How to write malware and learn how to fight it!

Antonio ‘s4tan’ Parata

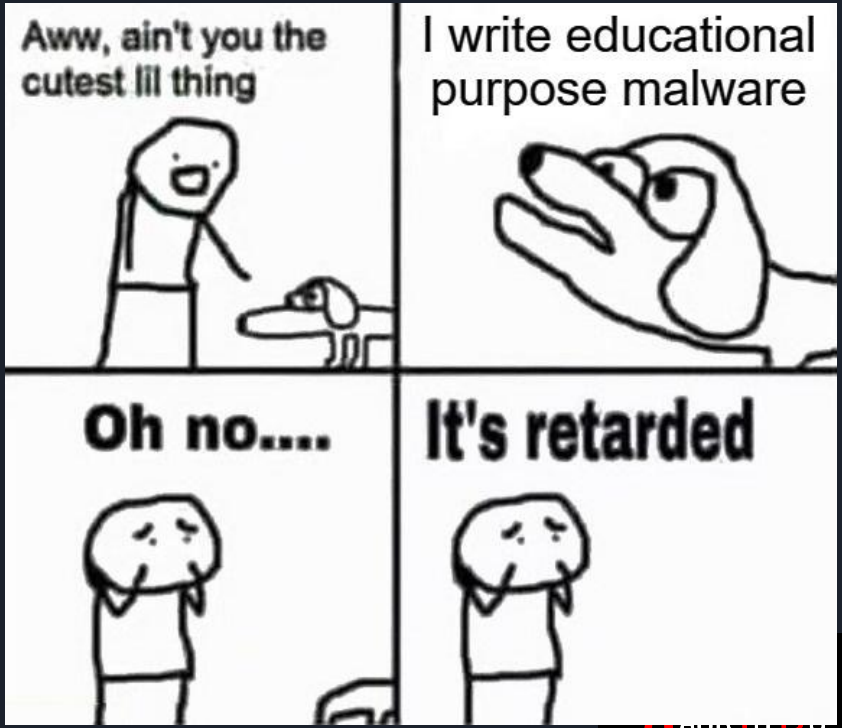
HackInBo® Winter 2019 Edition
13ª EDIZIONE

Disclaimer

This presentation is not intended to teach to the bad guys how to write malware. There are already too many “education purpose projects” in GitHub, we don’t need another one :)

The goal of the presentation is to show how to analyze malicious code by considering how a malware author think.

But remember… CODING IS NOT A CRIME!


Aww, ain't you the cutest lil thing
I write educational purpose malware
Oh no....
It's retarded

Disclaimer

This presentation is not intended to teach the bad guys how to write malware. There are already too many “education purpose projects” in GitHub, we don’t need another one :)

The goal of the presentation is to show how to analyze malicious code by considering how a malware author think.

whoami.exe

We have more Cyber-Security guru on LinkedIn than IPv4 addresses

THE LinkedIn EFFECT

JULIA
Sales

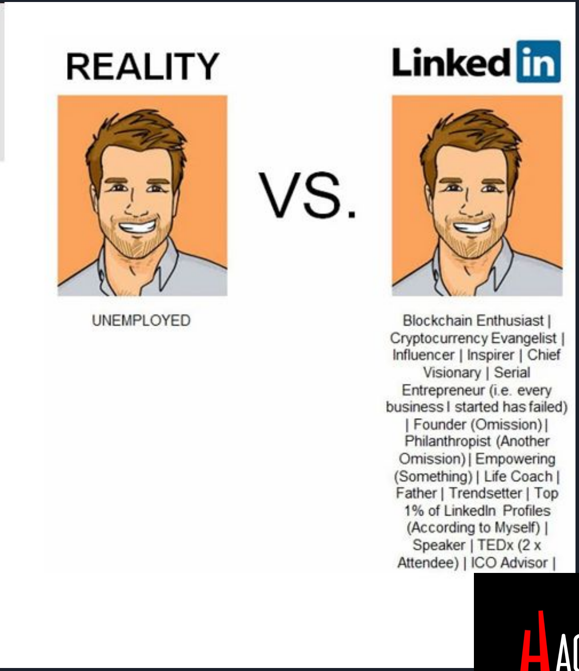
JULIA
Global Sales Team Leader &
Business Developpment Manager
Lead Gen Expert

LEA
HR

LEA
Senior Talent Acquisition Specialist
& Headhunter EMEA

JOHN
Freelance

JOHN
CEO & Board member
Serial Entrepreneur



REALITY

VS.

LinkedIn

UNEMPLOYED

Blockchain Enthusiast | Cryptocurrency Evangelist | Influencer | Inspirer | Chief Visionary | Serial Entrepreneur (i.e. every business I started has failed) | Founder (Omission) | Philanthropist (Another Omission) | Empowering (Something) | Life Coach | Father | Trendsetter | Top 1% of LinkedIn Profiles (According to Myself) | Speaker | TEDx (2 x Attendee) | ICO Advisor |

whoami.exe

01 Fourth time attendee at HackInBo (three as speaker)

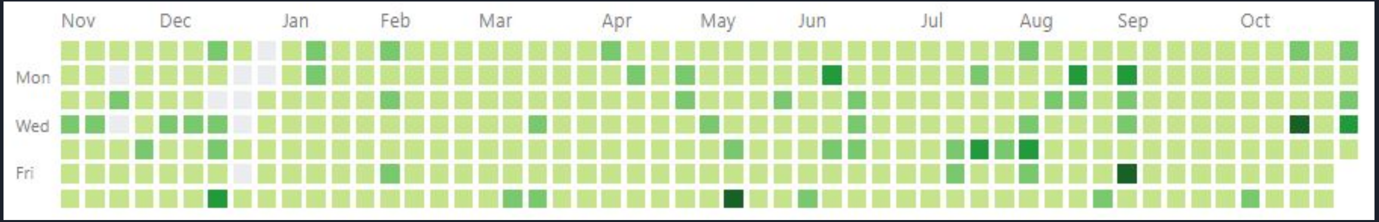
02 Senior Security Researcher CrowdStrike

03 Owasp Italy Board since 2006

04 Phrack Author
http://www.phrack.org/papers/dotnet_instrumentation.html

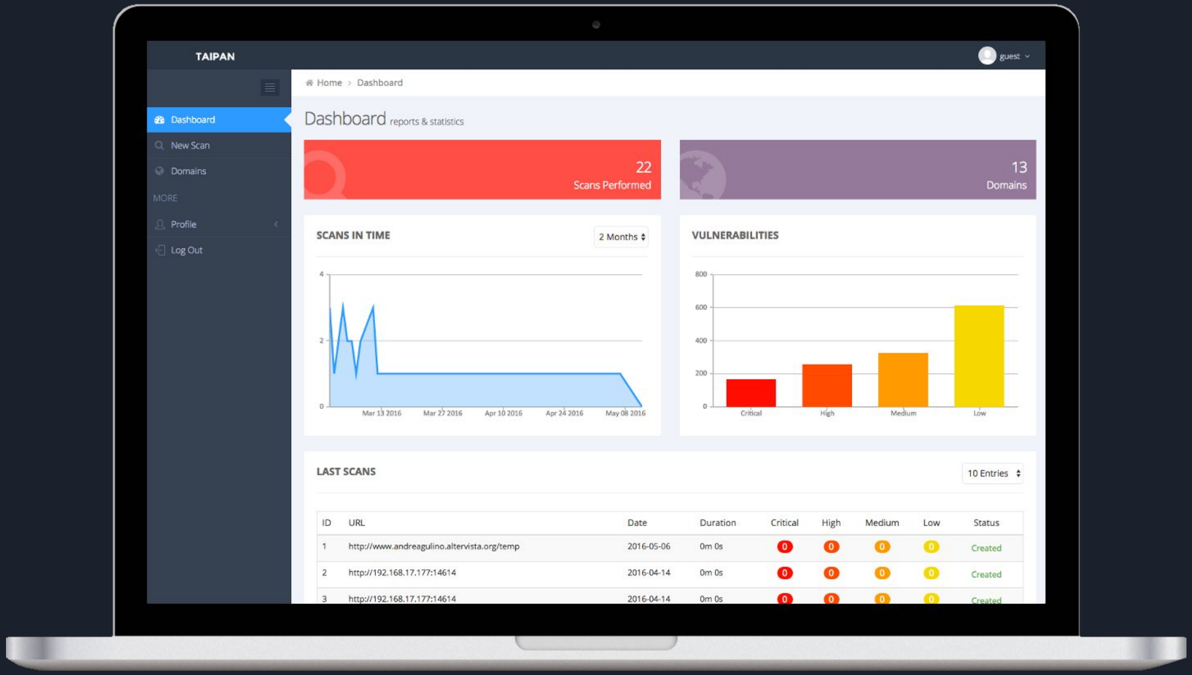
05 Passionate F# developer
https://github.com/sponsors/enkomio ♥ GitHub Sponsors

