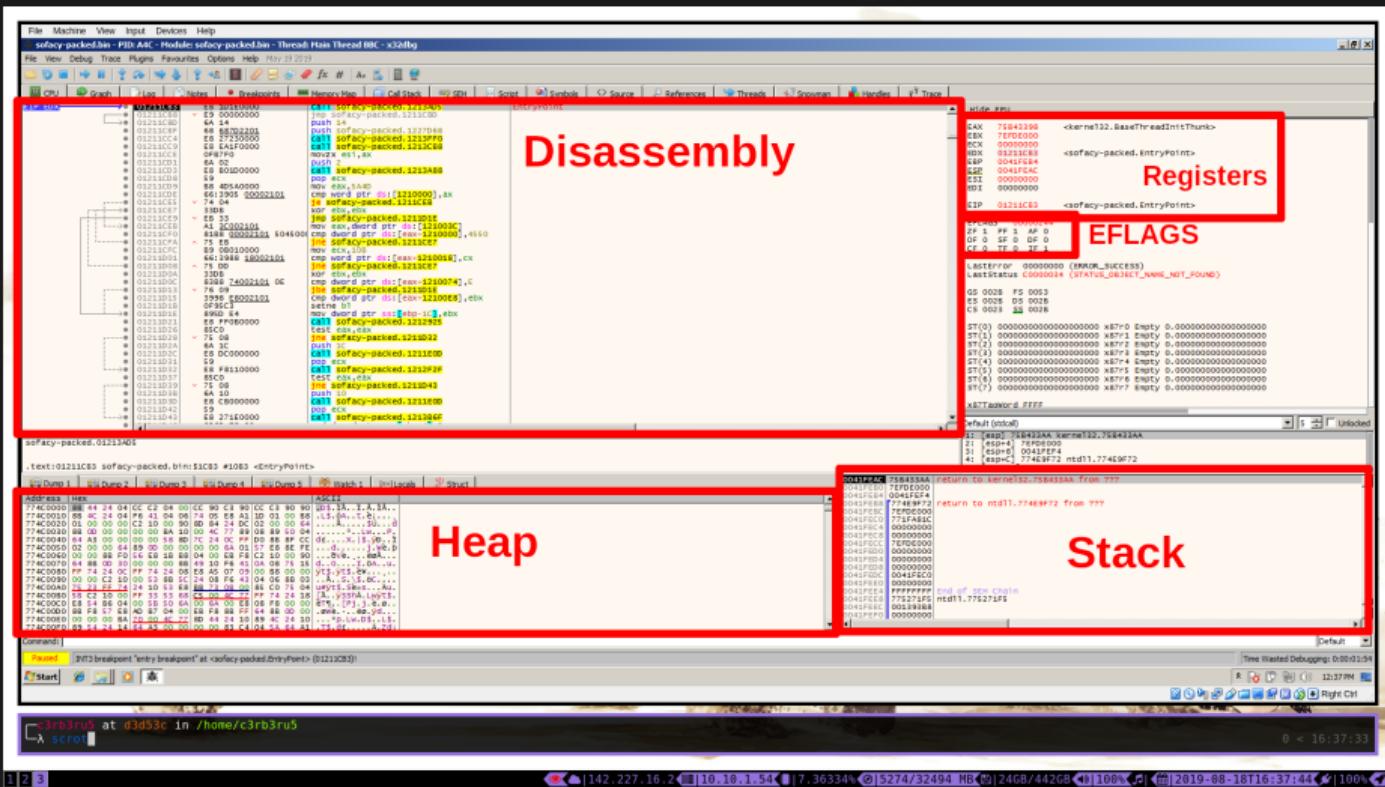
- Snapshots
- Security Layer
- Multiple Systems



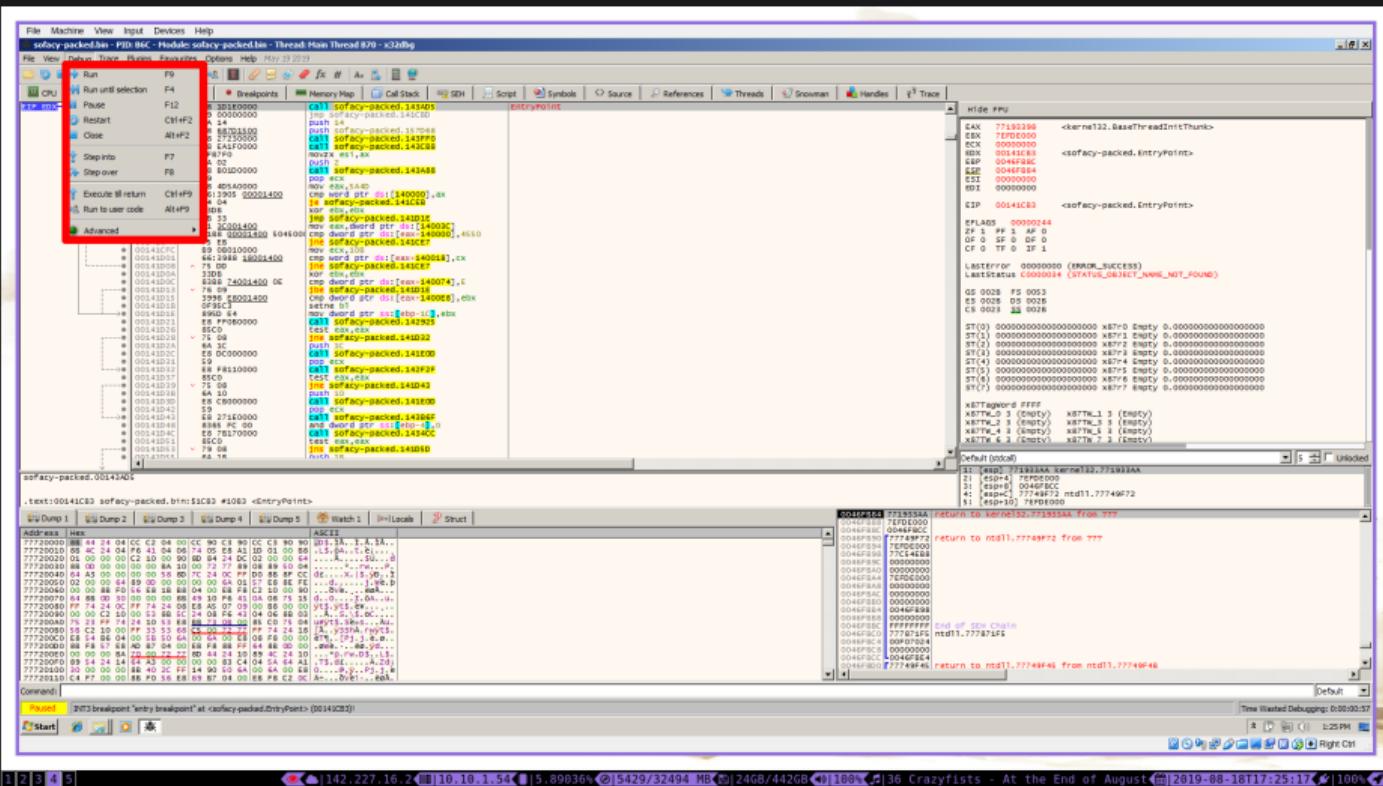
- Resolving APIs
- Dumping Memory
- Modify Control Flow
- Identify Key Behaviors



