

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

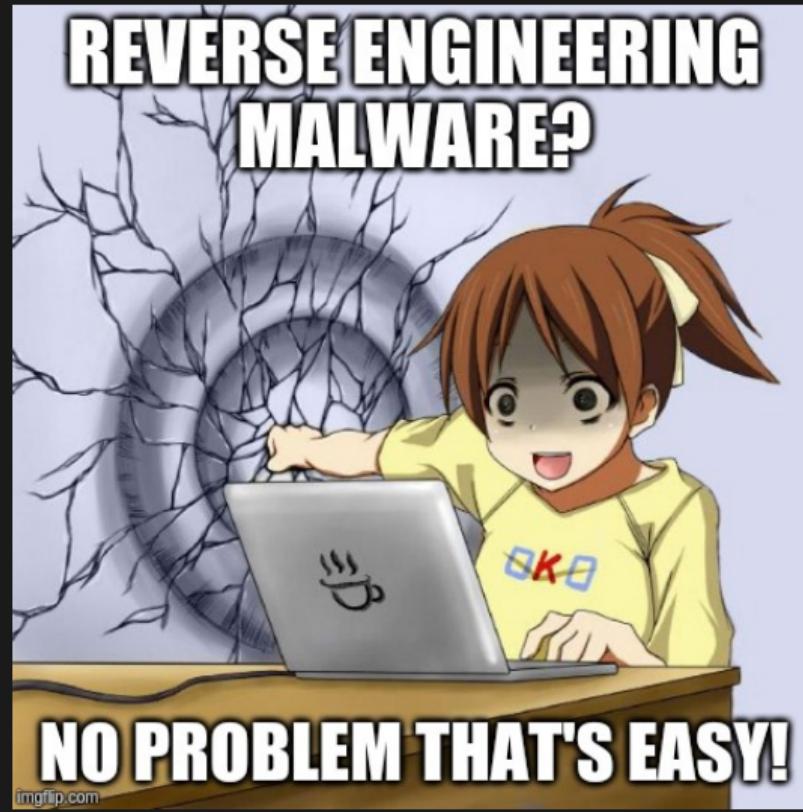
I (the speaker) do not assume any responsibility and shall not be held liable for anyone who infects their machine with the malware supplied for this workshop.

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

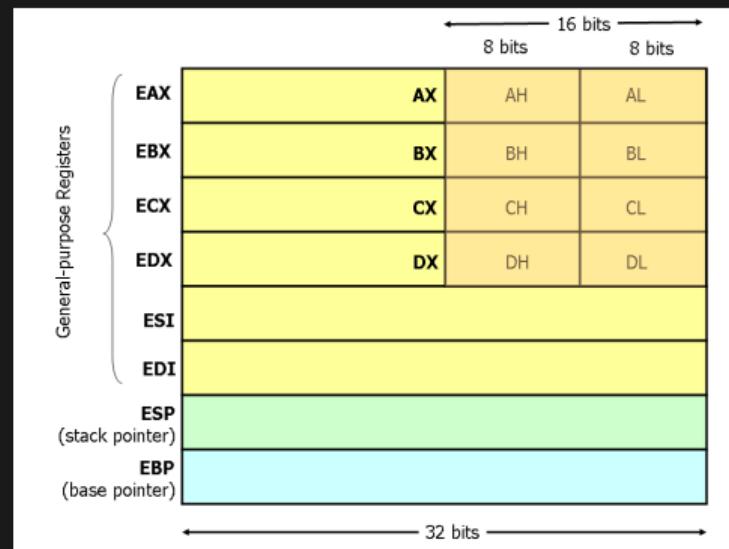
We choose to reverse engineer! We choose to reverse engineer... We choose to reverse engineer and do the other things, not because they are easy, but because they are hard; because that goal will serve to organize and measure the best of our energies and skills, because that challenge is one that we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



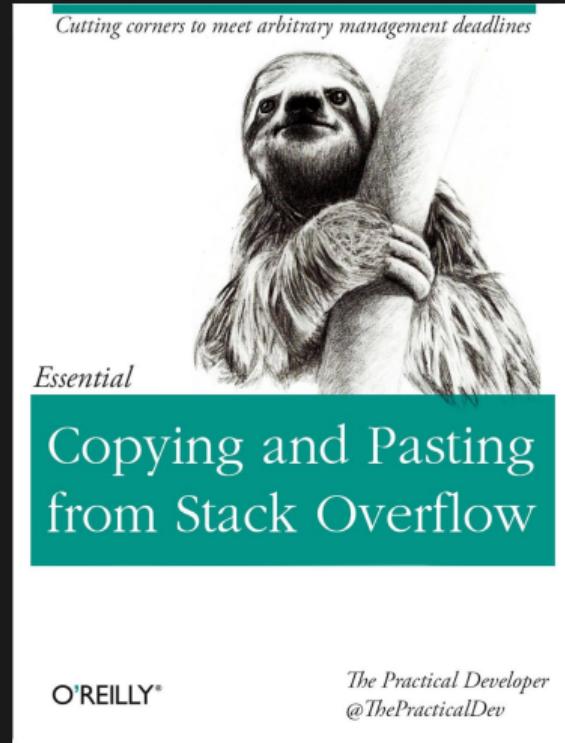
Did You Know: In computer architecture, a processor register is a quickly accessible location available to a computer's central processing unit (CPU).

Registers

reverse_engineering: 0x01

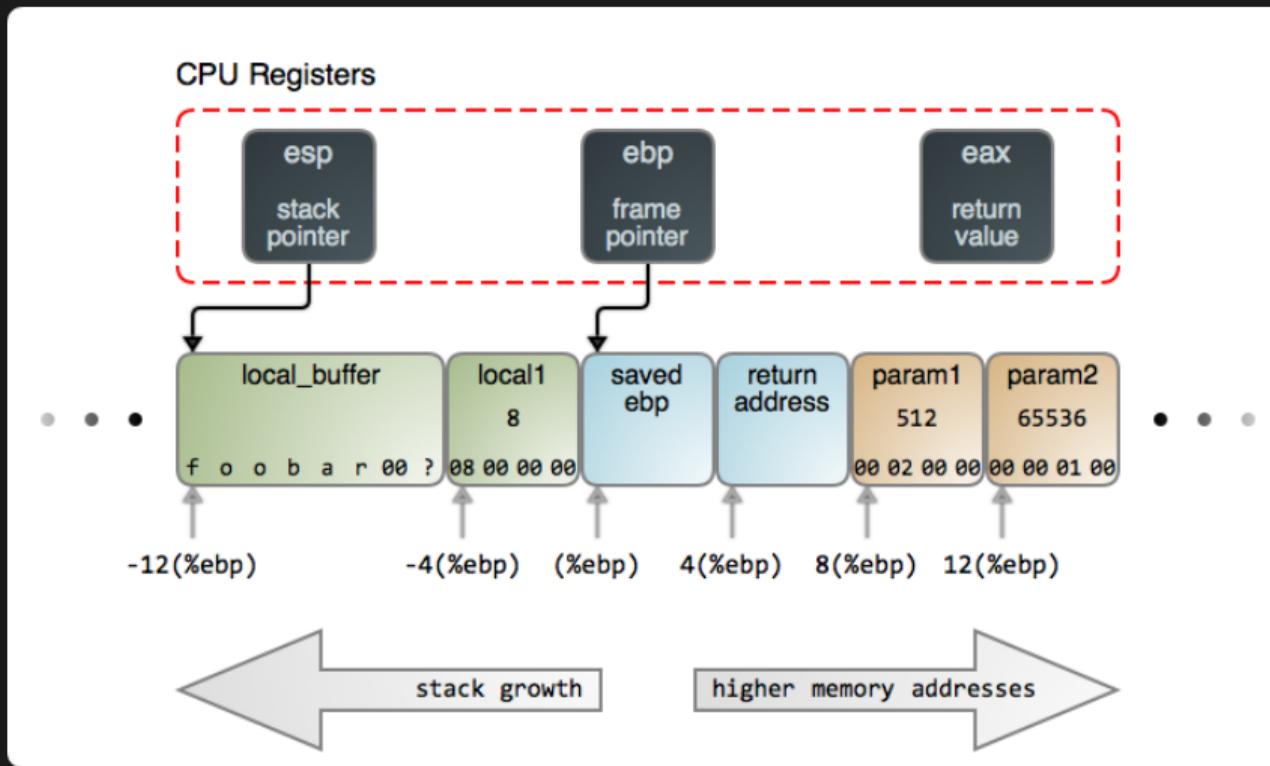


- Last-In First-Out
- Downward Growth
- Function Local Variables
- ESP
- Increment / Decrement = 4
 - Double-Word Aligned