CROWDSTRIKE



HACKINBO®
Winter 2019 Edition
13ª EDIZIONE

whoami.exe



ID	URL	Date	Duration	Critical	High	Medium	Low	Status
1	http://www.andreagulino.altervista.org/temp	2016-05-06	0m 0s	0	0	0	0	Created
2	http://192.168.17.177:14614	2016-04-14	0m 0s	0	0	0	0	Created
3	http://192.168.17.177:14614	2016-04-14	0m 0s	0	0	0	0	Created

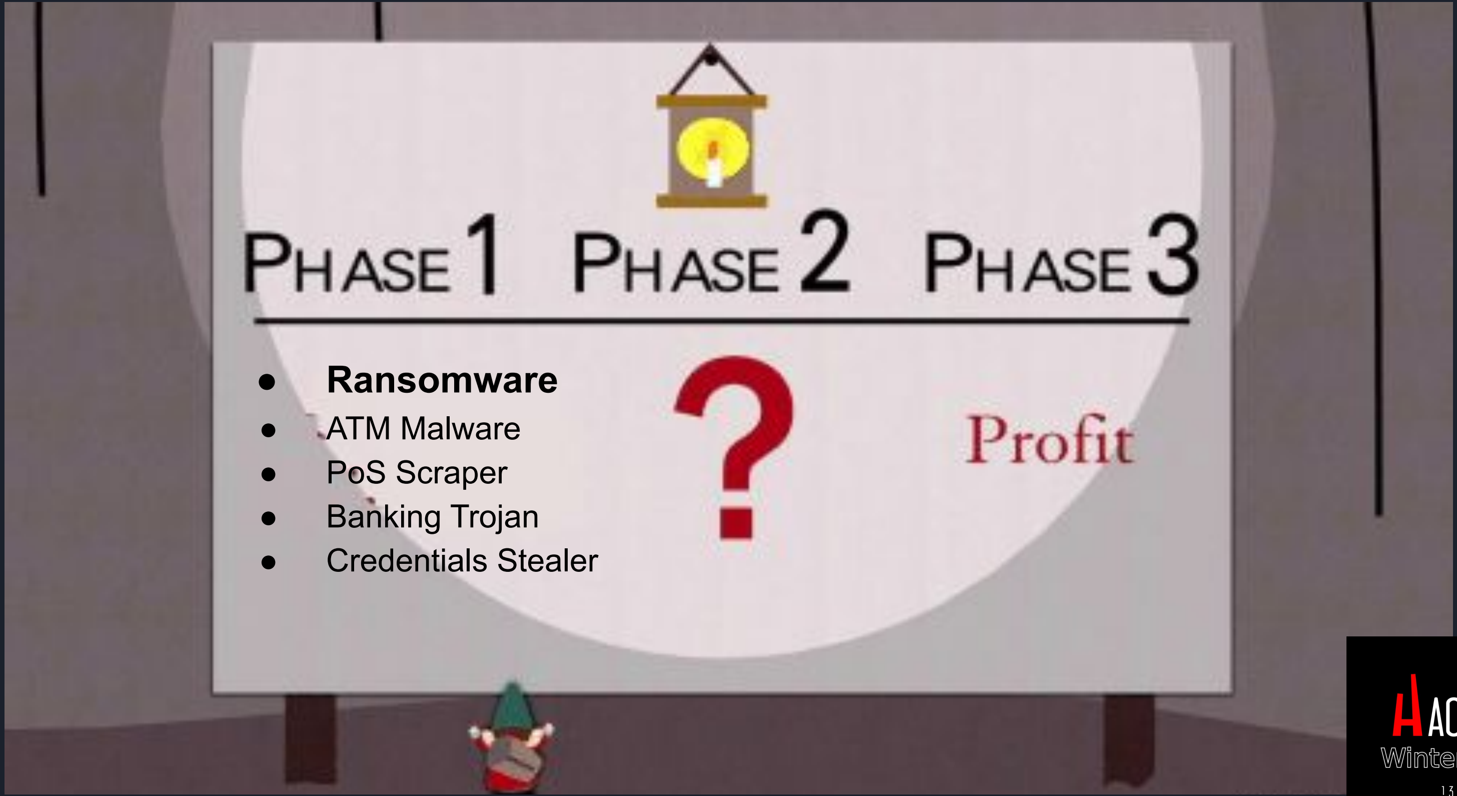
Taipan Web Vulnerability Scanner - https://taipansec.com

HackInBo®
Winter 2019 Edition
13ª EDIZIONE

Cyber-Crime

- We are not talking about amateur malware (skiddies writing a .NET RAT and posting it on HackForums)
- Professional cyber-criminal are very well organized:
  - They have a dedicated GIT repository
  - A testing botnet
  - A customer support platform (typically in form of Jabber chat)
  - A crypto service to evade AVs
  - They use a bulletproof hosting provider for their botnet
  - VPN service to hide his/her real IP
  - A distribution network (SPAM)
  - A mule network (monetization)

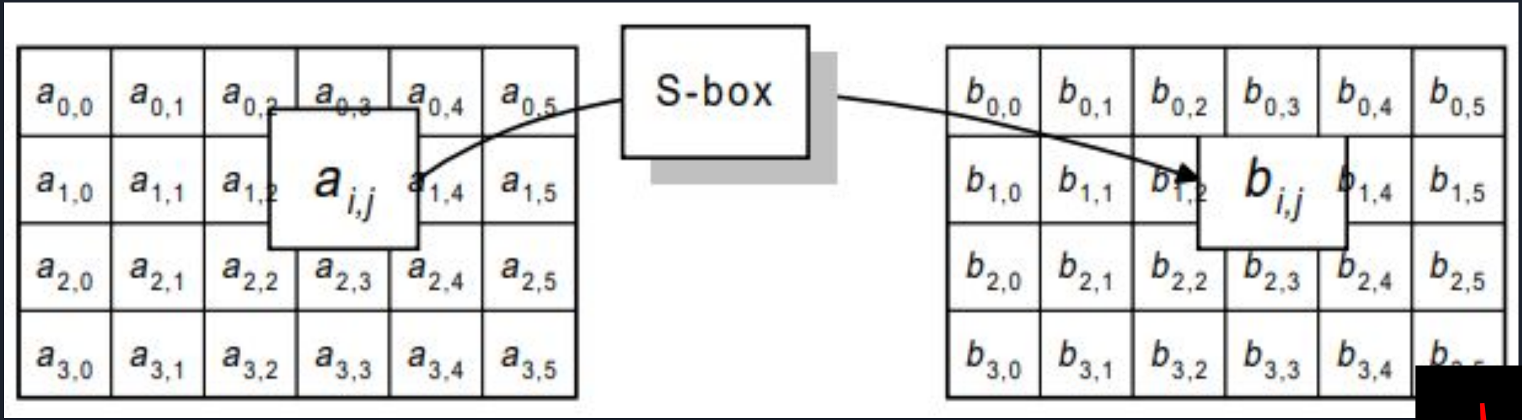
How to write a malware and make money



- **Ransomware**
- ATM Malware
- PoS Scraper
- Banking Trojan
- Credentials Stealer

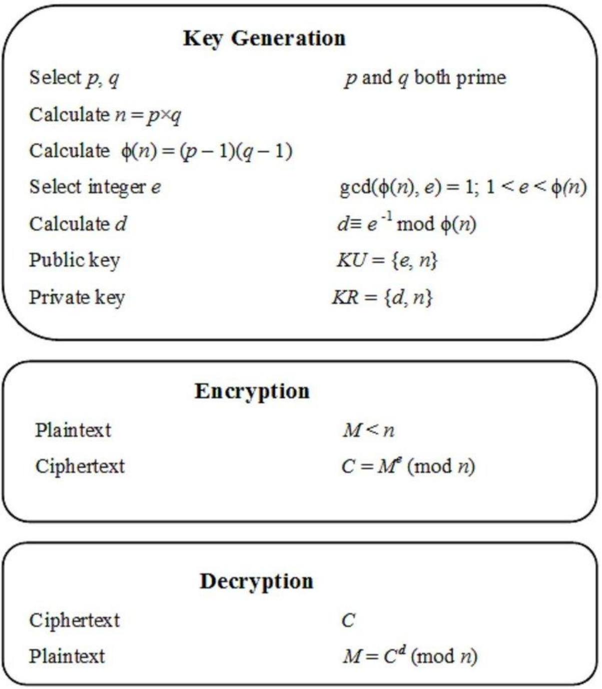
Reversing AES

Pretty easy if S-Box is not obfuscated, just use FindCrypt(2) IDA plugin to identify the code that use the S-Box