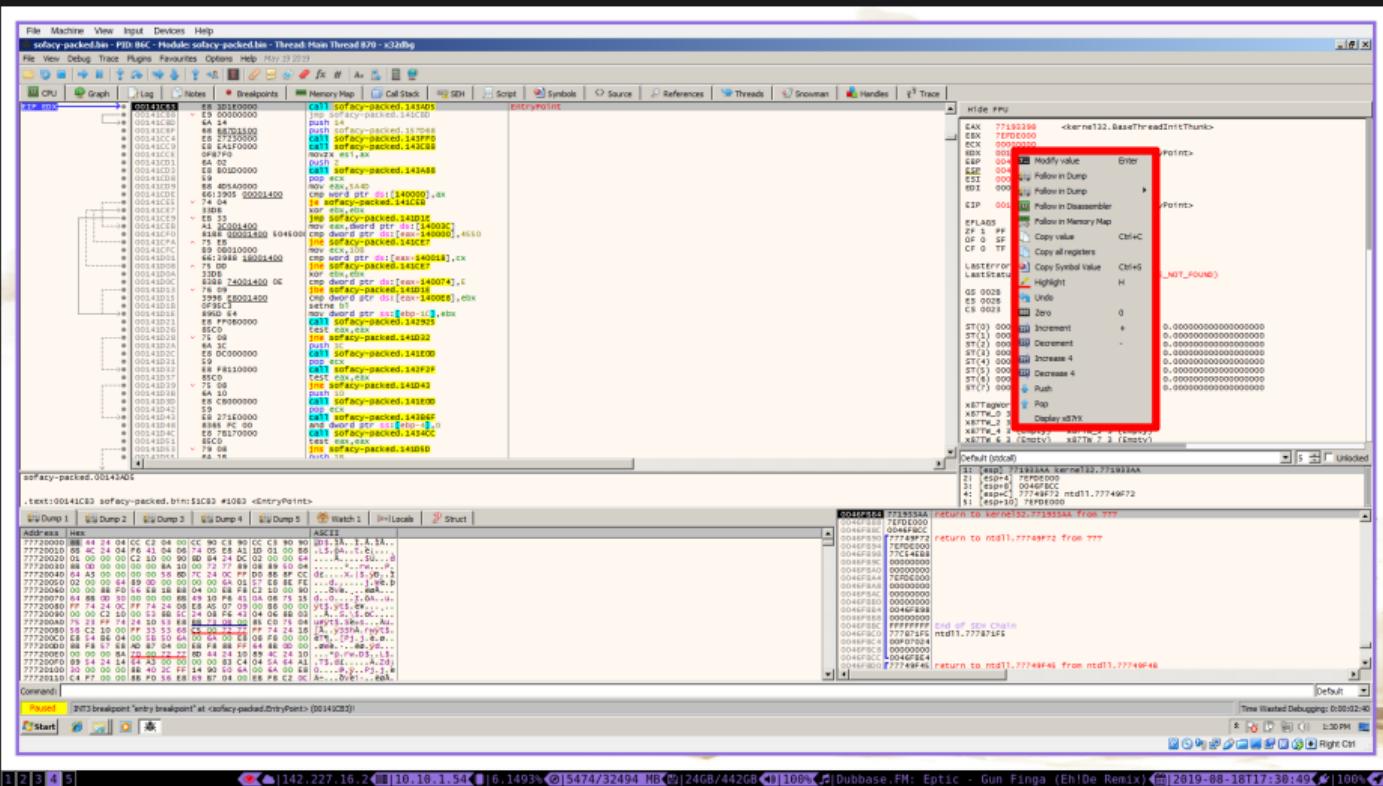
tools\_of\_the\_trade: 0x02



tools\_of\_the\_trade: 0x03

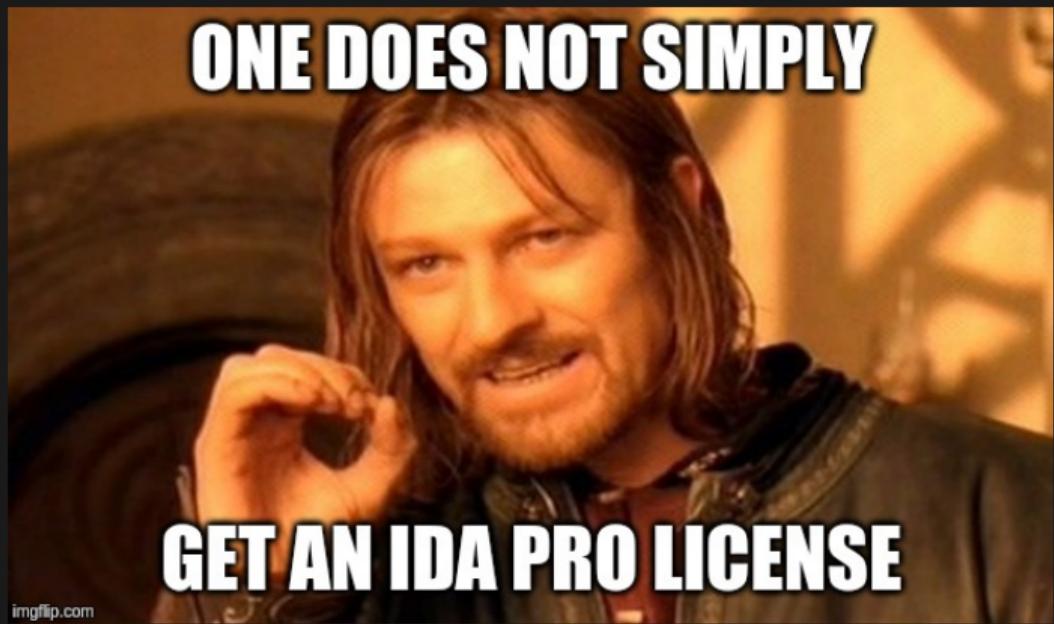


tools\_of\_the\_trade: 0x04



tools\_of\_the\_trade: 0x05

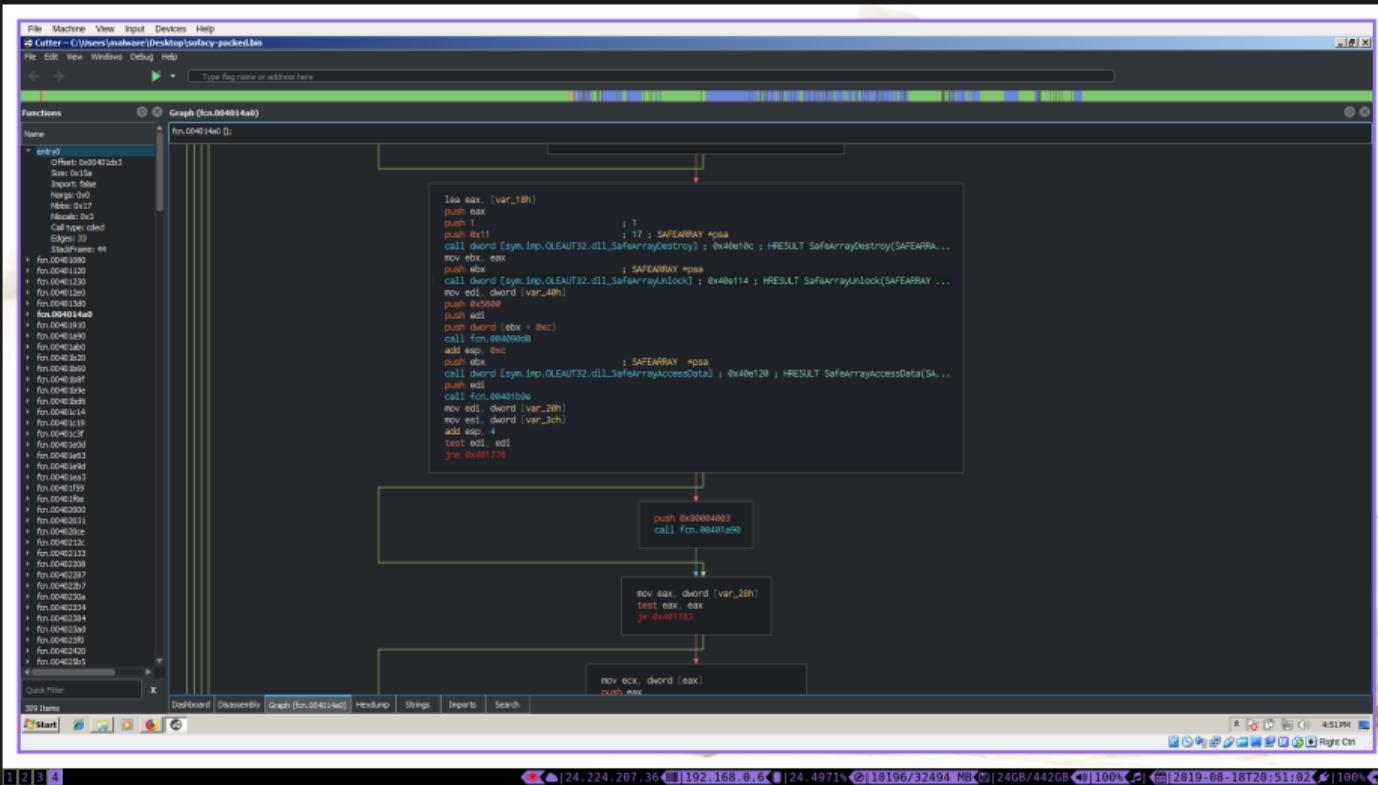
- Markup Reverse Engineered Code
- Control Flow Navigation
- Pseudo Code