Stack Structure

reverse_engineering: 0x03



reverse_engineering: 0x04

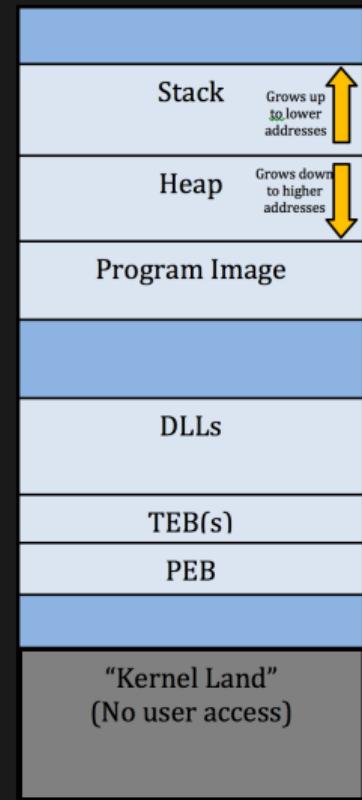
- Conditionals
 - CMP
 - TEST
 - JMP
 - JNE
 - JNZ
- EFLAGS
 - ZF / Zero Flag
 - SF / Sign Flag
 - CF / Carry Flag
 - OF/Overflow Flag



- 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

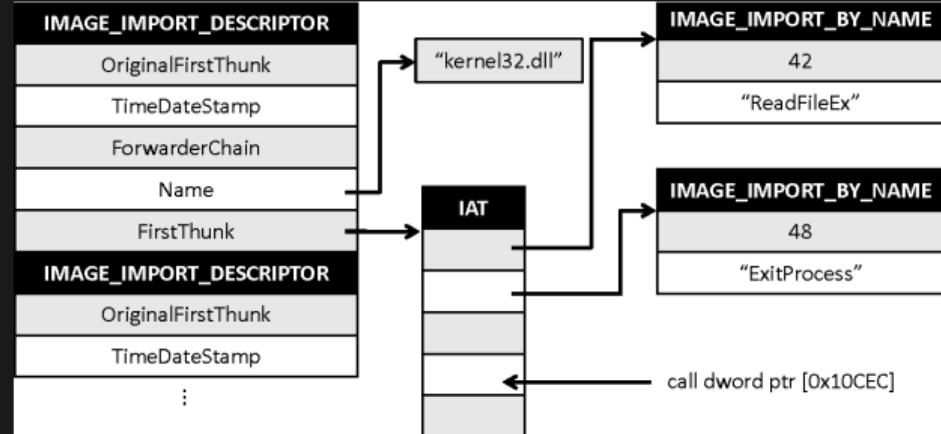


- Stack - Grows up to lower addresses
- Heap - Grows down to higher addresses
- Program Image
- TEB - Thread Environment Block
 - GetLastError()
 - GetVersion()
 - Pointer to the PEB
- PEB - Process Environment Block
 - Image Name
 - Global Context
 - Startup Parameters
 - Image Base Address
 - IAT (Import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

Assembly STDCALL (Windows)



reverse_engineering: 0x0b

stdcall.c

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

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

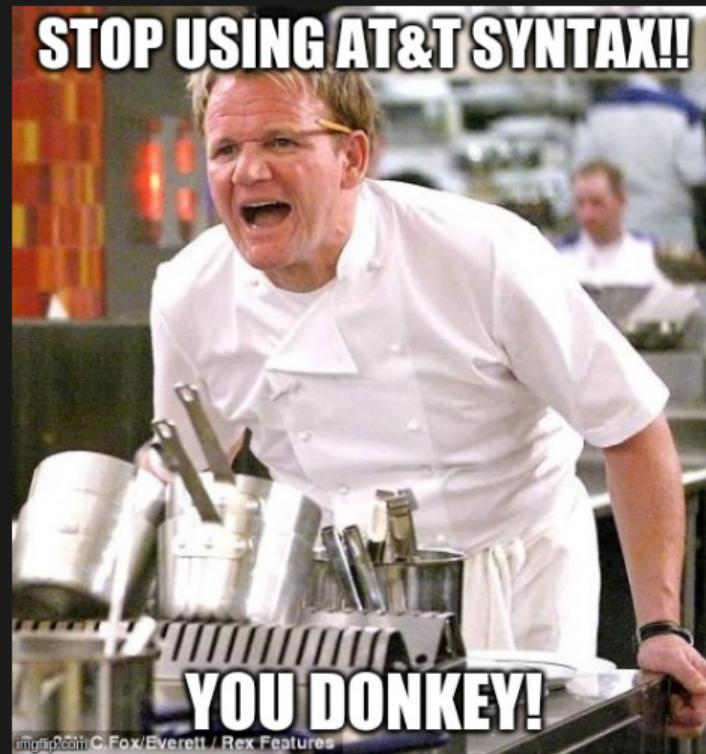
hello.asm

```
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 ~$
```