$a_{0,0}$ $a_{0,1}$ $a_{0,2}$ $a_{0,3}$ $a_{0,4}$ $a_{0,5}$
$a_{1,0}$ $a_{1,1}$ $a_{1,2}$ $a_{1,4}$ $a_{1,5}$
$a_{2,0}$ $a_{2,1}$ $a_{2,2}$ $a_{2,3}$ $a_{2,4}$ $a_{2,5}$
$a_{3,0}$ $a_{3,1}$ $a_{3,2}$ $a_{3,3}$ $a_{3,4}$ $a_{3,5}$
$a_{i,j}$
S-box
$b_{i,j}$
$b_{0,0}$ $b_{0,1}$ $b_{0,2}$ $b_{0,3}$ $b_{0,4}$ $b_{0,5}$
$b_{1,0}$ $b_{1,1}$ $b_{1,2}$ $b_{1,4}$ $b_{1,5}$
$b_{2,0}$ $b_{2,1}$ $b_{2,2}$ $b_{2,3}$ $b_{2,4}$ $b_{2,5}$
$b_{3,0}$ $b_{3,1}$ $b_{3,2}$ $b_{3,3}$ $b_{3,4}$

Reversing RSA

- **No hard coded constants but...**
- From Wikipedia:
  - *the most commonly chosen value for e is 216 + 1 = 65,537*
- So, if you find very weird math operations involving:
  - Two numbers
  - One of them is very big
  - The other number is 65537 (0x10001)
- Maybe you found an RSA encryption routine!



Key Generation

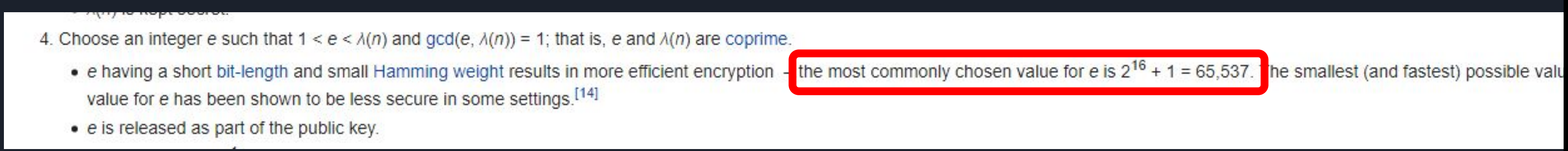
Select p, q	p and q both prime
Calculate $n = p \times q$	
Calculate $\phi(n) = (p-1)(q-1)$	
Select integer e	$\gcd(\phi(n), e) = 1;\ 1 < e < \phi(n)$
Calculate d	$d \equiv e^{-1} \bmod \phi(n)$
Public key	$KU = \{e, n\}$
Private key	$KR = \{d, n\}$

Encryption

Plaintext	$M < n$
Ciphertext	$C = M^e \pmod{n}$

Decryption

Ciphertext	C
Plaintext	$M = C^d \pmod{n}$



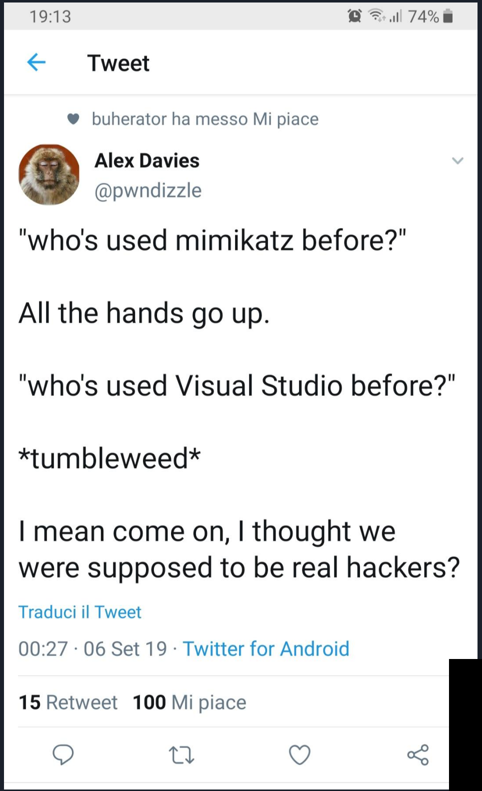
λ(n) is kept secret.

4. Choose an integer e such that $1 < e < \lambda(n)$ and $\gcd(e, \lambda(n)) = 1$; that is, e and $\lambda(n)$ are coprime.
   - e having a short bit-length and small Hamming weight results in more efficient encryption – the most commonly chosen value for e is $2^{16} + 1 = 65{,}537$. The smallest (and fastest) possible valu value for e has been shown to be less secure in some settings.[14]
   - e is released as part of the public key.

Reverse Engineering

What means being a *reverser*?

- Be able to code
- Knowledge about OS
- Knowledge about computer architecture
- Be able to read machine code



19:13 74%

Tweet

buherator ha messo Mi piace

Alex Davies
@pwndizzle

"who's used mimikatz before?"

All the hands go up.

"who's used Visual Studio before?"

tumbleweed

I mean come on, I thought we were supposed to be real hackers?

Traduci il Tweet

00:27 · 06 Set 19 · Twitter for Android

15 Retweet 100 Mi piace

Reversing like a PRO

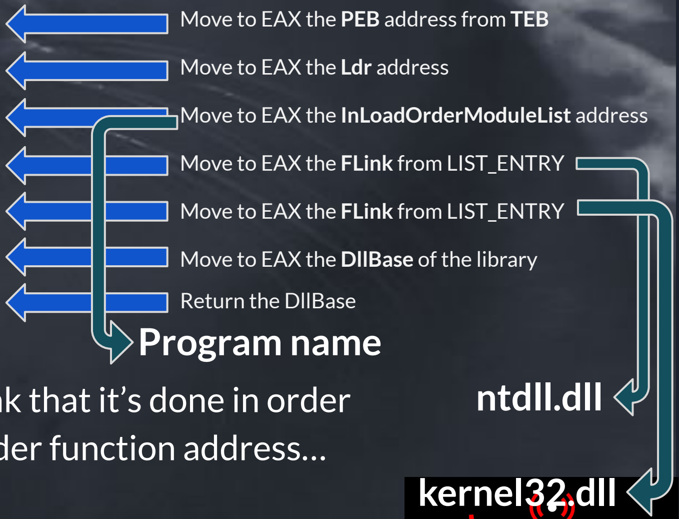
```
00406936 | 64:A1 30000000 | mov eax,dword ptr fs:[30]
0040693C | 8B40 0C        | mov eax,dword ptr ds:[eax+C]
0040693F | 8B40 0C        | mov eax,dword ptr ds:[eax+C]
00406942 | 8B00           | mov eax,dword ptr ds:[eax]
00406944 | 8B00           | mov eax,dword ptr ds:[eax]
00406946 | 8B40 18        | mov eax,dword ptr ds:[eax+18]
00406949 | C3             | ret
```

Move to EAX the value of FS[30]
Move to EAX the value at address EAX+C
Move to EAX the value at address EAX+C
Move to EAX the value at address EAX
Move to EAX the value at address EAX
Move to EAX the value at address EAX + 18
return

C0ngratz u r now an 31337 hax0r!!1

Reversing like a ~~PRO~~ cat

```
00406936 | 64:A1 30000000 | mov eax,dword ptr fs:[30]
0040693C | 8B40 0C        | mov eax,dword ptr ds:[eax+C]
0040693F | 8B40 0C        | mov eax,dword ptr ds:[eax+C]
00406942 | 8B00           | mov eax,dword ptr ds:[eax]
00406944 | 8B00           | mov eax,dword ptr ds:[eax]
00406946 | 8B40 18        | mov eax,dword ptr ds:[eax+18]
00406949 | C3             | ret
```