# 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:

```

fn.004014a0 ;+00000000000000000000000000000000
    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] ; 0x40100c ; HRESULT SafeArrayDestroy(SAFEARRAY **psa, ...
    add esp, 4

    push ebx
    push 0x5000 ; SAFEARRAY *psa
    call dword [sym.imp.OLEAUT32.dll_SafeArrayUnlock] ; 0x401114 ; HRESULT SafeArrayUnlock(SAFEARRAY **psa, ...
    add esp, 4

    push ebx
    push 0x10000000 ; var_40h
    mov edi, dword [var_40h]
    push 0x5000
    push edi
    push word [ebx + 0xc]
    call dword [oleaut32.dll_OleSafeArrayAccessData]
    add esp, 4

    push ebx
    push 0x10000000 ; var_40h
    mov edi, dword [var_40h]
    mov esi, dword [var_3ch]
    add esp, 4
    test edi, edi
    jne 0x401778

    push 0x00004003
    call fcn.00401400

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

```

The imports pane lists the following functions:

Address	Type	Safety	Name
0x0040120	FUNC		OLEAUT32.dll_SafeArrayAccessData
0x004010c	FUNC		OLEAUT32.dll_SafeArrayDestroy
0x0040124	FUNC		OLEAUT32.dll_SafeArrayDelete
0x0040128	FUNC		OLEAUT32.dll_SafeArrayLock