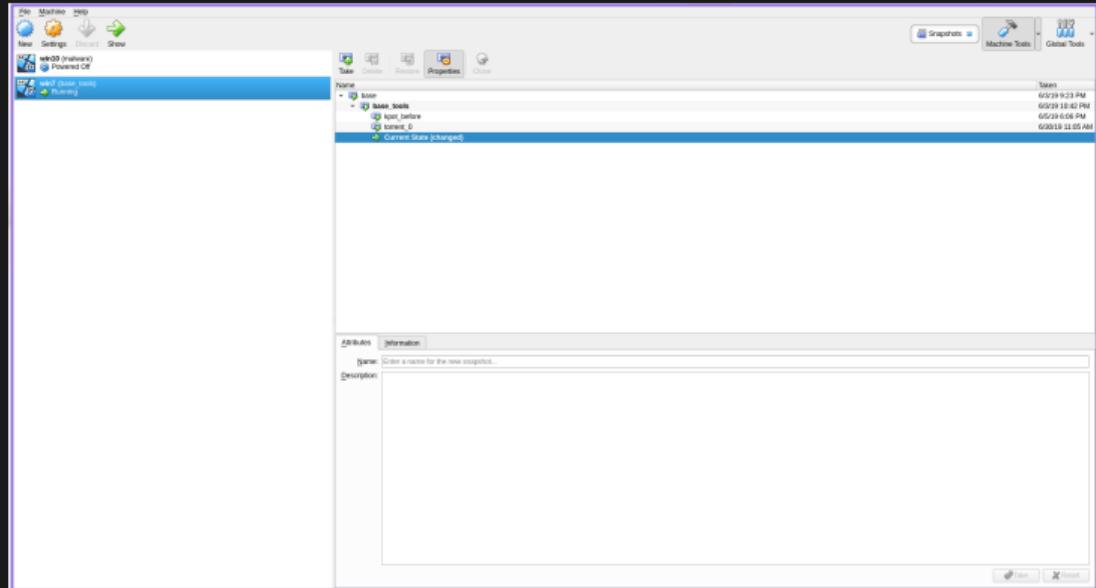


imgtip.com C. Fox/Everett / Rex Features



tools_of_the_trade: 0x00

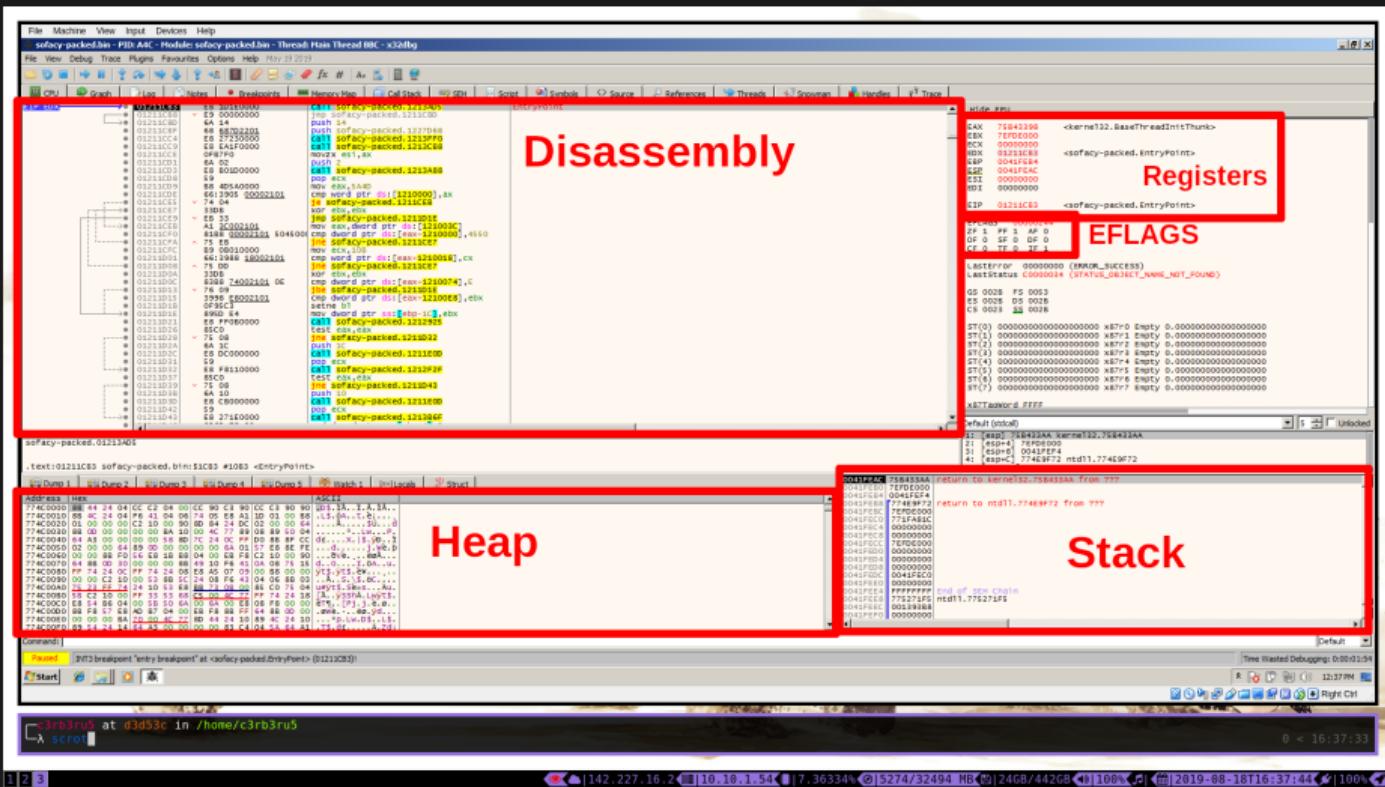
- Snapshots
- Security Layer
- Multiple Systems



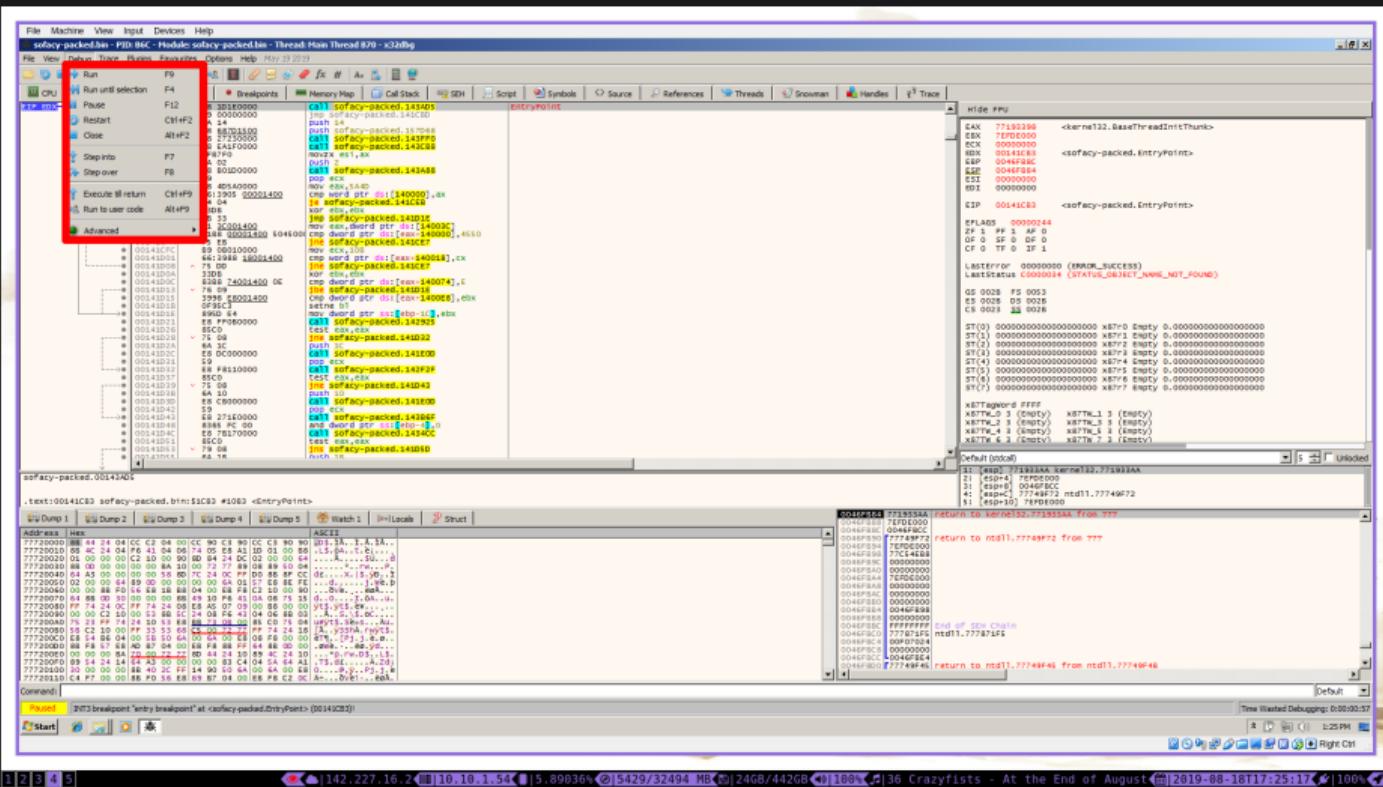
- Resolving APIs
- Dumping Memory
- Modify Control Flow
- Identify Key Behaviors