Move to EAX the **PEB** address from **TEB**

Move to EAX the **Ldr** address

Move to EAX the **InLoadOrderModuleList** address

Move to EAX the **FLink** from LIST_ENTRY

Move to EAX the **FLink** from LIST_ENTRY

Move to EAX the **DllBase** of the library

Return the DllBase

Program name

ntdll.dll

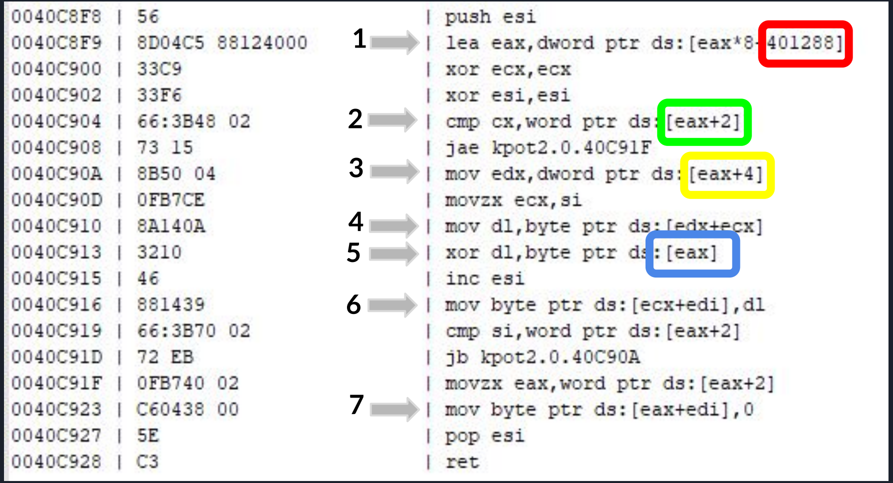
kernel32.dll

This function resolves the base address of Kernel32. If you think that it’s done in order to walk the EAT (Export Address Table) and to resolve the desider function address…

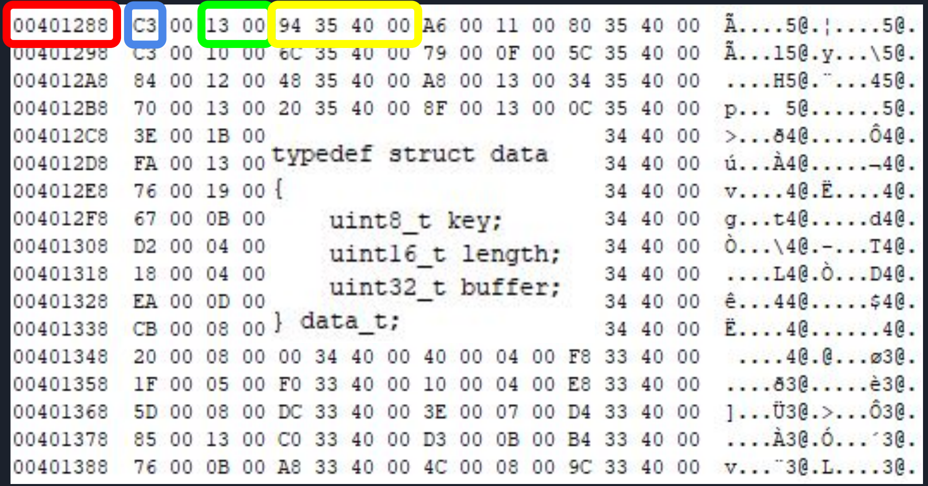
...

you are right! (more soon...)

One more Reversing exercise



```
0040C8F8 | 56                  | push esi
0040C8F9 | 8D04C5 88124000  1 → | lea eax,dword ptr ds:[eax*8+401288]
0040C900 | 33C9                | xor ecx,ecx
0040C902 | 33F6                | xor esi,esi
0040C904 | 66:3B48 02       2 → | cmp cx,word ptr ds:[eax+2]
0040C908 | 73 15               | jae kpot2.0.40C91F
0040C90A | 8B50 04          3 → | mov edx,dword ptr ds:[eax+4]
0040C90D | 0FB7CE              | movzx ecx,si
0040C910 | 8A140A           4 → | mov dl,byte ptr ds:[edx+ecx]
0040C913 | 3210             5 → | xor dl,byte ptr ds:[eax]
0040C915 | 46                  | inc esi
0040C916 | 881439           6 → | mov byte ptr ds:[ecx+edi],dl
0040C919 | 66:3B70 02          | cmp si,word ptr ds:[eax+2]
0040C91D | 72 EB               | jb kpot2.0.40C90A
0040C91F | 0FB740 02           | movzx eax,word ptr ds:[eax+2]
0040C923 | C60438 00        7 → | mov byte ptr ds:[eax+edi],0
0040C927 | 5E                  | pop esi
0040C928 | C3                  | ret
```



```
00401288  C3 00 13 00 94 35 40 00 A6 00 11 00 80 35 40 00  Ã....5@.¦....5@.
00401298  C3 00 10 00 6C 35 40 00 79 00 0F 00 5C 35 40 00  Ã...l5@.y...\5@.
004012A8  84 00 12 00 48 35 40 00 A8 00 13 00 34 35 40 00  ....H5@.¨...45@.
004012B8  70 00 13 00 20 35 40 00 8F 00 13 00 0C 35 40 00  p... 5@......5@.
004012C8  3E 00 1B 00                         34 40 00  >...ð4@.....Ô4@.
004012D8  FA 00 13 00                         34 40 00  ú...À4@.....¬4@.
004012E8  76 00 19 00                         34 40 00  v....4@.Ë....4@.
004012F8  67 00 0B 00                         34 40 00  g...t4@.....d4@.
00401308  D2 00 04 00                         34 40 00  Ò...\4@.-...T4@.
00401318  18 00 04 00                         34 40 00  ....L4@.Ò...D4@.
00401328  EA 00 0D 00                         34 40 00  ê...44@.....$4@.
00401338  CB 00 08 00                         34 40 00  Ë....4@......4@.
00401348  20 00 08 00 00 34 40 00 40 00 04 00 F8 33 40 00   ....4@.@...ø3@.
00401358  1F 00 05 00 F0 33 40 00 10 00 04 00 E8 33 40 00  ....ð3@.....è3@.
00401368  5D 00 08 00 DC 33 40 00 3E 00 07 00 D4 33 40 00  ]...Ü3@.>...Ô3@.
00401378  85 00 13 00 C0 33 40 00 D3 00 0B 00 B4 33 40 00  ....À3@.Ó...´3@.
00401388  76 00 0B 00 A8 33 40 00 4C 00 08 00 9C 33 40 00  v...¨3@.L....3@.
```

```
typedef struct data
{
    uint8_t key;
    uint16_t length;
    uint32_t buffer;
} data_t;
```

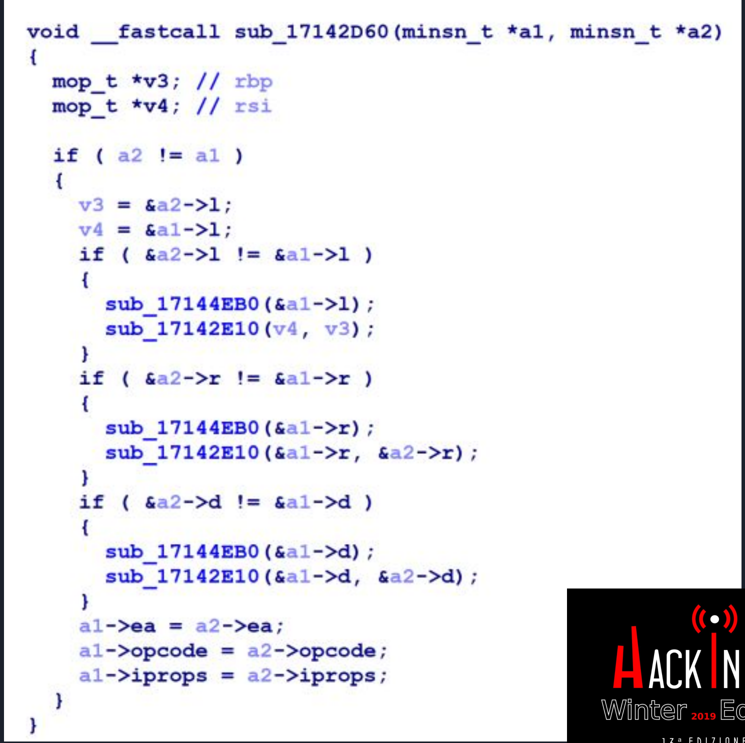
Any idea?

HackInBo®
Winter 2019 Edition
13ª EDIZIONE

Decompiler FTW!