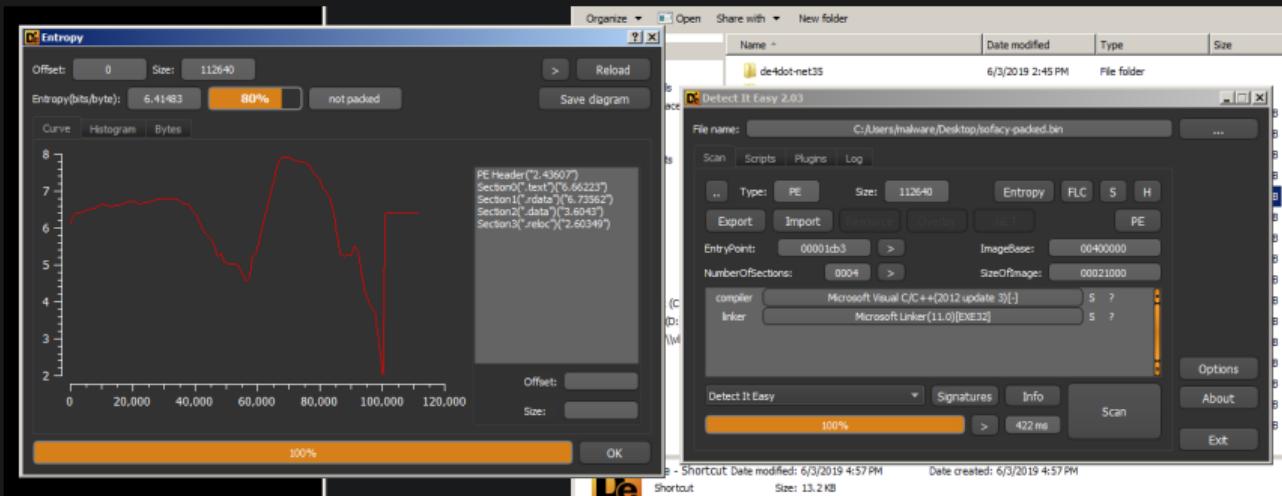
tools\_of\_the\_trade: 0x08

# Detect it Easy

tools\_of\_the\_trade: 0x09

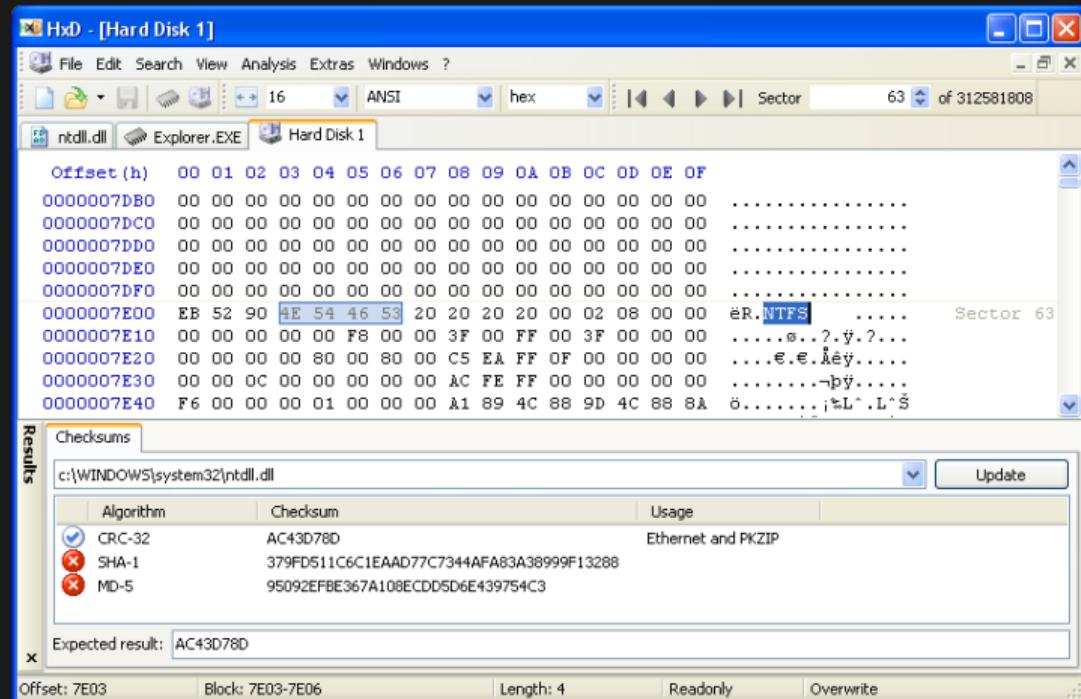
GoSECURE

- Type
- Packer
- Linker
- Entropy



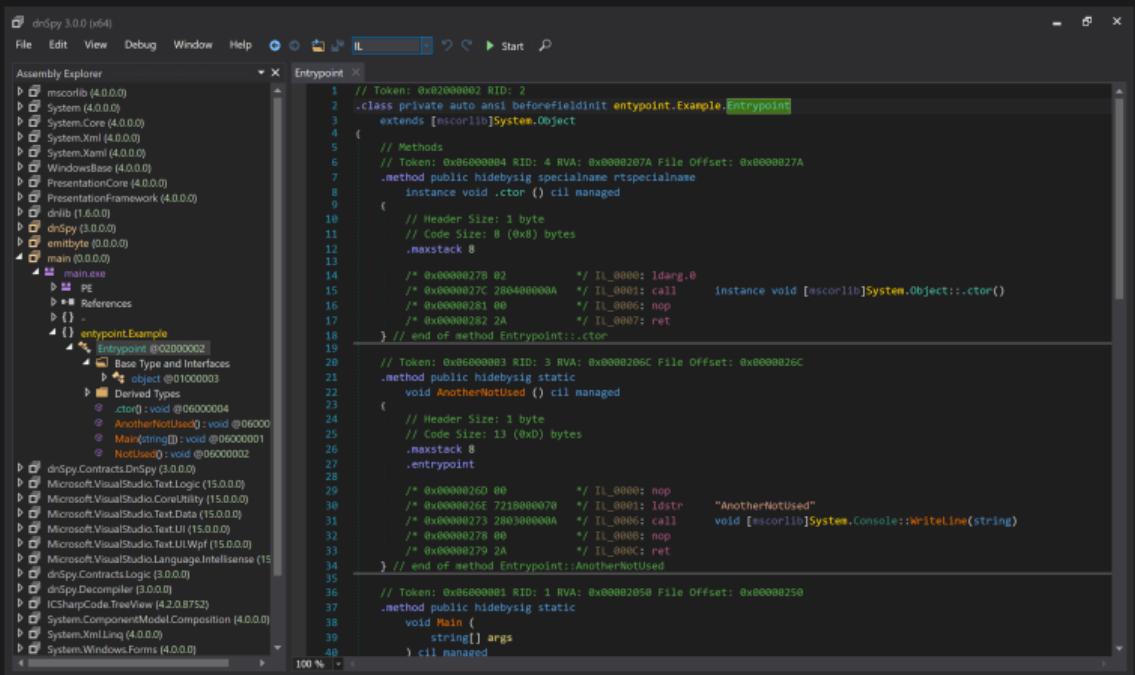
tools\_of\_the\_trade: 0x0a

- 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 Editor tabs selected. 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 expanded to show its internal types: maineexe, entrypoint.Example, and entrypoint.AnotherNotUsed. The Assembly Editor pane displays the IL code for the entrypoint.Example class. The code defines a private auto ansi beforefieldinit constructor for entrypoint.Example. The constructor calls the base class constructor and contains a single method AnotherNotUsed which prints "AnotherNotUsed" to the console. The code is annotated with assembly offsets and comments.

```
// Token: 0x02000002 RID: 2
.class private auto ansi beforefieldinit entrypoint.Example
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()
    } // end of method entrypoint..ctor

    // Token: 0x06000083 RID: 3 RVA: 0x0000206C File Offset: 0x0000026C
    .method public hidebysig static
        void AnotherNotUsed () cil managed
    {
        // Header Size: 1 byte
        // Code Size: 13 (0xD) bytes
        .maxstack 8
        .entrypoint
        /* 0x00000260 00 */ // IL_0000: nop
        /* 0x0000026E 721B0000070 */ // IL_0001: ldstr "AnotherNotUsed"
        /* 0x00000273 2B0300000A */ // IL_0006: call void [mscorlib]System.Console::WriteLine(string)
        /* 0x00000278 00 */ // IL_0008: nop
        /* 0x00000279 2A */ // IL_000C: ret
    } // end of method entrypoint::AnotherNotUsed

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

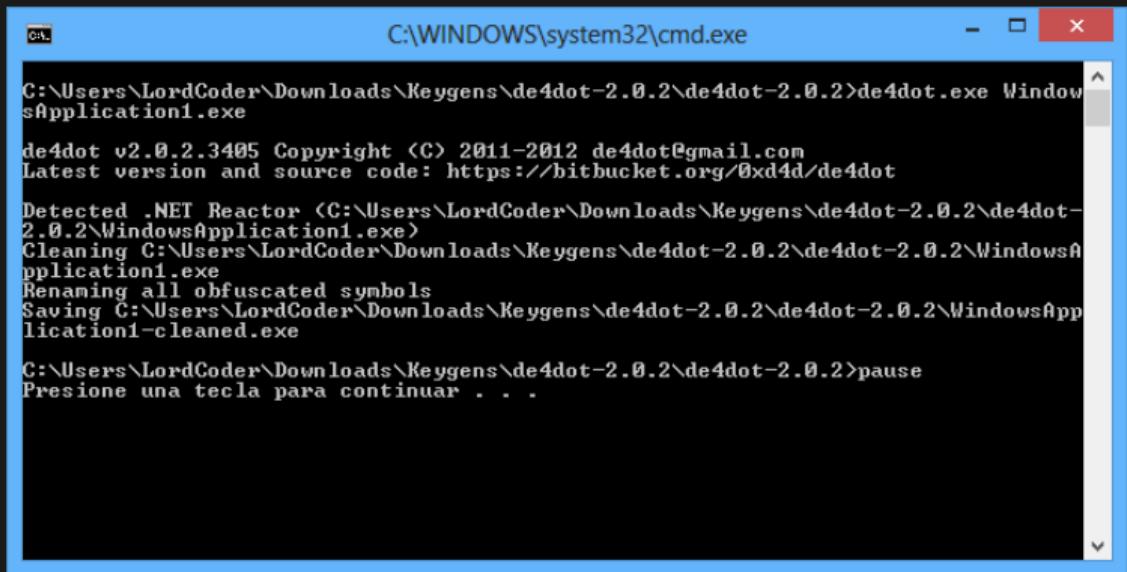
tools\_of\_the\_trade: 0x0c

## 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



```
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--
./shelic : 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)

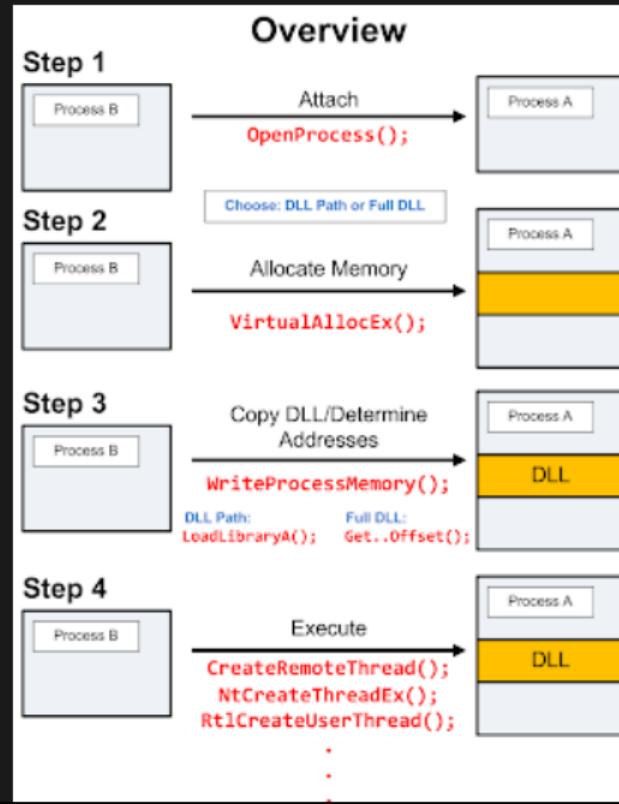
--output options--
./ofilter <ofilter_id>
      : Filter the dumped output.
*ofilter_id:
      0 - no filter: dump everything (default)
      1 - don't dump the modified PEs, but save the report
      2 - don't dump any files
./quiet   : Print only the summary. Do not log on stdout during the scan.
./json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