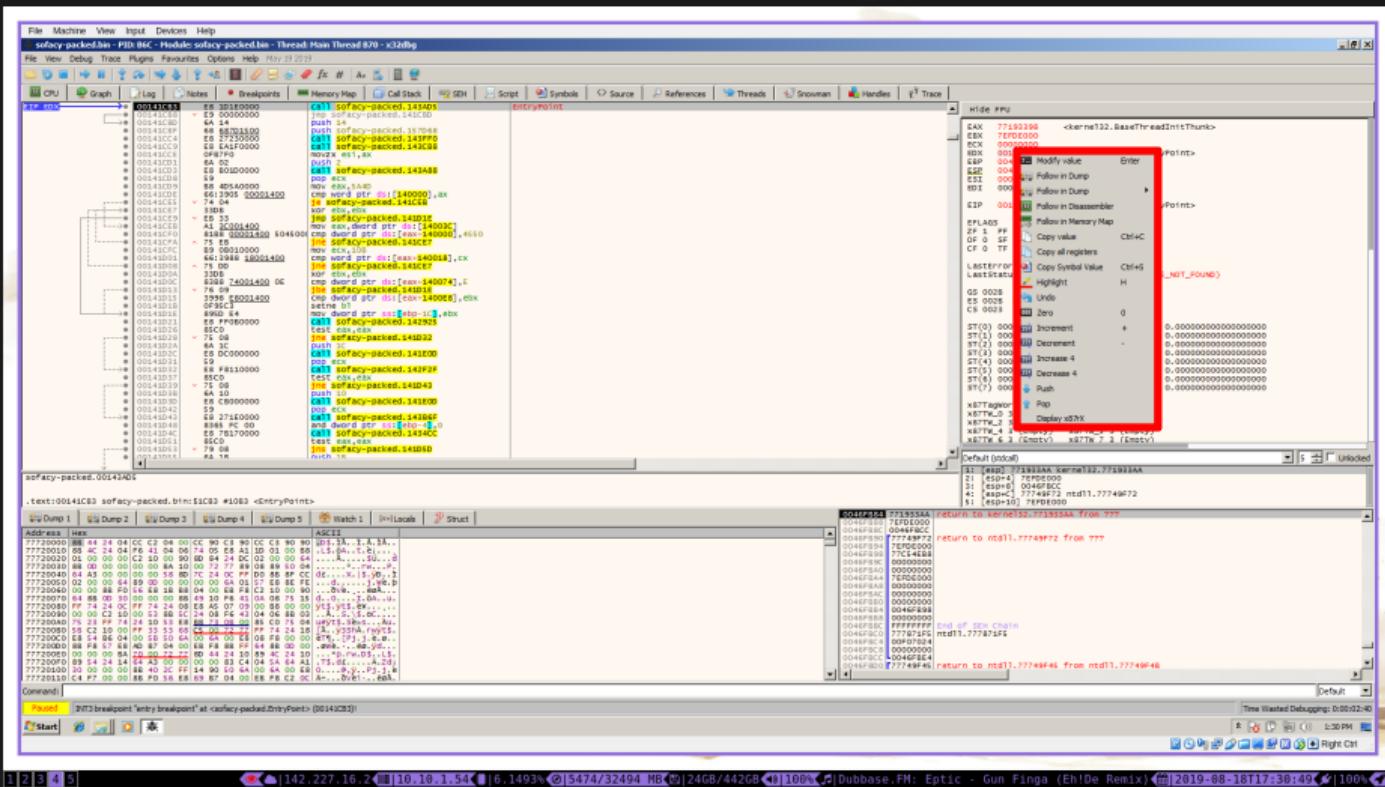
tools_of_the_trade: 0x02



tools_of_the_trade: 0x03

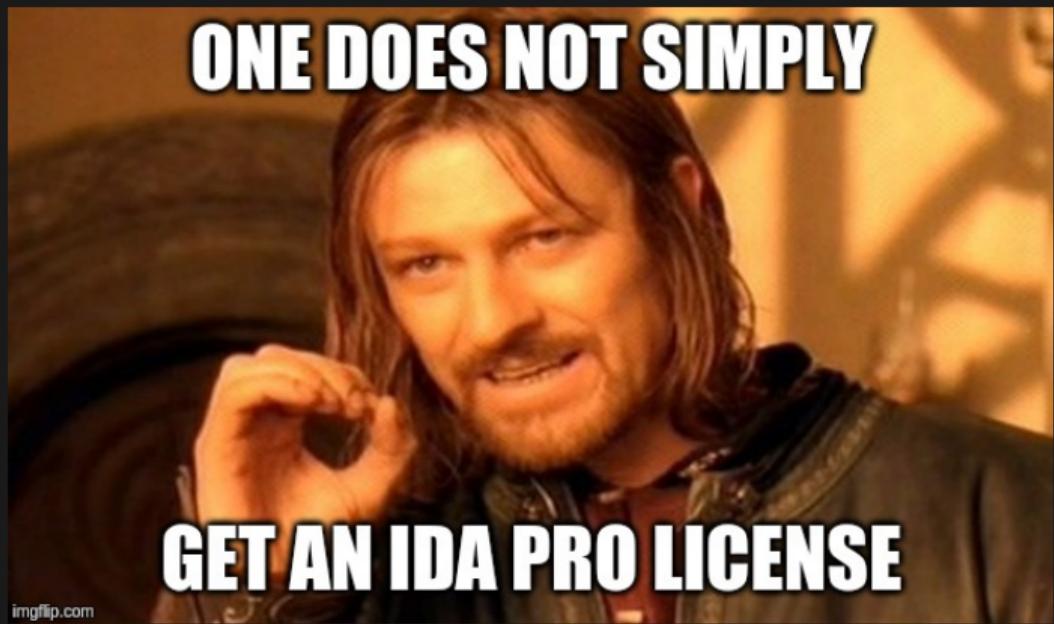


tools_of_the_trade: 0x04

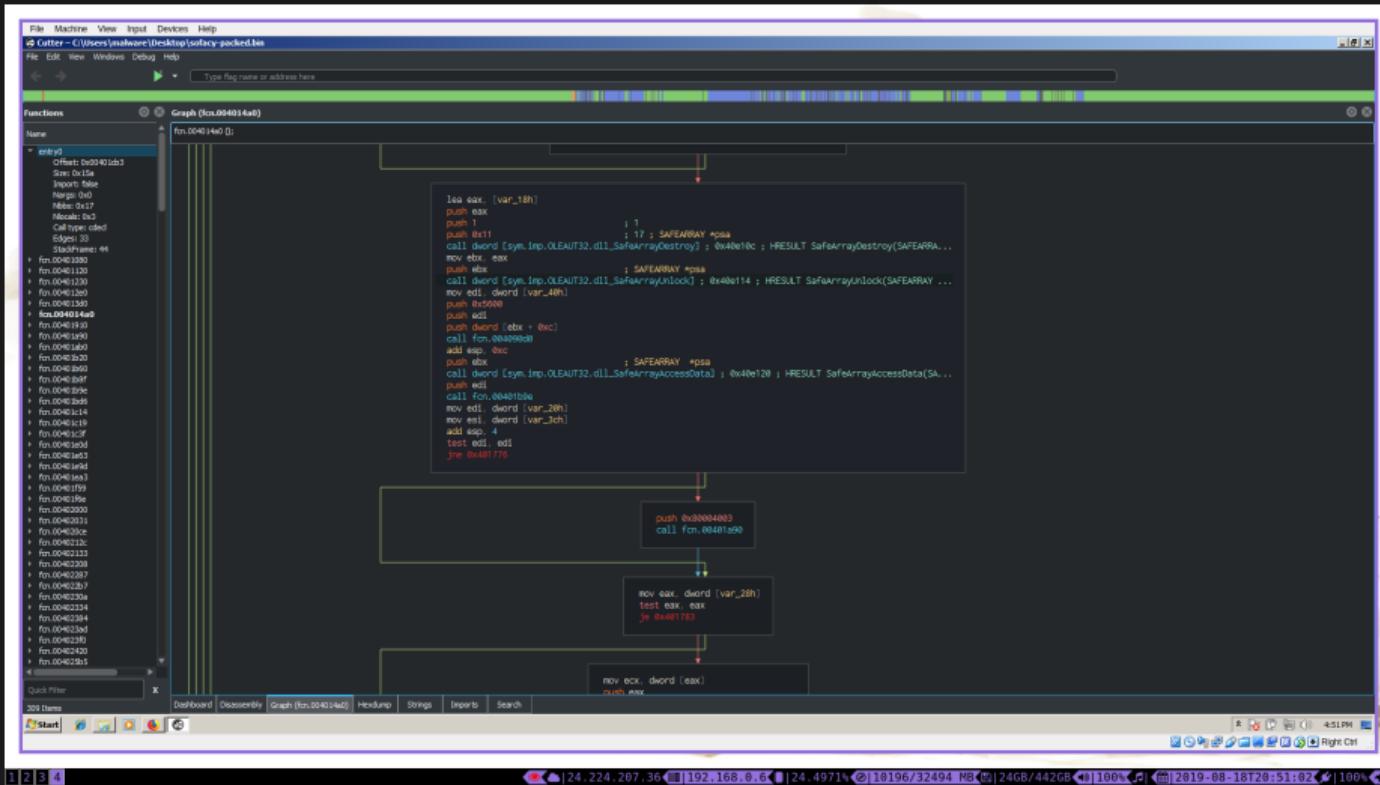


tools_of_the_trade: 0x05

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



tools_of_the_trade: 0x06



Cutter

 GoSECURE

tools_of_the_trade: 0x07

The screenshot shows the Immunity Debugger interface with the following details:

- File Menu:** File, Machine, View, Input, Devices, Help.
- Toolbar:** Cut/Copy/Paste, New, Windows, Debug, Help.
- Search Bar:** Type flag name or address here.
- Functions Tab:** Shows a list of functions including `fn.004014a0`.
- Assembly Tab:** Displays assembly code for `fn.004014a0`:


```

      mov eax,dword [var_40h]
      mov byte al, 3
      add ebx, 4
      cmp esl, 0x5000
      jb 0x401780

      push eax
      push 1 ; 17 : SAFEARRAY *psa
      call dword [sym.imp.OLEAUT32.dll_SafeArrayDestroy] ; 0x40e10c ; HRESULT SafeArrayDestroy(SAFEARRAY ...
      mov edi, [eax+40h]
      push edi
      push dword [sym.imp.OLEAUT32.dll_SafeArrayUnlock] ; 0x40e114 ; HRESULT SafeArrayUnlock(SAFEARRAY ...
      mov esl, 0x5000
      mov esp, 4
      push edx
      push 1 ; 17 : SAFEARRAY *psa
      call dword [sym.imp.OLEAUT32.dll_SafeArrayAccessData] ; 0x40e120 ; HRESULT SafeArrayAccessData(SA...
      mov edi, [eax+40h]
      push edi
      push dword [sym.imp.OLEAUT32.dll_SafeArrayGetDesc] ; 0x40e128 ; HRESULT SafeArrayGetDesc(SAFEARR...
      mov esl, 0x5000
      mov esp, 4
      test edi, edi
      jne 0x401776

      push 0x00000403
      call fcn.004014a0

      mov eax,dword [var_28h]
      test max, max
      je 0x401783
      
```
- Imports Tab:** Shows imports from OLEAUT32.dll:

Address	Type	Safety	Name
0x00401200	FUNC		OLEAUT32.dll_SafeArrayAccessData
0x0040120C	FUNC		OLEAUT32.dll_SafeArrayDestroy
0x00401214	FUNC		OLEAUT32.dll_SafeArrayGetDesc
0x00401218	FUNC		OLEAUT32.dll_SafeArrayUnlock
- Registers Tab:** Shows CPU register values.
- Bottom Status Bar:** Address 0x004014a0, Dashboard, Hexdump, Strings, Search, Classes, Vtable, 4 Items, 100%, 2019-08-18T21:08:20+, 100%.