- Decompilers (like Hex-Rays, Ghidra, ILSpy, …) are able to translate machine-code in pseudo code like C or C#.
- This make the RCE task way easier!
- Unfortunately bad guys know this and they use obfuscators or other anti-analysis tricks to avoid decompilation

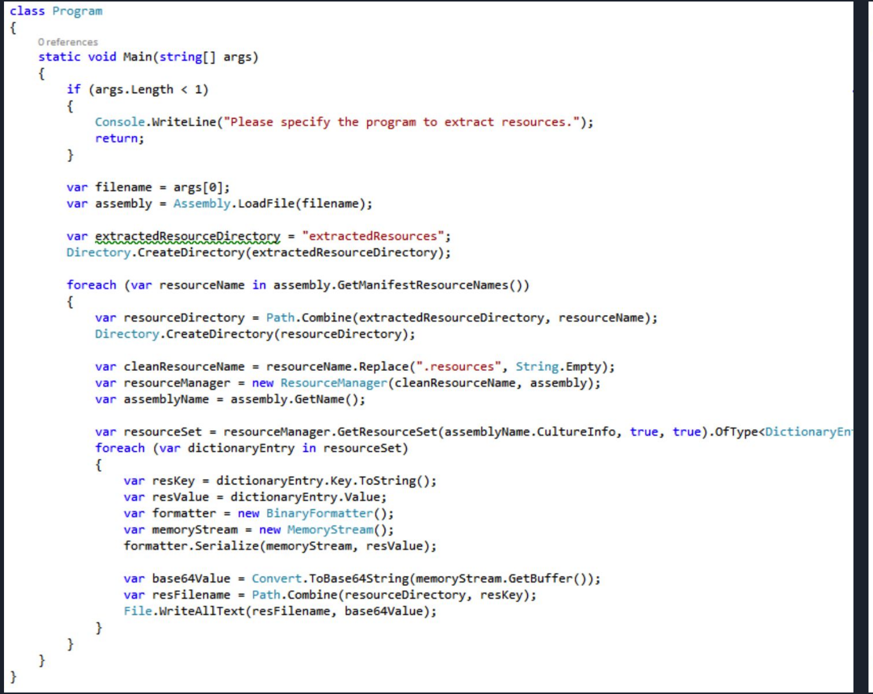
© Rolf Rolles: Automation Techniques in C++ Reverse Engineering



```
void __fastcall sub_17142D60(minsn_t *a1, minsn_t *a2)
{
  mop_t *v3; // rbp
  mop_t *v4; // rsi

  if ( a2 != a1 )
  {
    v3 = &a2->l;
    v4 = &a1->l;
    if ( &a2->l != &a1->l )
    {
      sub_17144EB0(&a1->l);
      sub_17142E10(v4, v3);
    }
    if ( &a2->r != &a1->r )
    {
      sub_17144EB0(&a1->r);
      sub_17142E10(&a1->r, &a2->r);
    }
    if ( &a2->d != &a1->d )
    {
      sub_17144EB0(&a1->d);
      sub_17142E10(&a1->d, &a2->d);
    }
    a1->ea = a2->ea;
    a1->opcode = a2->opcode;
    a1->iprops = a2->iprops;
  }
}
```

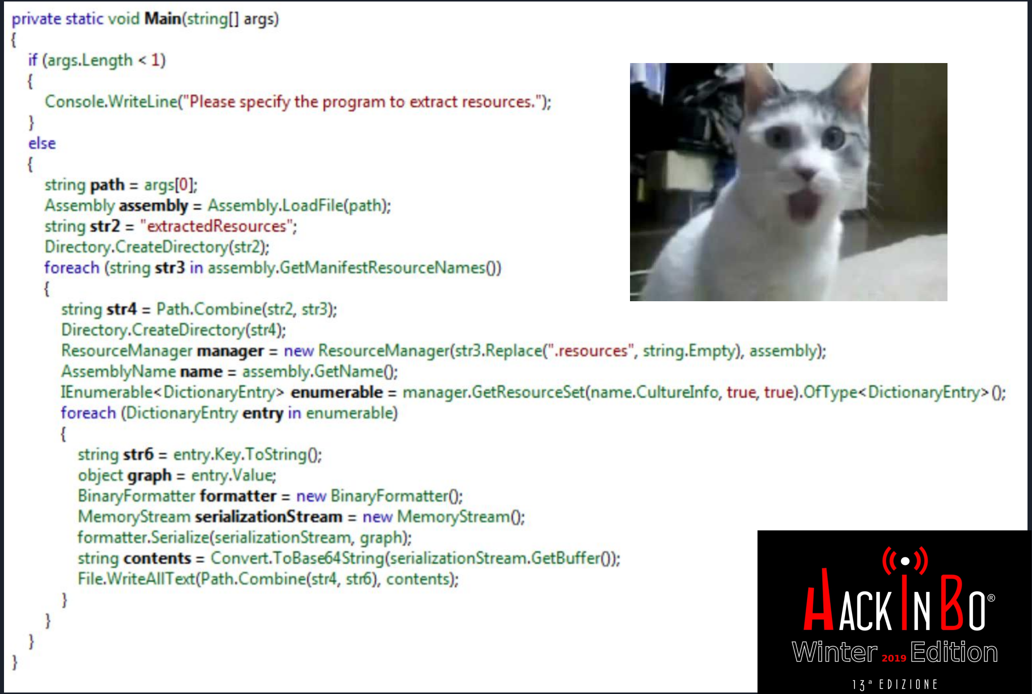
.NET decompilers

Original



```
class Program
{
    0 references
    static void Main(string[] args)
    {
        if (args.Length < 1)
        {
            Console.WriteLine("Please specify the program to extract resources.");
            return;
        }

        var filename = args[0];
        var assembly = Assembly.LoadFile(filename);

        var extractedResourceDirectory = "extractedResources";
        Directory.CreateDirectory(extractedResourceDirectory);

        foreach (var resourceName in assembly.GetManifestResourceNames())
        {
            var resourceDirectory = Path.Combine(extractedResourceDirectory, resourceName);
            Directory.CreateDirectory(resourceDirectory);

            var cleanResourceName = resourceName.Replace(".resources", String.Empty);
            var resourceManager = new ResourceManager(cleanResourceName, assembly);
            var assemblyName = assembly.GetName();

            var resourceSet = resourceManager.GetResourceSet(assemblyName.CultureInfo, true, true).OfType<DictionaryEn
            foreach (var dictionaryEntry in resourceSet)
            {
                var resKey = dictionaryEntry.Key.ToString();
                var resValue = dictionaryEntry.Value;
                var formatter = new BinaryFormatter();
                var memoryStream = new MemoryStream();
                formatter.Serialize(memoryStream, resValue);

                var base64Value = Convert.ToBase64String(memoryStream.GetBuffer());
                var resFilename = Path.Combine(resourceDirectory, resKey);
                File.WriteAllText(resFilename, base64Value);
            }
        }
    }
}
```

Decompiled



```
private static void Main(string[] args)
{
  if (args.Length < 1)
  {
    Console.WriteLine("Please specify the program to extract resources.");
  }
  else
  {
    string path = args[0];
    Assembly assembly = Assembly.LoadFile(path);
    string str2 = "extractedResources";
    Directory.CreateDirectory(str2);
    foreach (string str3 in assembly.GetManifestResourceNames())
    {
      string str4 = Path.Combine(str2, str3);
      Directory.CreateDirectory(str4);
      ResourceManager manager = new ResourceManager(str3.Replace(".resources", string.Empty), assembly);
      AssemblyName name = assembly.GetName();
      IEnumerable<DictionaryEntry> enumerable = manager.GetResourceSet(name.CultureInfo, true, true).OfType<DictionaryEntry>();
      foreach (DictionaryEntry entry in enumerable)
      {
        string str6 = entry.Key.ToString();
        object graph = entry.Value;
        BinaryFormatter formatter = new BinaryFormatter();
        MemoryStream serializationStream = new MemoryStream();
        formatter.Serialize(serializationStream, graph);
        string contents = Convert.ToBase64String(serializationStream.GetBuffer());
        File.WriteAllText(Path.Combine(str4, str6), contents);
      }
    }
  }
}
```

Breaking .NET decompilers

IL_0014: nop

IL_0015: ldarg.0 // pointer to ***this*** argument, this value is expected by instance methods

IL_0016: call instance void ConsoleApplication.SimpleClass::SayHello()