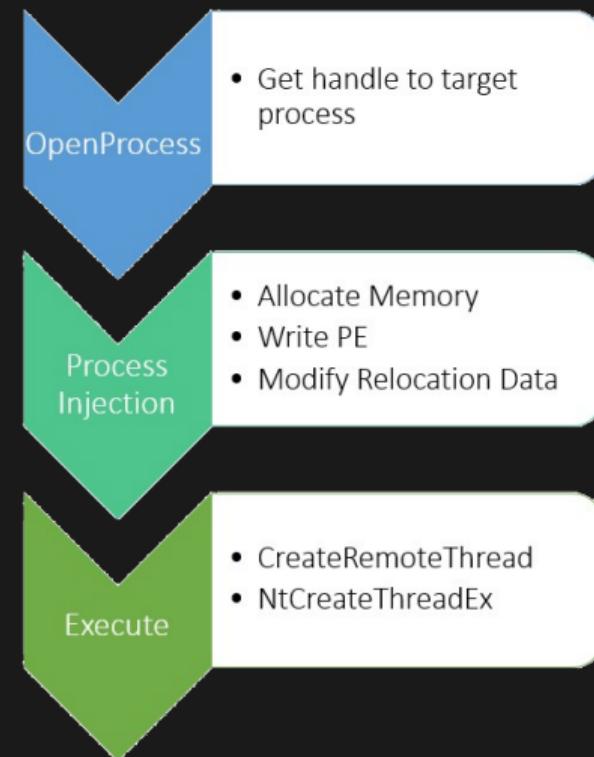
injection\_techniques: 0x00

- 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



# Atom Bombing

injection\_techniques: 0x03

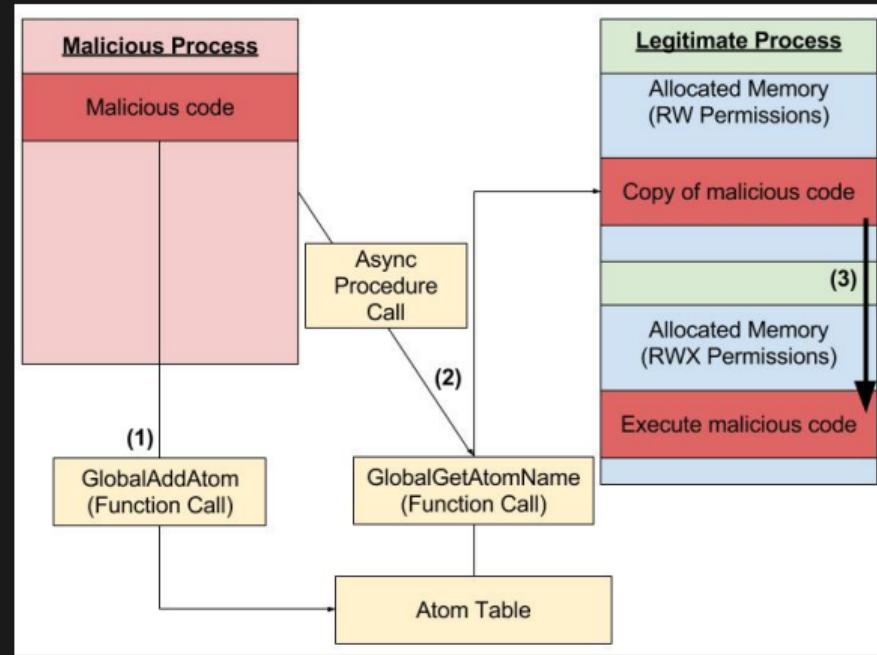


Atomic bomb test in Italy, 1957,  
colorized

# Atom Bombing

injection\_techniques: 0x04

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



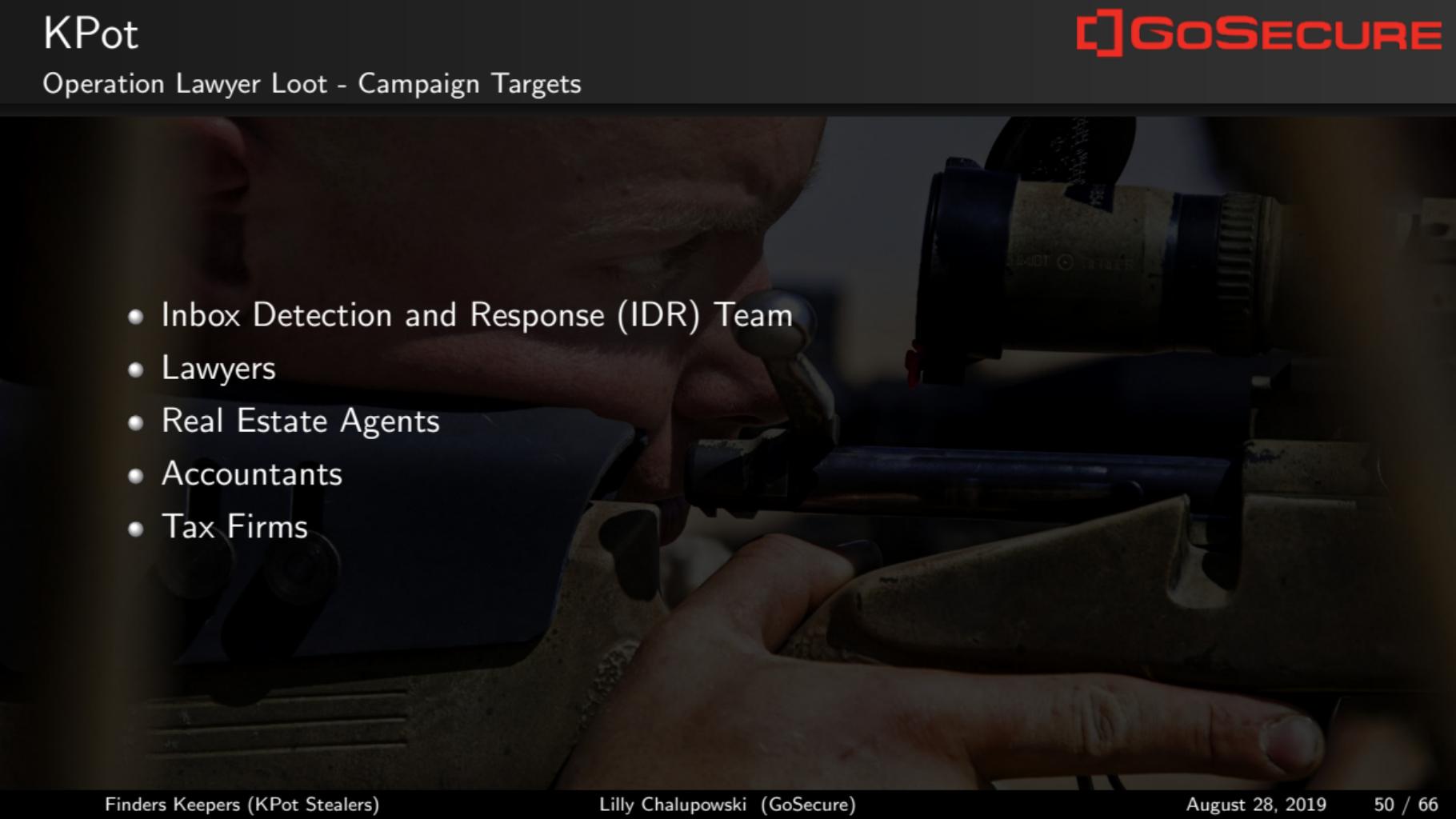
- 5466c52191ddd1564b4680060dc329cb



## Initial Infection Vector

MalSpam Email	
<b>From:</b>	< <a href="mailto:sidney.m@carmellaw.com">sidney.m@carmellaw.com</a> >
<b>Subject:</b>	Wire confirmation
<b>Attached:</b>	img-000310519000.img
<b>Body:</b>	<p>Hello,</p> <p>Kindly find attached the confirmation for your reference</p> <p>please revert back ASAP</p> <p>Regards</p> <p>Sidney morris</p>

## Operation Lawyer Loot - Campaign Targets

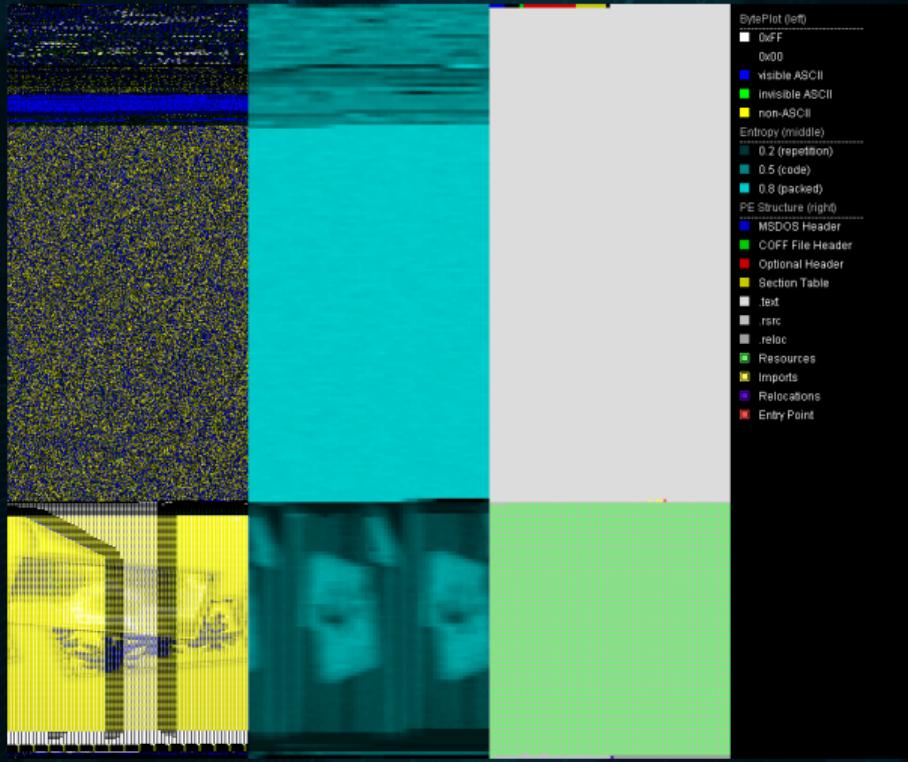
- 
- Inbox Detection and Response (IDR) Team
  - Lawyers
  - Real Estate Agents
  - Accountants
  - Tax Firms

## Operation Lawyer Loot - Extracting the Payload

terminal

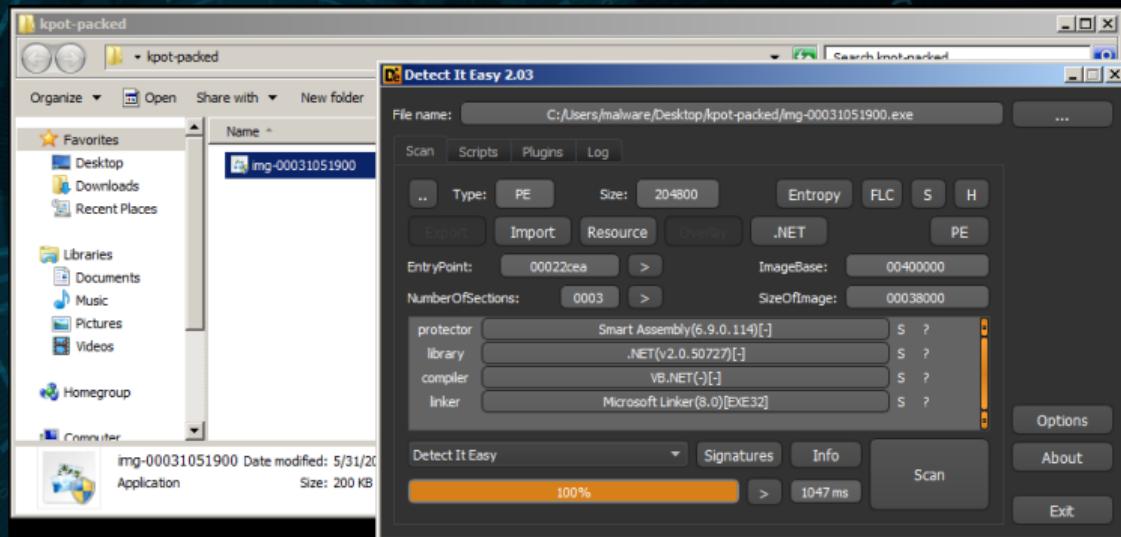
```
malware@localhost ~$ file img-000310519000.img
UDF filesystem data (version 1.5) NEW_FOLDER_2_
malware@localhost ~$ 7z e img-000310519000.img
malware@localhost ~$ file img-00031051900.exe
PE32 executable (GUI) Intel 80386 Mono/.Net assembly, for MS
Windows
malware@localhost ~$
```