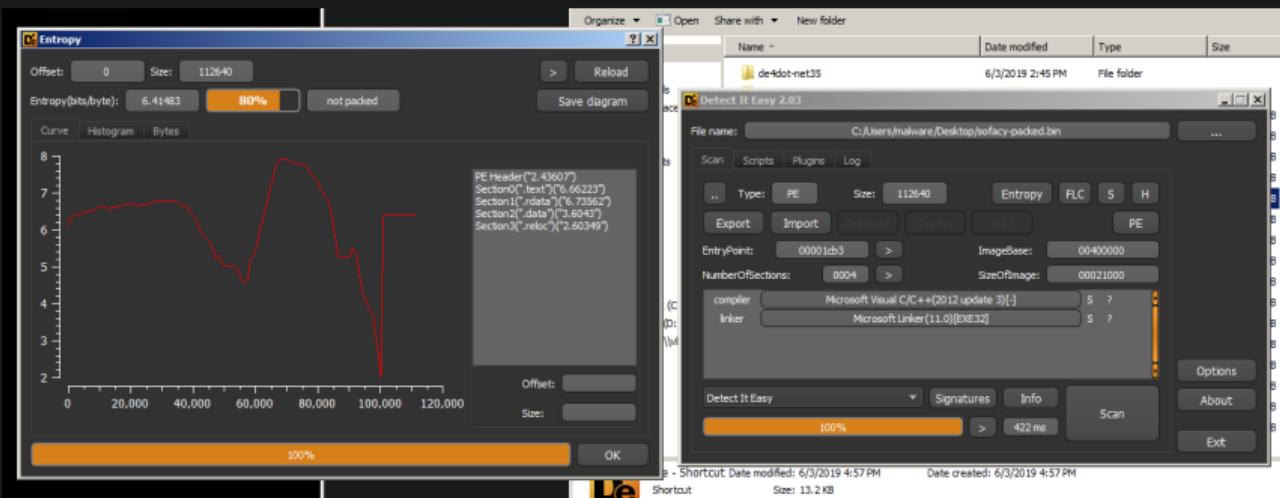
tools_of_the_trade: 0x08

Detect it Easy

tools_of_the_trade: 0x09

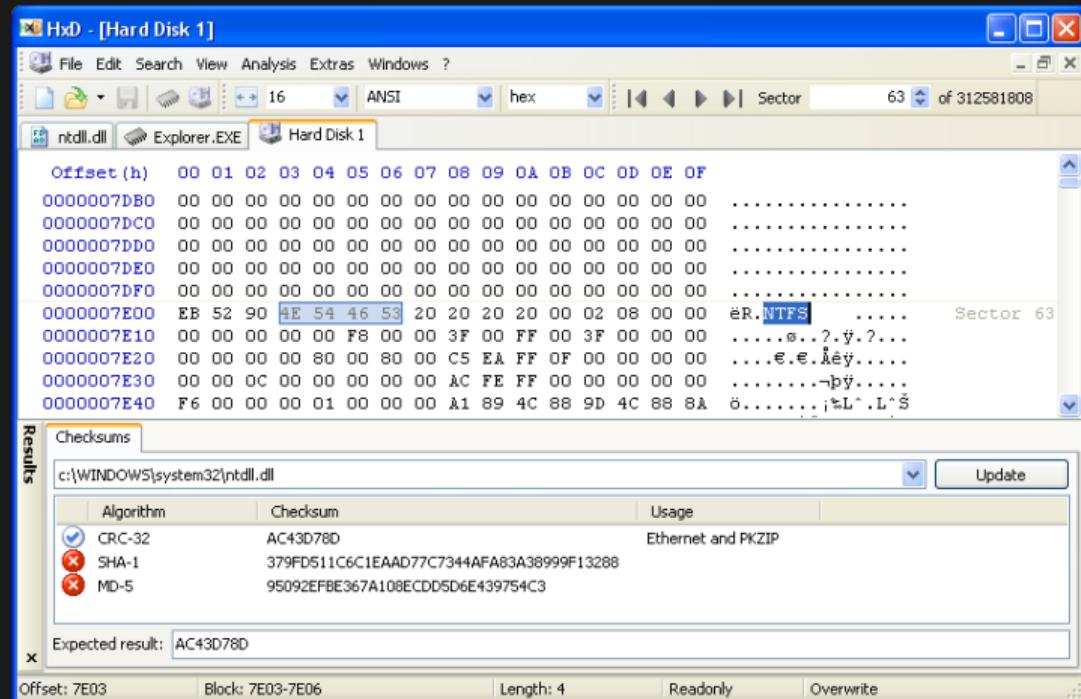
GoSECURE

- Type
- Packer
- Linker
- Entropy



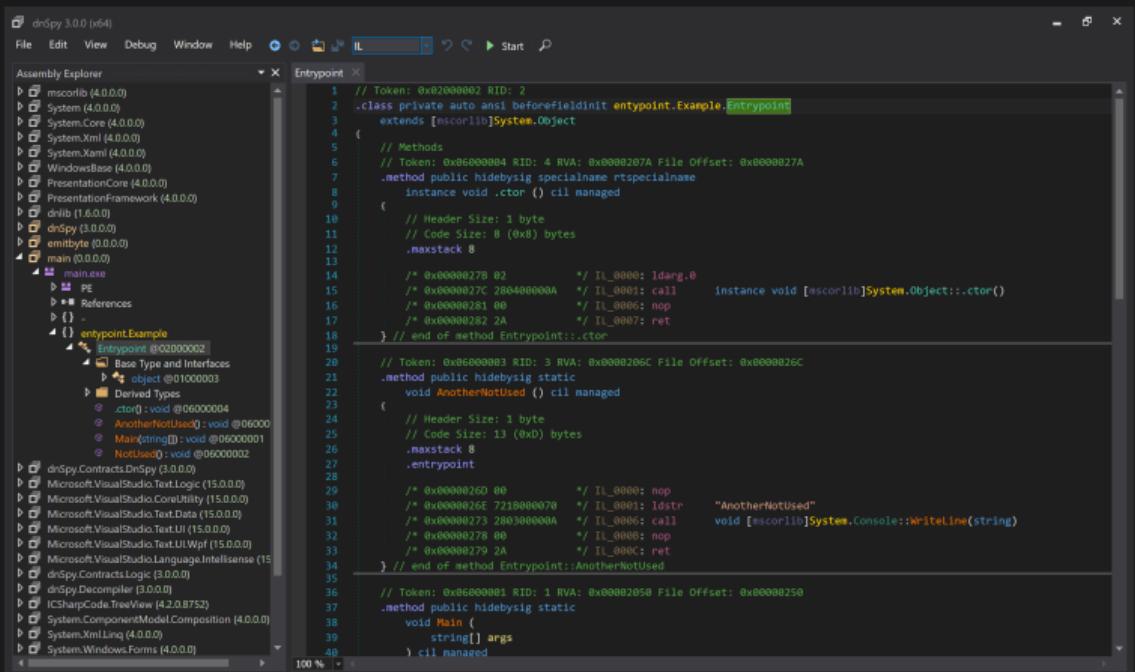
tools_of_the_trade: 0x0a

- 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 IL (Intermediate Language) tabs selected. The Assembly Explorer pane lists various .NET assemblies, including mscorelib, System, System.Core, System.Xml, System.Xaml, WindowsBase, PresentationCore, PresentationFramework, drilb, emitbyte, and the main assembly which contains the Entrypoint.Example class. The IL tab displays the assembly-level IL code:

```
// 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()
    } // end of method EntryPoint::.ctor

    // Token: 0x06000083 RID: 3 RVA: 0x0000209C File Offset: 0x0000029C
    .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: 0x00000258
    .method public hidebysig static
        void Main (
            string[] args
        ) cil managed
    
```