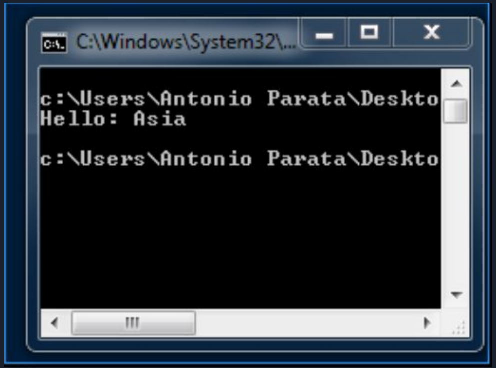
Assemble

IL_0014: br.s IL_0017

~~IL_0015: ldarg.0~~ // remove the push of the ***this*** argument and add a jump in order to avoid the call

IL_0016: call instance void ConsoleApplication.SimpleClass::SayHello()

IL_0017: nop

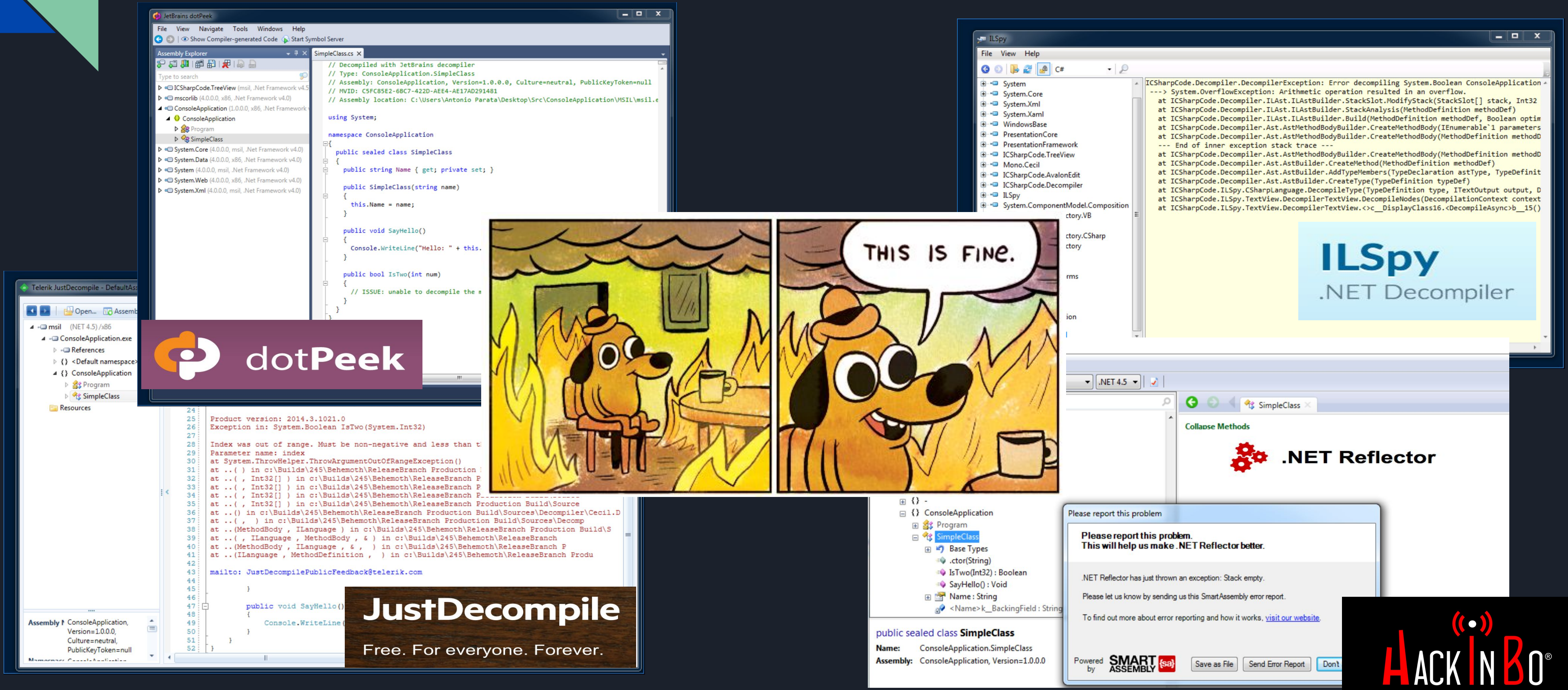


```
c:\Users\Antonio Parata\Deskto
Hello: Asia

c:\Users\Antonio Parata\Deskto
```

Decompile

HACK IN BO®
Winter 2019 Edition
13ª EDIZIONE

Breaking .NET decompilers



```
// Decompiled with JetBrains decompiler
// Type: ConsoleApplication.SimpleClass
// Assembly: ConsoleApplication, Version=1.0.0.0, Culture=neutral, PublicKeyToken=null
// MVID: C5FC85E2-6BC7-422D-AEE4-AE17AD291481
// Assembly location: C:\Users\Antonio Parata\Desktop\Src\ConsoleApplication\MSIL\msil.e

using System;

namespace ConsoleApplication
{
  public sealed class SimpleClass
  {
    public string Name { get; private set; }

    public SimpleClass(string name)
    {
      this.Name = name;
    }

    public void SayHello()
    {
      Console.WriteLine("Hello: " + this.
    }

    public bool IsTwo(int num)
    {
      // ISSUE: unable to decompile the m
    }
  }
}
```

```
ICSharpCode.Decompiler.DecompilerException: Error decompiling System.Boolean ConsoleApplication
 ---> System.OverflowException: Arithmetic operation resulted in an overflow.
   at ICSharpCode.Decompiler.ILAst.ILAstBuilder.StackSlot.ModifyStack(StackSlot[] stack, Int32
   at ICSharpCode.Decompiler.ILAst.ILAstBuilder.StackAnalysis(MethodDefinition methodDef)
   at ICSharpCode.Decompiler.ILAst.ILAstBuilder.Build(MethodDefinition methodDef, Boolean optim
   at ICSharpCode.Decompiler.Ast.AstMethodBodyBuilder.CreateMethodBody(IEnumerable`1 parameters
   at ICSharpCode.Decompiler.Ast.AstMethodBodyBuilder.CreateMethodBody(MethodDefinition methodD
   --- End of inner exception stack trace ---
   at ICSharpCode.Decompiler.Ast.AstMethodBodyBuilder.CreateMethodBody(MethodDefinition methodD
   at ICSharpCode.Decompiler.Ast.AstBuilder.CreateMethod(MethodDefinition methodDef)
   at ICSharpCode.Decompiler.Ast.AstBuilder.AddTypeMembers(TypeDeclaration astType, TypeDefinit
   at ICSharpCode.Decompiler.Ast.AstBuilder.CreateType(TypeDefinition typeDef)
   at ICSharpCode.ILSpy.CSharpLanguage.DecompileType(TypeDefinition type, ITextOutput output, D
   at ICSharpCode.ILSpy.TextView.DecompilerTextView.DecompileNodes(DecompilationContext context
   at ICSharpCode.ILSpy.TextView.DecompilerTextView.<>c__DisplayClass16.<DecompileAsync>b__15()
```

```

Product version: 2014.3.1021.0
Exception in: System.Boolean IsTwo(System.Int32)

Index was out of range. Must be non-negative and less than t
Parameter name: index
at System.ThrowHelper.ThrowArgumentOutOfRangeException()
at ..( ) in c:\Builds\245\Behemoth\ReleaseBranch Production
at ..( , Int32[] ) in c:\Builds\245\Behemoth\ReleaseBranch P
at ..( , Int32[] ) in c:\Builds\245\Behemoth\ReleaseBranch P
at ..( , Int32[] ) in c:\Builds\245\Behemoth\ReleaseBranch P
at ..( , Int32[] ) in c:\Builds\245\Behemoth\ReleaseBranch Production Build\Source
at ..() in c:\Builds\245\Behemoth\ReleaseBranch Production Build\Sources\Decompiler\Cecil.D
at ..( ,  ) in c:\Builds\245\Behemoth\ReleaseBranch Production Build\Sources\Decomp
at ..(MethodBody , ILanguage ) in c:\Builds\245\Behemoth\ReleaseBranch Production Build\S
at ..( , ILanguage , MethodBody , & ) in c:\Builds\245\Behemoth\ReleaseBranch
at ..(MethodBody , ILanguage , & ,  ) in c:\Builds\245\Behemoth\ReleaseBranch P
at ..(ILanguage , MethodDefinition ,  ) in c:\Builds\245\Behemoth\ReleaseBranch Produ

mailto: JustDecompilePublicFeedback@telerik.com

        }

        public void SayHello()
        {
            Console.WriteLine(
        }
    }
}
```

Please report this problem

Please report this problem.
This will help us make .NET Reflector better.