## Operation Lawyer Loot - Portex Analyzer Entropy



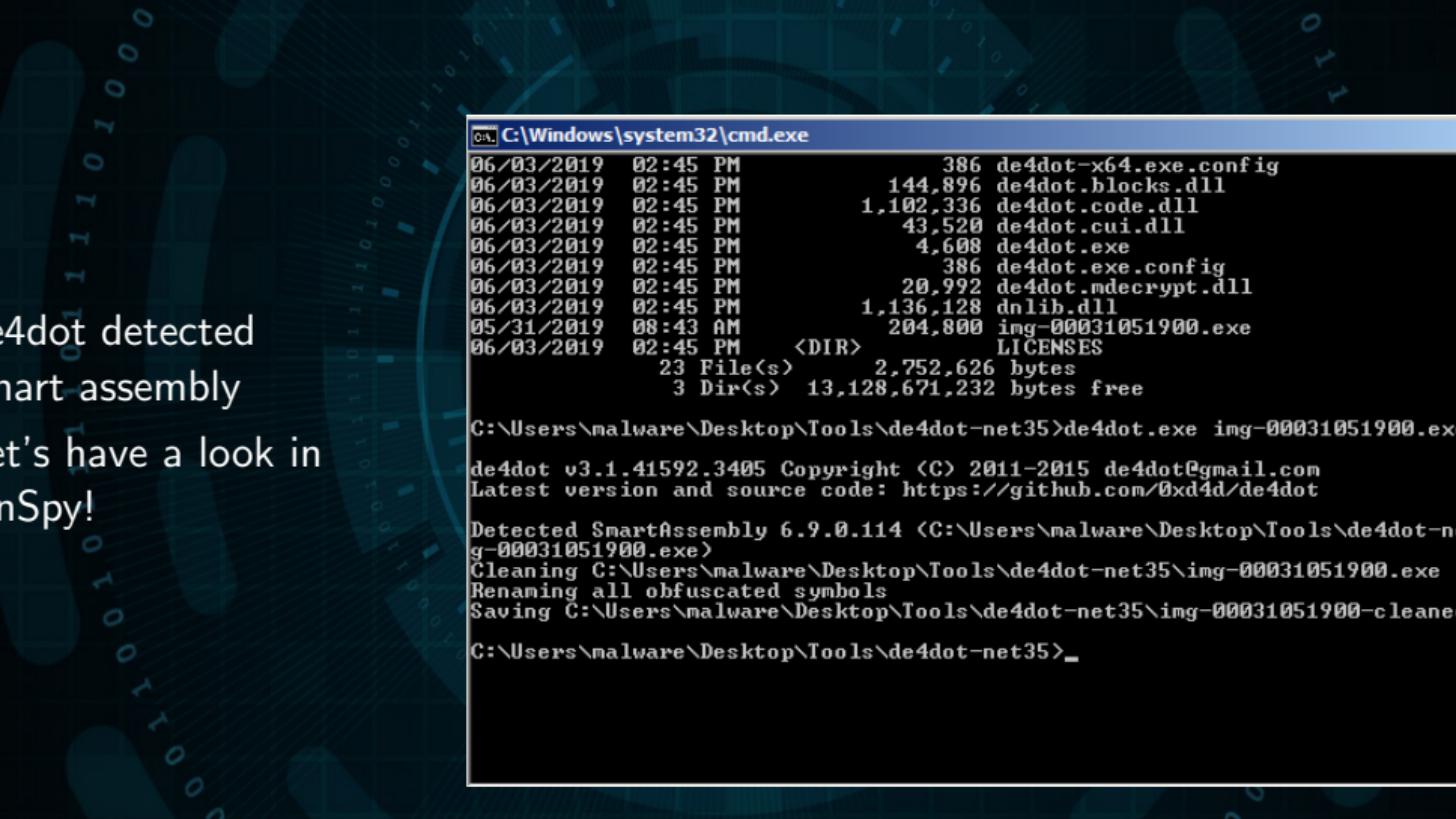
## Operation Lawyer Loot - Determining the Stage 1 Packer

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



## Operation Lawyer Loot - Unpacking Stage 1

- 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>
```

## Operation Lawyer Loot - Identifying Stage 2 Packed Data

**Packed Data in Resources**

```

Assembly Explorer
pgk2 (0.0.0)
  pgk2.exe
    PE
    References
    Resources
      dada.Resources.resources
        SmartAssembly.SmartUsageWithUI.ConfirmFeatureUsageReport.png
        SmartAssembly.SmartUsageWithUI.Resources.current.png
        SmartAssembly.SmartUsageWithUI.Resources.data.png
        SmartAssembly.SmartUsageWithUI.Resources.default.ico
        SmartAssembly.SmartUsageWithUI.Resources.error.png
        SmartAssembly.SmartUsageWithUI.Resources.error16.png
        SmartAssembly.SmartUsageWithUI.Resources.network.png
        SmartAssembly.SmartUsageWithUI.Resources.ok.png
        SmartAssembly.SmartUsageWithUI.Resources.warning16.png
        SmartAssembly.SmartUsageWithUI.Resources.(logo).png
      八港国港美七从
        八港国港美七从
  <Module> @02000001
  Class15 @02000028
  dada.My
  ns0
    Class0 @02000002
      Base Type and Interfaces
      Derived Types
        Class1 @02000003
        Class2 @02000005
        Class3 @02000004
    Class23 @02000032
  ns1
    Class4 @02000006
    Class5 @02000007
    Class8 @0200000A
    Class9 @0200001F
  ns2
  ns3
  ns4

```

**Get Packed Data**