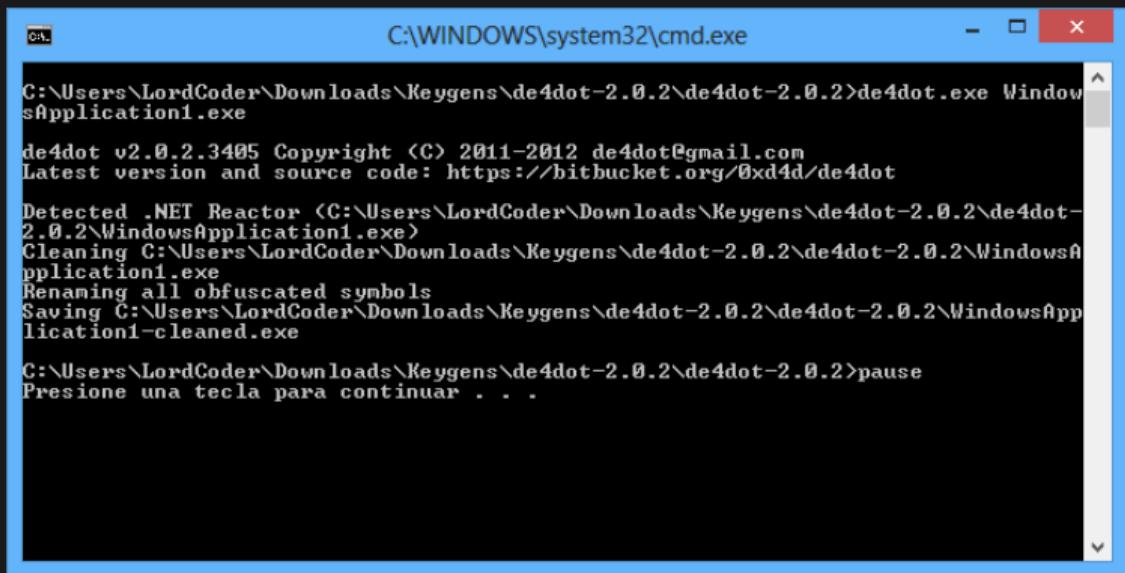
tools_of_the_trade: 0x0c

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)

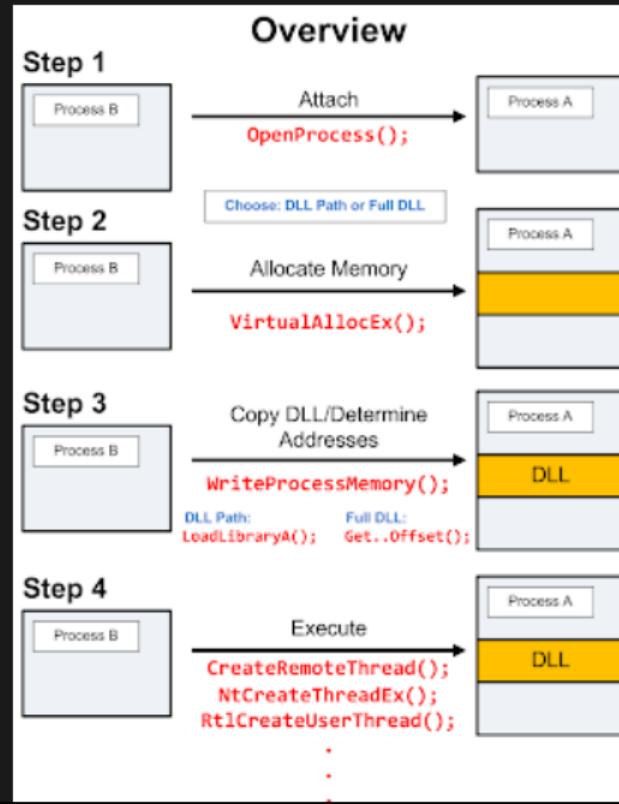
--output options--
/ofilter <ofilter_id>
      : Filter the dumped output.
*ofilter_id:
  0 - no filter: dump everything (default)
  1 - don't dump the modified PEs, but save the report
  2 - don't dump any files
/quiet  : Print only the summary. Do not log on stdout during the scan.
/json   : Print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