.NET Reflector has just thrown an exception: Stack empty.

Please let us know by sending us this SmartAssembly error report.

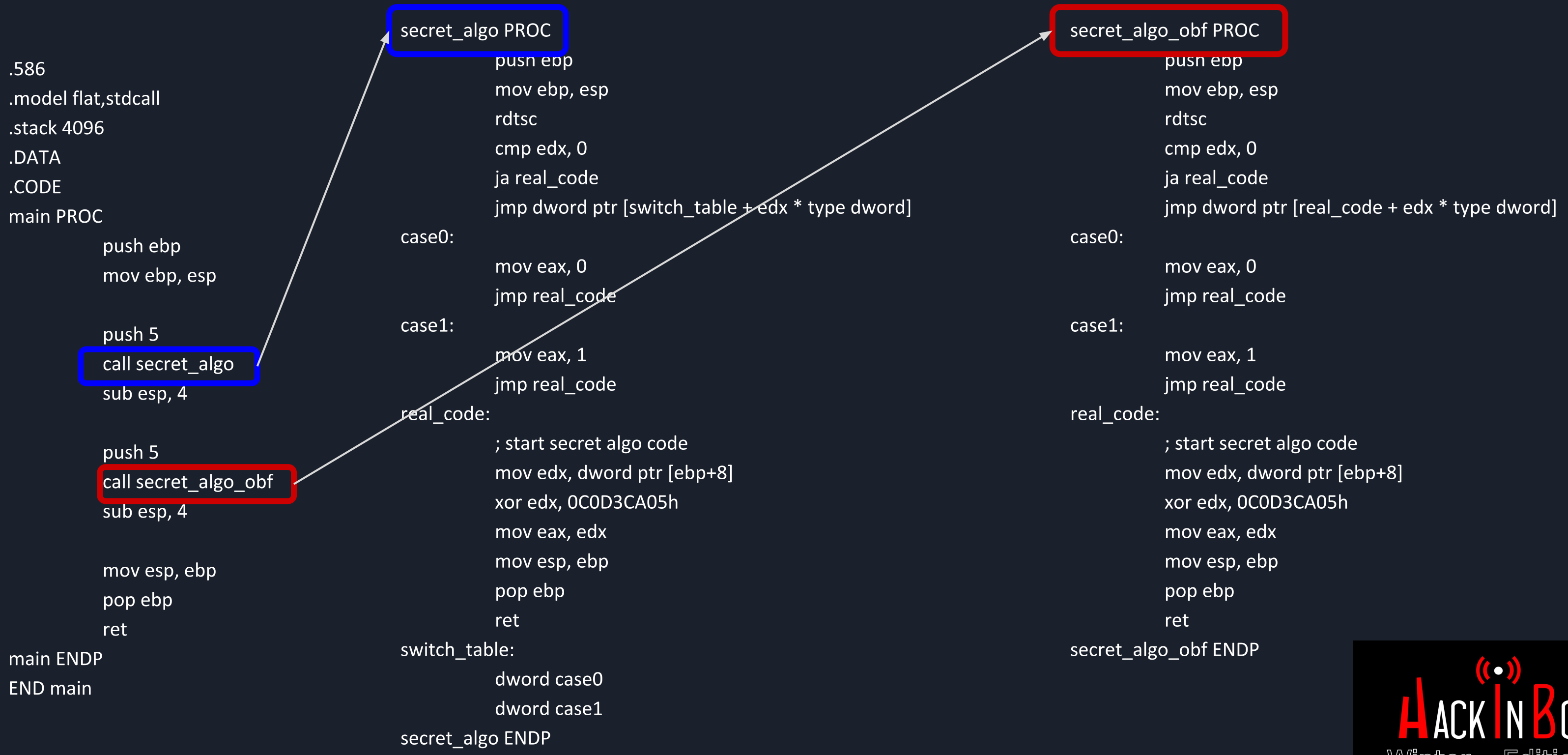
To find out more about error reporting and how it works, visit our website.

Powered by SMART ASSEMBLY {sa}

public sealed class SimpleClass
Name: ConsoleApplication.SimpleClass
Assembly: ConsoleApplication, Version=1.0.0.0

I did this test some time ago, the decompilers may have fixed this problem in the meantime

Anti-analysis - IDA Hex-Rays decompiler



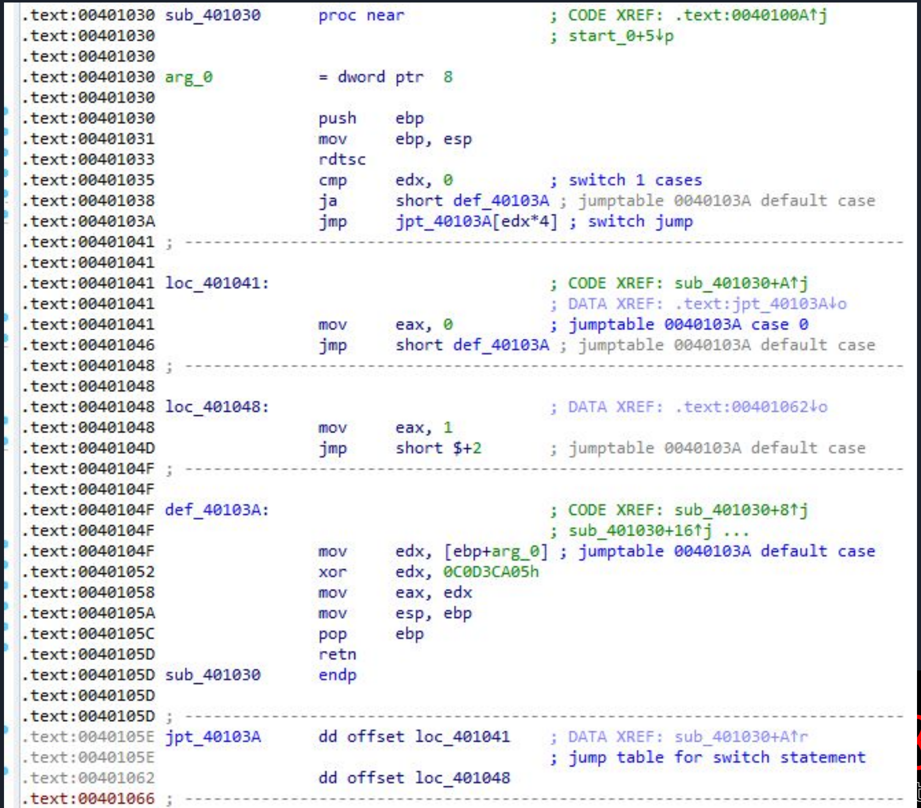
```
.586
.model flat,stdcall
.stack 4096
.DATA
.CODE
main PROC
        push ebp
        mov ebp, esp

        push 5
        call secret_algo
        sub esp, 4

        push 5
        call secret_algo_obf
        sub esp, 4

        mov esp, ebp
        pop ebp
        ret
main ENDP
END main
```

```
secret_algo PROC
        push ebp
        mov ebp, esp
        rdtsc
        cmp edx, 0
        ja real_code
        jmp dword ptr [switch_table + edx * type dword]
case0:
        mov eax, 0
        jmp real_code
case1:
        mov eax, 1
        jmp real_code
real_code:
        ; start secret algo code
        mov edx, dword ptr [ebp+8]
        xor edx, 0C0D3CA05h
        mov eax, edx
        mov esp, ebp
        pop ebp
        ret
switch_table:
        dword case0
        dword case1
secret_algo ENDP
```

```
secret_algo_obf PROC
        push ebp
        mov ebp, esp
        rdtsc
        cmp edx, 0
        ja real_code
        jmp dword ptr [real_code + edx * type dword]
case0:
        mov eax, 0
        jmp real_code
case1:
        mov eax, 1
        jmp real_code
real_code:
        ; start secret algo code
        mov edx, dword ptr [ebp+8]
        xor edx, 0C0D3CA05h
        mov eax, edx
        mov esp, ebp
        pop ebp
        ret
secret_algo_obf ENDP
```