```

1347 if (Operators.ConditionalCompareObjectEqual(executablePath, text + "#nsgdfgdsp$$$$.exe$$$"), false))
1348 {
1349   goto IL_150;
1350 }
IL_15A:
1351 num2 = 23;
1352 IL_15D:
1353 num2 = 26;
1354 string sourceFileName = Interaction.Environ(Class15.smethod_30("8HMhLTGvQOih4lcE50F9A==")) + Class15.smethod_30("rPYkL:
1355 Class15.smethod_30("8HM1hM12pL90Lp7wY5d2wg==");
1356 IL_18A:
1357 num2 = 27;
1358 IL_18D:
1359 num2 = 28;
1360 IL_190:
1361 num2 = 29;
1362 string destFileName = "" + str + "\\\" + text2;
1363 IL_1A6:
1364 num2 = 30;
1365 byte[] byte_ = (byte[])resourceManager.GetObject("八港国港美七从");
1366 
1367 num2 = 31;
1368 byte[] array2 = Class15.smethod_6(byte_, Class15.smethod_30("Hb7onq2ZzgBw+mmuIDsSYXZug4//mjETwOU+Mi913jw="));
1369 
1370 num2 = 32;
1371 File.Delete(text + text2);
1372 IL_1E2:
1373 num2 = 33;
1374 File.Copy(sourceFileName, destFileName, true);
1375 IL_1EF:
1376 num2 = 34;
1377 Class15.smethod_16(new object[])
1378 {
1379   string.Empty,
1380   array2,
1381   false,
1382   false,
1383   Application.ExecutablePath
1384 });
1385 
1386 num2 = 35;

```

**Unpacking Packed Data**

**Injection Funtion**

## Operation Lawyer Loot - Stage 2 String Decryptor

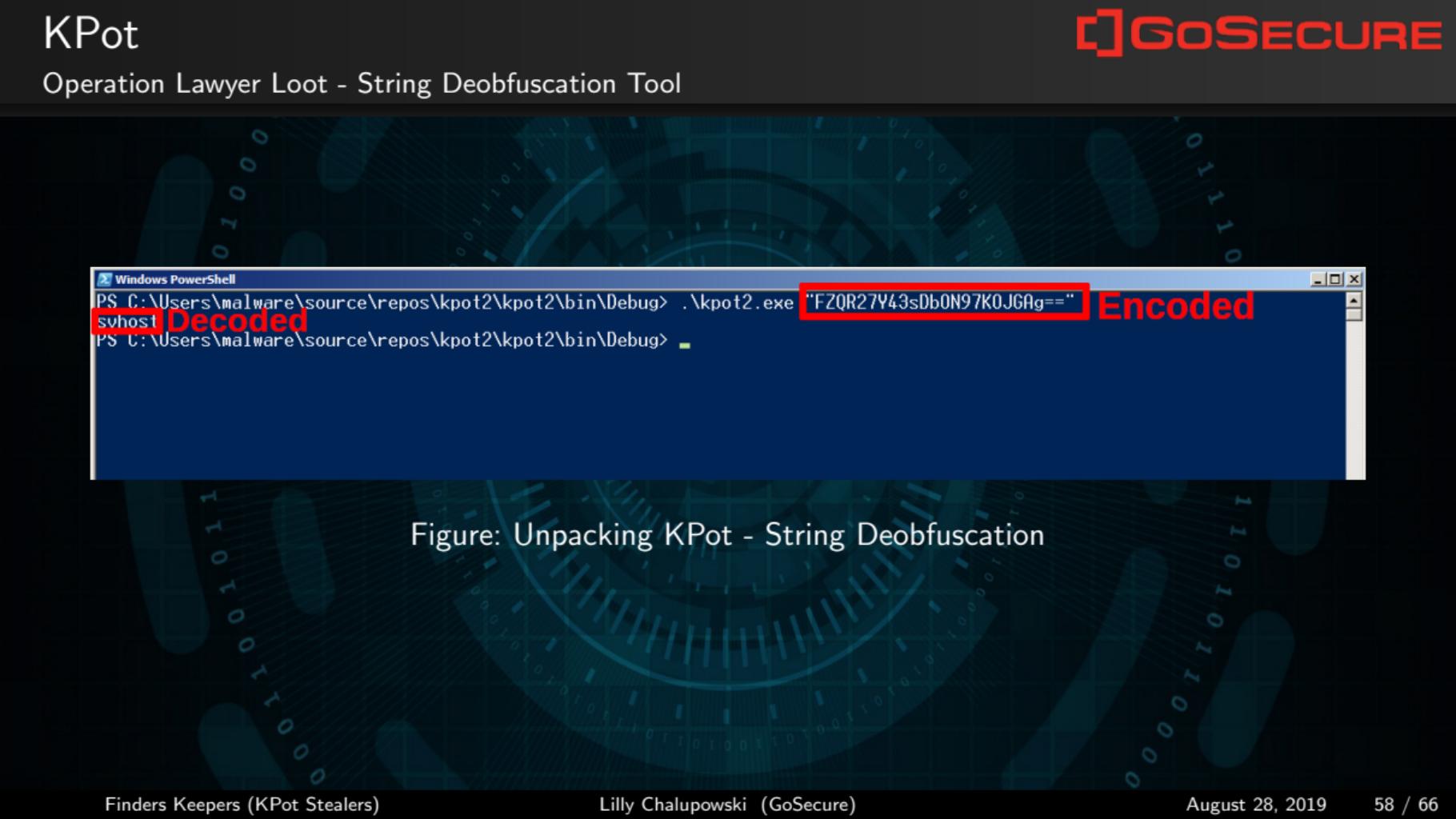
- Uses Rijndael for string obfuscation
- We can use C# in Visual Studio!

```
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;
}
```

## Operation Lawyer Loot - Stage 2 String Decryptor in Visual Studio

```
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]));
    }
}
```

## Operation Lawyer Loot - String Deobfuscation Tool



```
Windows PowerShell
PS C:\Users\malware\source\repos\kpot2\kpot2\bin\Debug> .\kpot2.exe "FZQR27Y43sDb0N97KOJGAg==" Encoded
svhost Decoded
PS C:\Users\malware\source\repos\kpot2\kpot2\bin\Debug>
```

Figure: Unpacking KPot - String Deobfuscation

## Operation Lawyer Loot - Stage 2 Injection Method

```
static bool smethod_27(object[] object_0)
{
    Class5.Class6 @class = new Class5.Class6();
    Class8.Delegate1 @delegate = Class5.smethod_0<Class8.Delegate1>(Class15.smethod_36["uzbZJ0XMI1ti/o9VzPqHexQ=="], Class15.smethod_36["kvDwt2m955ig9LvIKq8J7Q=="]);
    Class8.Delegate2 delegate2 = Class5.smethod_0<Class8.Delegate2>(Class15.smethod_36["uzbZJ0XMI1ti/o9VzPqHexQ=="], Class15.smethod_36["M+Cx1wPRKQCPhR5g7XQekaqAMuhh600EKoaAbNvIk=="]);
    Class8.Delegate3 delegate3 = Class5.smethod_0<Class8.Delegate3>(Class15.smethod_36["uzbZJ0XMI1ti/o9VzPqHexQ=="], Class15.smethod_36["OCVByDSkb4dymngcmPozmijex0C140ik/RQ2beXPcDo=="]);
    Class8.Delegate4 delegate4 = Class5.smethod_0<Class8.Delegate4>(Class15.smethod_36["uzbZJ0XMI1ti/o9VzPqHexQ=="], Class15.smethod_36["FjiFXnN7CBcQa33vgPA3B55YktkkXqzt0GLjZJiitb8=="]);
    Class8.Delegate5 delegate5 = Class5.smethod_0<Class8.Delegate5>(Class15.smethod_36["eDAECoHuT4guqKSv0Gp4yg=="], Class15.smethod_36["e4WJnV1/R9vgIA+MBU/59/Urjhz+fgwIRBB0+3YKw=="]);
    Class8.Delegate6 delegate6 = Class5.smethod_0<Class8.Delegate6>(Class15.smethod_36["eDAECoHuT4guqKSv0Gp4yg=="], Class15.smethod_36["oKL4yaE9EBm/y95+kqvCkaq4w3M218t7/g7gAFxxVD4=="]);
    Class8.Delegate7 delegate7 = Class5.smethod_0<Class8.Delegate7>(Class15.smethod_36["uzbZJ0XMI1ti/o9VzPqHexQ=="], Class15.smethod_36["BGwD0JvCcYuc0GAsN3P9tw=="]);
    Class8.Delegate8 delegate8 = Class5.smethod_0<Class8.Delegate8>(Class15.smethod_36["eDAECoHuT4guqKSv0Gp4yg=="], Class15.smethod_36["iy1lijydF6nv90hRn++0LQ=="]);

    string text = (string)object_0[0];
    byte[] array = (byte[])object_0[1];
    bool flag = (bool)object_0[2];
    bool flag2 = (bool)object_0[3];
    string text2 = (string)object_0[4];
    int num = 0;
    string text3 = string.Format("\\"{0}\\\"", text2);
    Class8.Struct1 @struct = default(Class8.Struct1);
    @class.struct0_0 = default(Class8.Struct0);
    @struct.uint_0 = Convert.ToInt32(Marshal.SizeOf(typeof(Class8.Struct1)));
    bool result;
    try
    {
        Class5.Class6.Class7 class2 = new Class5.Class6.Class7();
        class2.class6_0 = @class;
        if (!string.IsNullOrEmpty(text))
            class2.class6_0.smethod_27(text3, array, flag, flag2, num);
    }
}
```

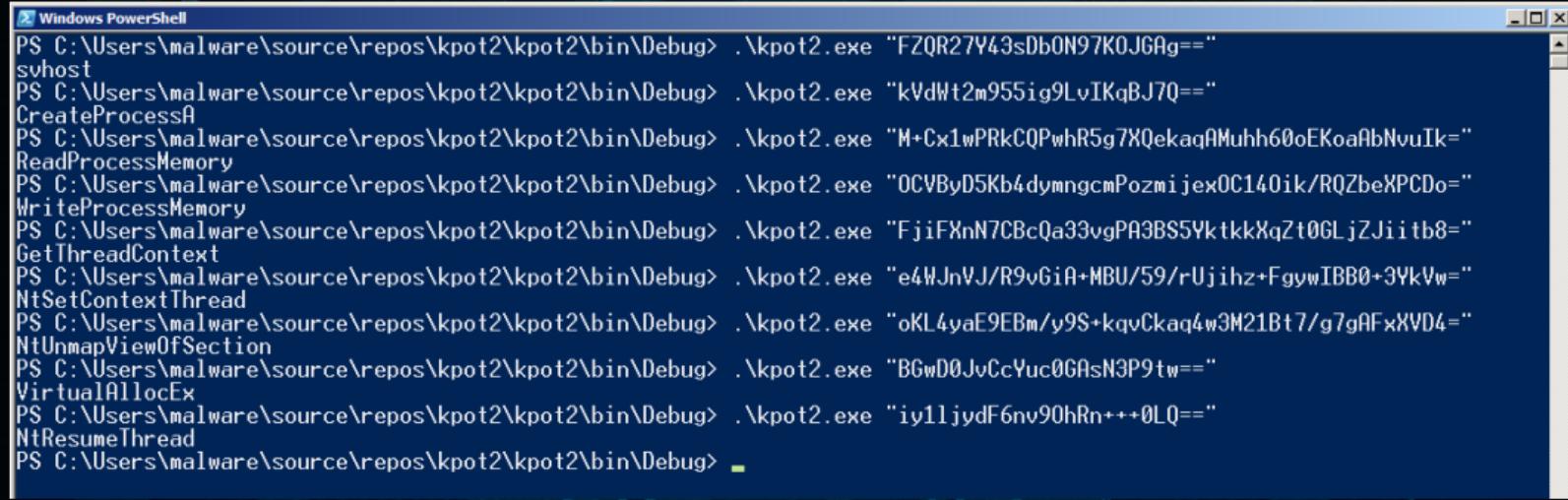
Figure: Unpacking KPot - API Function Name Encryption

## Operation Lawyer Loot - Stage 2 Injection API Function Name Decryption