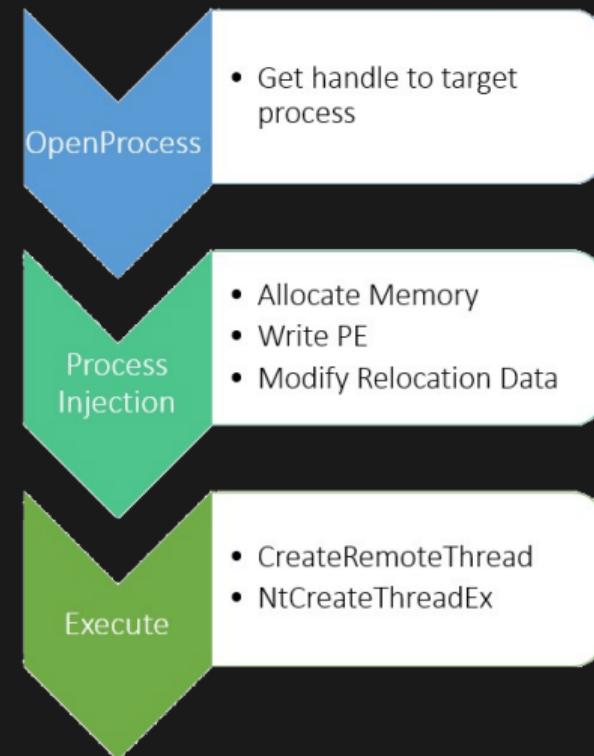
injection_techniques: 0x00

- 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



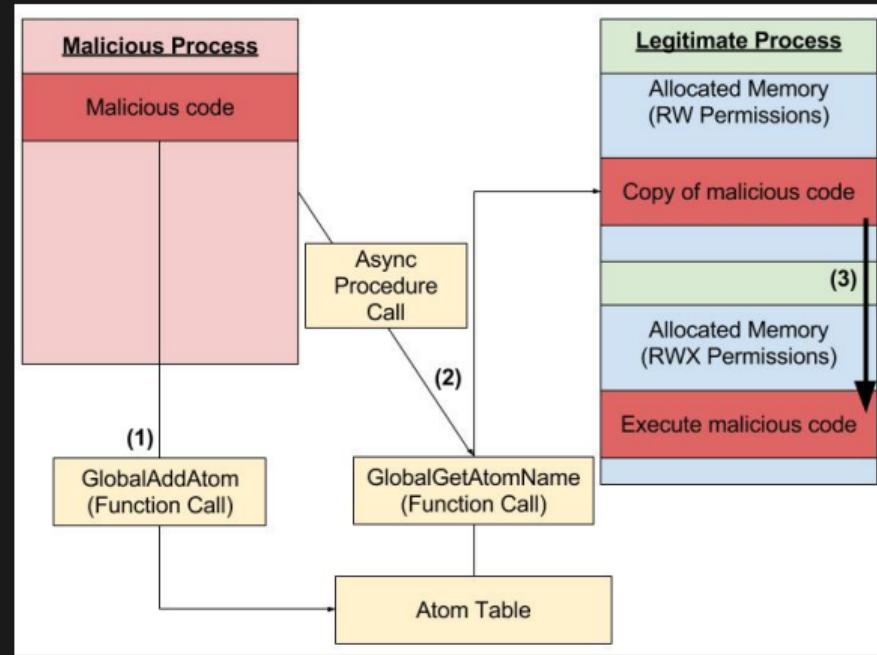


Atomic bomb test in Italy, 1957,
colorized

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



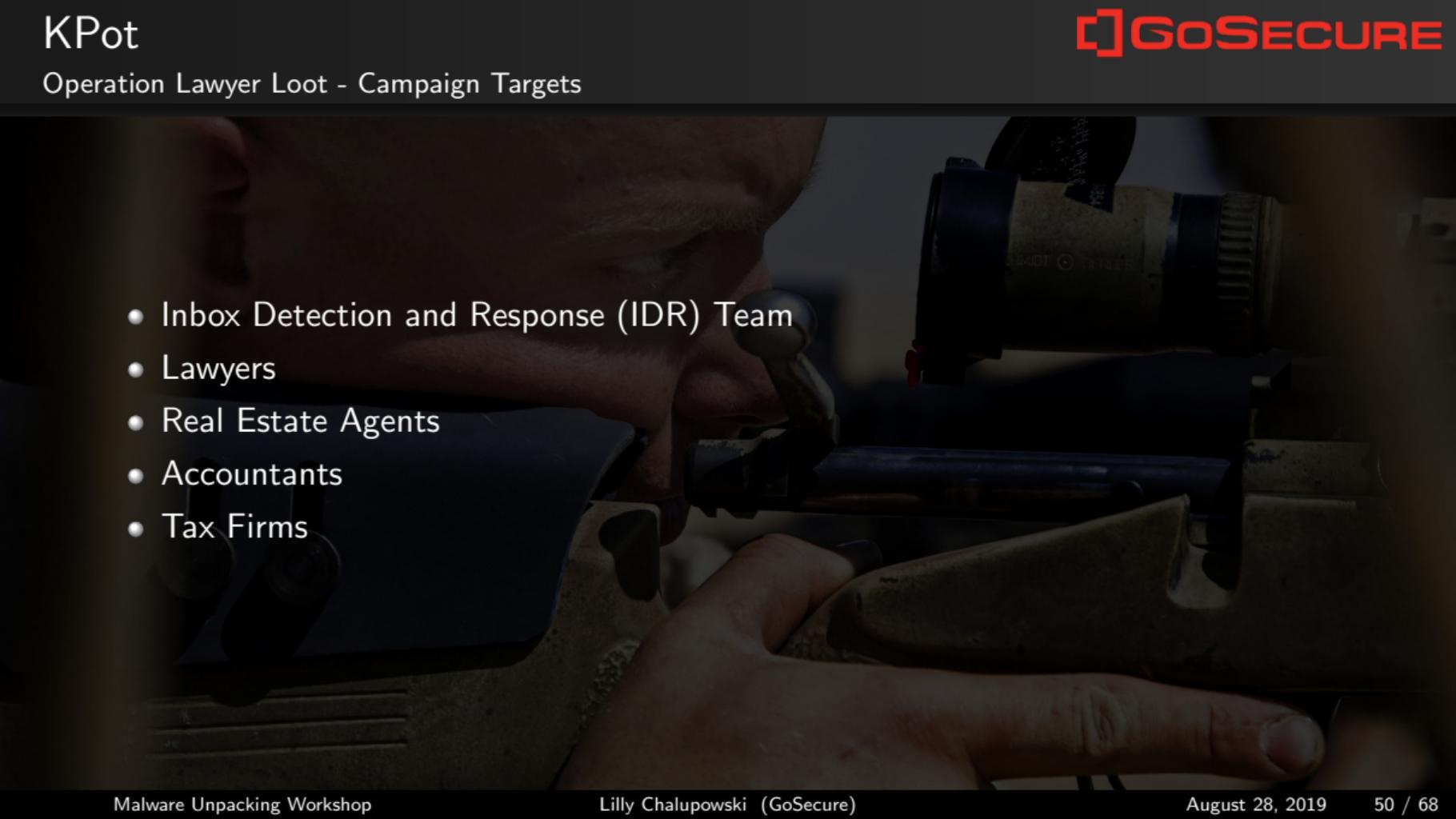
- 5466c52191ddd1564b4680060dc329cb



Initial Infection Vector

MalSpam Email	
From:	< sidney.m[at]carmellaw[.]com >
Subject:	Wire confirmation
Attached:	img-000310519000.img
Body:	<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

- 
- A photograph of a person from the side and slightly behind, wearing a dark hoodie and sunglasses. They are holding a rifle with a scope, looking through it. The background is dark and out of focus.
- 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 ~$
```

Unpacking KPot

unpacking: 0x00

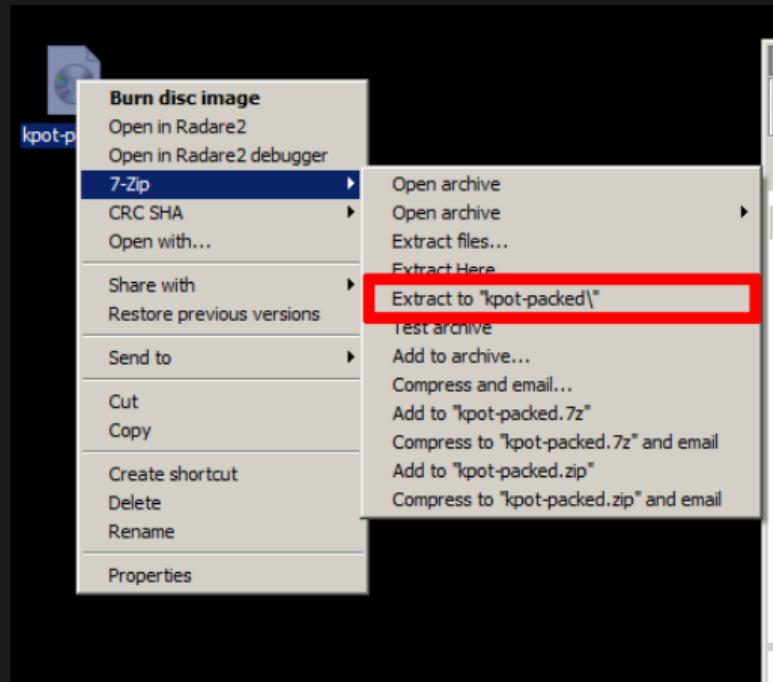
- Looks like this is an ISO
- Let's extract it!

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 1.0000, and the file is identified as ISO 9660. The status bar at the bottom shows the detection engine is "Detect It Easy" with a progress of 100% and a scan time of 125 ms. The application has tabs for Scan, Scripts, Plugins, and Log, and buttons for Options, About, and Exit.

Unpacking KPot

solutions: 0x01

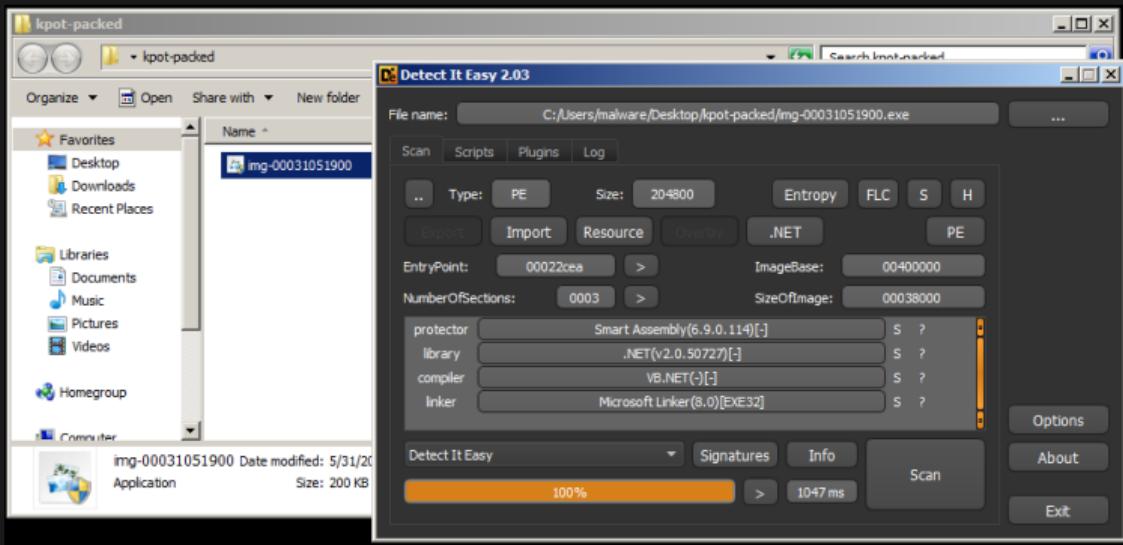
- Extract the ISO Image



Unpacking KPot

unpacking: 0x02

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



Unpacking KPot

unpacking: 0x03

- 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

unpacking: 0x04



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("8HMh1hM12pL90Lp7wYsd2wg==");
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
1368     num2 = 31;
1369     byte[] array2 = Class15.smethod_6(byte_, Class15.smethod_30("Hb7onq2ZgBw+mmuID$SYXZug4//mjETWOU+Mi913jw=="));
1370
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
1379     Class15.smethod_16(new object[])
1380     {
1381         string.Empty,
1382         array2,
1383         false,
1384         false,
1385         Application.ExecutablePath
1386     };
1387
1388     num2 = 35;
```

Get Packed Data

Unpacking Packed Data

Injection Funtion

Unpacking KPot



unpacking: 0x05

- 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

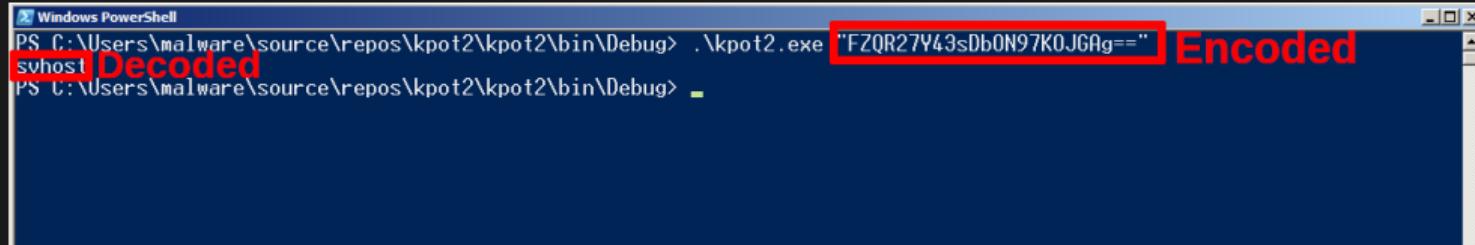
unpacking: 0x07

```
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

unpacking: 0x08



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 string "FZQR27Y43sDb0N97K0JG@g==" is highlighted with a red box and labeled "Encoded". Below the command, the output "svhost" is visible, also highlighted with a red box and labeled "Decoded". The PowerShell window has a blue background.

Figure: Unpacking KPot - String Deobfuscation

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

Unpacking KPot

unpacking: 0x09