```
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 - Decrypt Injection API Function Names

## Operation Lawyer Loot - Stage 2 Injection API Function Name Decryption



```
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 "OCVByD5Kb4dyMngcmPozmijexOC140ik/RQZbeXPCDo==">
WriteProcessMemory
PS C:\Users\malware\source\repos\kpot2\kpot2\bin\Debug> .\kpot2.exe "FjiFXnN7CBcQa33vgPA3BS5YtkkXqZt0GLjZJiitb8==">
GetThreadContext
PS C:\Users\malware\source\repos\kpot2\kpot2\bin\Debug> .\kpot2.exe "e4WJnVJ/R9vGiA+MBU/59/rUjihz+FgywIBB0+3Vkvw==">
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 "iy11jydF6nv90hRn+++0LQ==">
NtResumeThread
PS C:\Users\malware\source\repos\kpot2\kpot2\bin\Debug>
```

Figure: Unpacking KPot - Process Hollowing Functions

## Operation Lawyer Loot - Stage 2 Unpacking Process Hollowing with pe-sieve

drapy v6.0.5 (32-bit, Administrator, Debugging)

File Edit View Debug Window Help

Assembly Explorer

Class15

```

981 }
982     if ((ulong)delegate5@class0.intptr_0.array2) != 0UL)
983     {
984         throw new Exception();
985     }
986     if (flag.s)
987     {
988         new Thread(new ThreadStart(class2.method_0)).Start();
989     }
990     goto IL_07E;
991     uint num11 = 0u;
992     if (delegate0@class0.intptr_1.out_num11 == -1)
993     {
994         throw new Exception();
995     }

```

**Set Breakpoint Here**

NtResumeThread

Administrator: Windows PowerShell

```

PS C:\ProgramData\chocolatey\lib\pesieve\tools> Ape-sieve.exe /imp /pid 1272 /dmode 0
PID: 1272
Modules filter: all accessible (default)
Output filter: no filter: dump everything (default)
Dump mode: autodetect (default)
Scanning workingset: 35 memory regions.
[+] Report dumped to: process_1272
[+] Dumped module to: process_1272\400000.rec.exe as Unmapped
[+] Dumped modified to: process_1272
PID: 1272
---
SUMMARY:
Total scanned: 0
Skipped: 0
-
Hooked: 0
Replaced: 0
Detached: 0
Implanted: 1
Other: 0
Total suspicious: 1
---
PS C:\ProgramData\chocolatey\lib\pesieve\tools> 

```

Unpack with pe-sieve

Process Hacker [malware-pc\malware]

Name	PID	CPU	I/O total ...	Private b...	User name	Description
audiod.exe	1528	15.39 MB				Windows Audio Device Gra
dwm.exe	1244	7.5 MB				Host Process for Windows
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
spoolv.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				Host Process for Windows
sppsvc.exe	1648	5.59 MB				Microsoft Software Protect
explorer.exe	1084	0.04	43.96 MB	malware-pc\malware		Windows Explorer
VBoxTray.exe	1652	2.77 MB				VirtualBox Guest Additions
drapy-v6.exe	2688	183.41 MB				drapy-v6
img-00030531900-ds...	2864	9.63 MB				Maxthon Installer
img-00030531900...	1272	456 MB				Maxthon Installer
powernell.exe	2490	36.19 MB				Windows PowerShell
ProcessHacker.exe	3060	35.77 MB				Process Hacker
powernell.exe	3060	35.77 MB				Windows PowerShell
SearchIndexer.exe	2224	21.55 MB				Microsoft Windows Search
imrnbtik.exe	2324	10.51 MB				Windows Media Player Net
svchost.exe	2584	10.71 MB				Host Process for Windows
svchost.exe	2056	63.05 MB				Host Process for Windows
crohost.exe	932	7.12 MB				General Windows Host

**Target Suspended Process**

0.class19\_0, int\_1, byte\_0

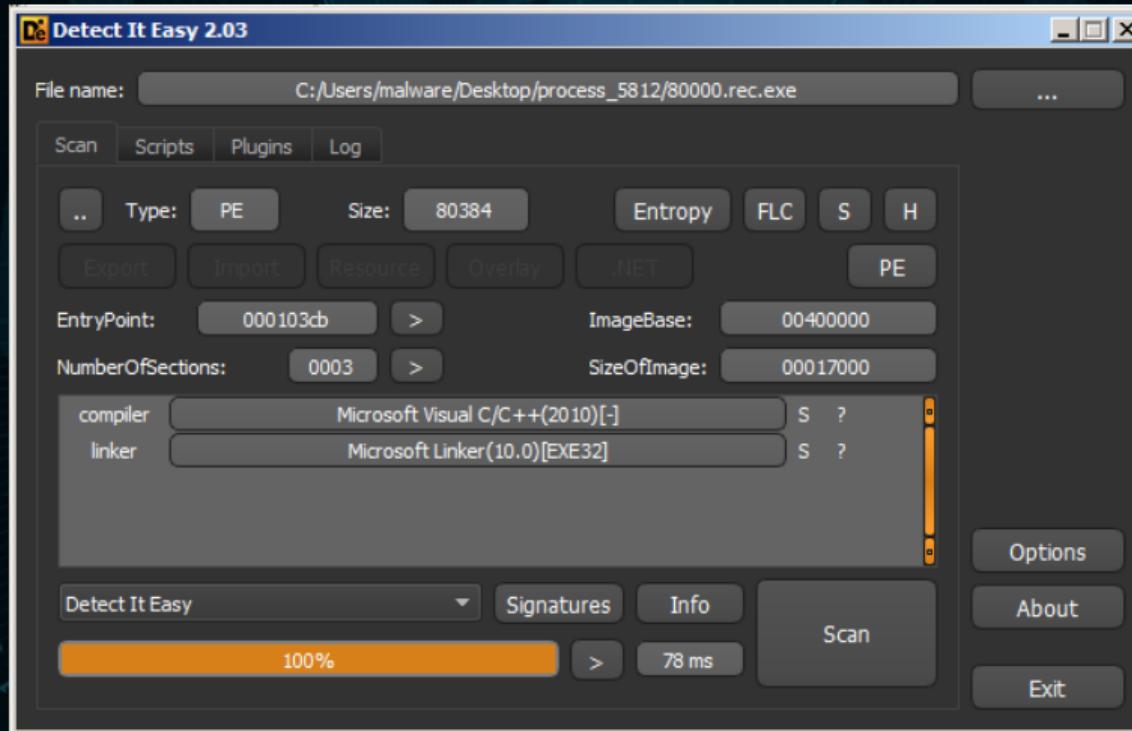
0.class19\_0, int\_1, byte\_0

0.class19\_0, int\_1, byte\_0

Type  
object  
ns1.Class5.Class6  
ns1.Class5.Parameter1

Operation Lawyer Loot - Stealer Strings

## Operation Lawyer Loot - Final Payload C++?



Are we done?

GoSECURE



imgflip.com

- [https://en.wikibooks.org/wiki/X86\\_Disassembly/Calling\\_Conventions](https://en.wikibooks.org/wiki/X86_Disassembly/Calling_Conventions)
- <https://github.com/m0n0ph1/Process-Hollowing>
- <http://blog.sevagas.com/?PE-injection-explained>
- [https://en.wikipedia.org/wiki/DLL\\_injection](https://en.wikipedia.org/wiki/DLL_injection)