```
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["OCVByDSKb4dymgcmPozmijexOC140ik/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+rUjhz+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

unpacking: 0x0A

```
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!

unpacking: 0x0B

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;
}
```

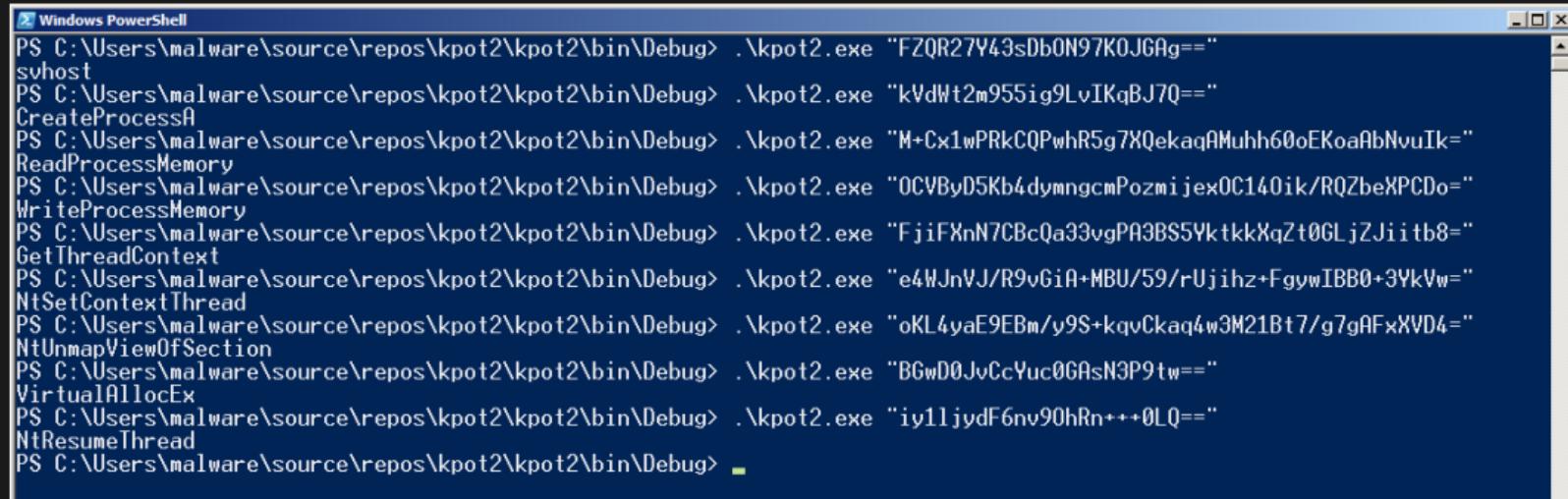
unpacking: 0x0C

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

unpacking: 0x0D



```
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 "0CVByD5Kb4dymngcmPozmijexOC140ik/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

 GoSECURE

unpacking: 0x0E

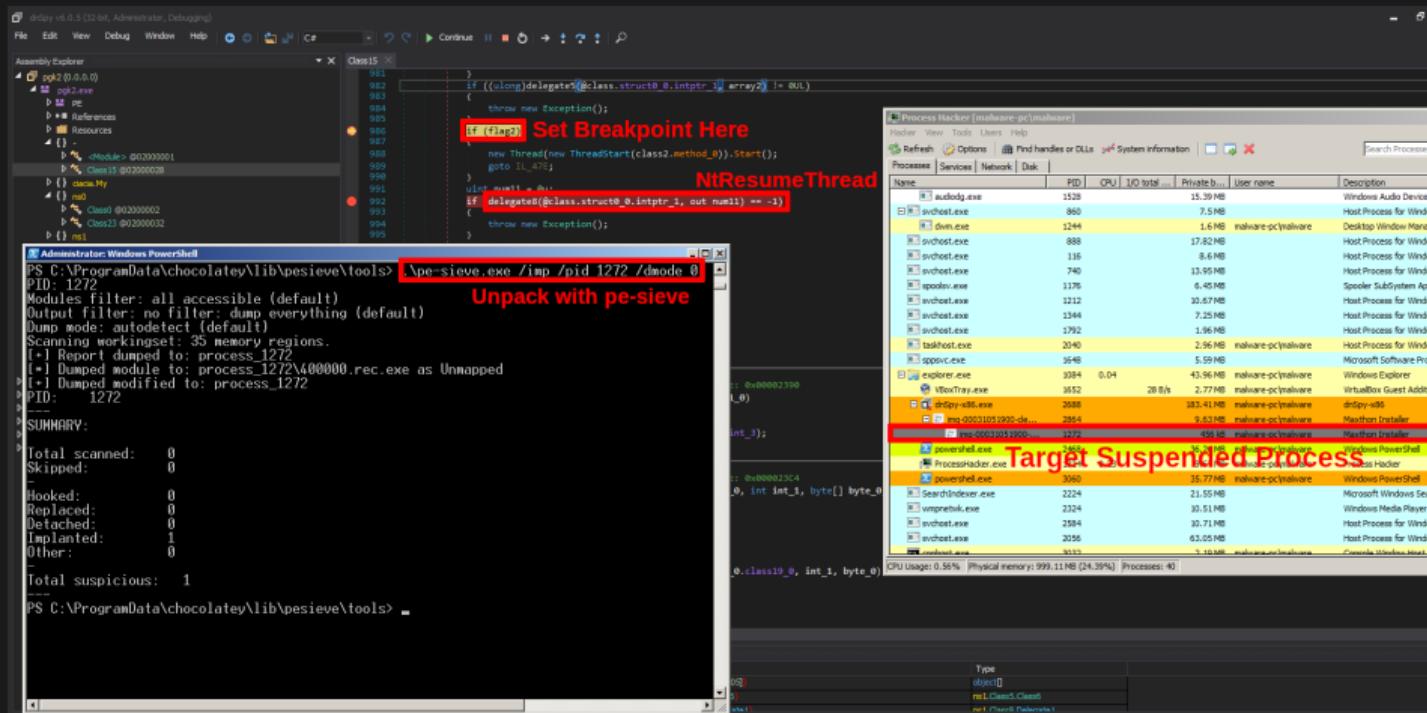


Figure: Unpacking KPot - Now we need to extract it with pe-seive!

Unpacking KPot

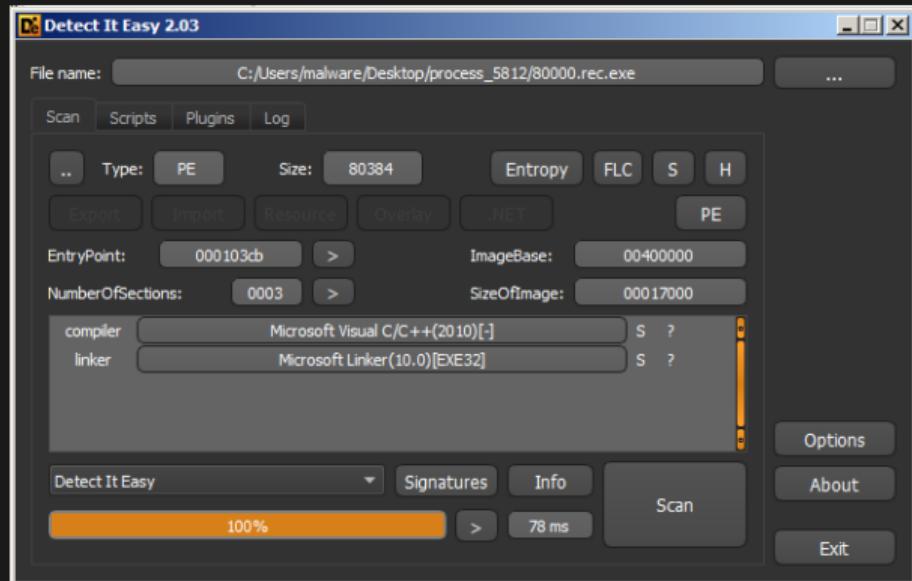
unpacking: 0x0F

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

Unpacking KPot

unpacking: 0x10

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



- 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