Anti-analysis - IDA Hex-Rays decompiler

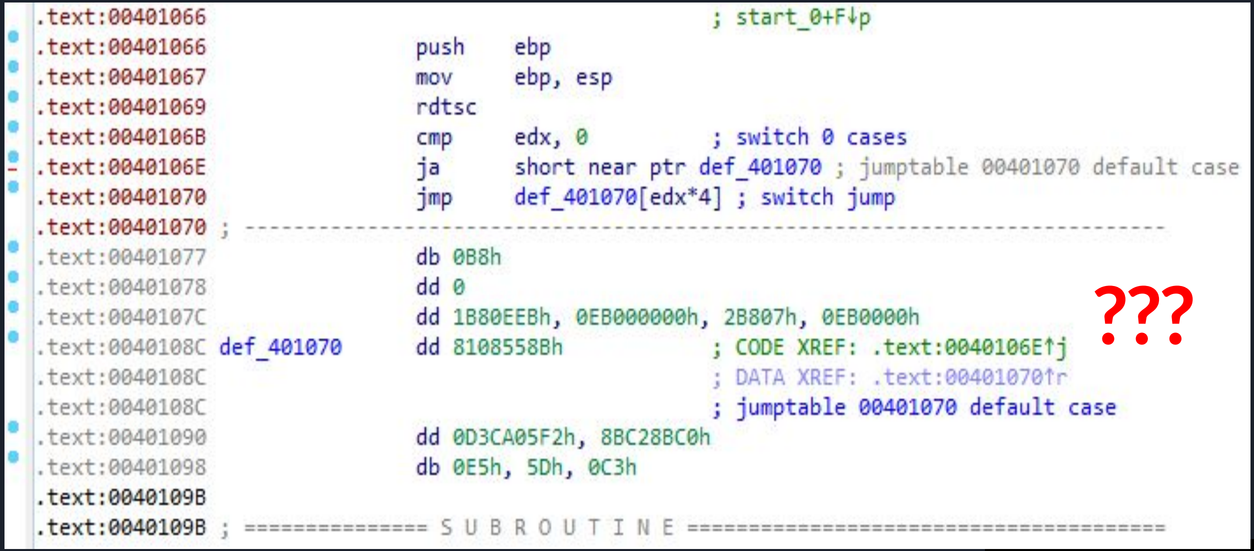
```
secret_algo PROC
        push ebp
        mov ebp, esp
        rdtsc
        cmp edx, 0
        ja real_code
        jmp dword ptr [switch_table + edx * type dword]
case0:
        mov eax, 0
        jmp real_code
case1:
        mov eax, 1
        jmp real_code
real_code:
        ; start secret algo code
        mov edx, dword ptr [ebp+8]
        xor edx, 0C0D3CA05h
        mov eax, edx
        mov esp, ebp
        pop ebp
        ret
switch_table:
        dword case0
        dword case1
secret_algo ENDP
```



```
.text:00401030 sub_401030      proc near               ; CODE XREF: .text:0040100A↑j
.text:00401030                                         ; start_0+5↓p
.text:00401030
.text:00401030 arg_0           = dword ptr  8
.text:00401030
.text:00401030                 push    ebp
.text:00401031                 mov     ebp, esp
.text:00401033                 rdtsc
.text:00401035                 cmp     edx, 0          ; switch 1 cases
.text:00401038                 ja      short def_40103A ; jumptable 0040103A default case
.text:0040103A                 jmp     jpt_40103A[edx*4] ; switch jump
.text:00401041 ; ---------------------------------------------------------------------------
.text:00401041
.text:00401041 loc_401041:                             ; CODE XREF: sub_401030+A↑j
.text:00401041                                         ; DATA XREF: .text:jpt_40103A↓o
.text:00401041                 mov     eax, 0          ; jumptable 0040103A case 0
.text:00401046                 jmp     short def_40103A ; jumptable 0040103A default case
.text:00401048 ; ---------------------------------------------------------------------------
.text:00401048
.text:00401048 loc_401048:                             ; DATA XREF: .text:00401062↓o
.text:00401048                 mov     eax, 1
.text:0040104D                 jmp     short $+2       ; jumptable 0040103A default case
.text:0040104F ; ---------------------------------------------------------------------------
.text:0040104F
.text:0040104F def_40103A:                             ; CODE XREF: sub_401030+8↑j
.text:0040104F                                         ; sub_401030+16↑j ...
.text:0040104F                 mov     edx, [ebp+arg_0] ; jumptable 0040103A default case
.text:00401052                 xor     edx, 0C0D3CA05h
.text:00401058                 mov     eax, edx
.text:0040105A                 mov     esp, ebp
.text:0040105C                 pop     ebp
.text:0040105D                 retn
.text:0040105D sub_401030      endp
.text:0040105D
.text:0040105D ; ---------------------------------------------------------------------------
.text:0040105E jpt_40103A      dd offset loc_401041    ; DATA XREF: sub_401030+A↑r
.text:0040105E                                         ; jump table for switch statement
.text:00401062                 dd offset loc_401048
.text:00401066 ; ---------------------------------------------------------------------------
```

Anti-analysis - IDA Hex-Rays decompiler

```
secret_algo_obf PROC
        push ebp
        mov ebp, esp
        rdtsc
        cmp edx, 0
        ja real_code
        jmp dword ptr [real_code + edx * type dword]
case0:
        mov eax, 0
        jmp real_code
case1:
        mov eax, 1
        jmp real_code
real_code:
        ; start secret algo code
        mov edx, dword ptr [ebp+8]
        xor edx, 0C0D3CA05h
        mov eax, edx
        mov esp, ebp
        pop ebp
        ret
secret_algo_obf ENDP
```



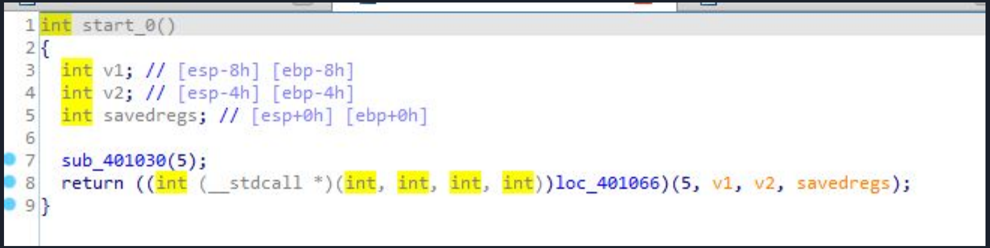
```
.text:00401066                                   ; start_0+F↓p
.text:00401066                 push    ebp
.text:00401067                 mov     ebp, esp
.text:00401069                 rdtsc
.text:0040106B                 cmp     edx, 0          ; switch 0 cases
.text:0040106E                 ja      short near ptr def_401070 ; jumptable 00401070 default case
.text:00401070                 jmp     def_401070[edx*4] ; switch jump
.text:00401070 ; ---------------------------------------------------------------------------
.text:00401077                 db 0B8h
.text:00401078                 dd 0
.text:0040107C                 dd 1B80EEBh, 0EB000000h, 2B807h, 0EB0000h
.text:0040108C def_401070      dd 8108558Bh            ; CODE XREF: .text:0040106E↑j
.text:0040108C                                         ; DATA XREF: .text:00401070↑r
.text:0040108C                                         ; jumptable 00401070 default case
.text:00401090                 dd 0D3CA05F2h, 8BC28BC0h
.text:00401098                 db 0E5h, 5Dh, 0C3h
.text:0040109B
.text:0040109B ; =============== S U B R O U T I N E =======================================
```

???

HACKINBO®
Winter 2019 Edition
13ª EDIZIONE

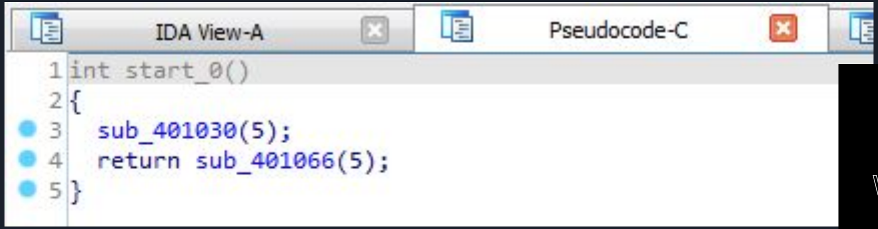
Anti-analysis - IDA Hex-Rays decompiler

```
.586
.model flat,stdcall
.stack 4096
.DATA
.CODE
main PROC
          push ebp
          mov ebp, esp

          push 5
          call secret_algo
          sub esp, 4

          push 5
          call secret_algo_obf
          sub esp, 4

          mov esp, ebp
          pop ebp
          ret
main ENDP
END main
```



```
int start_0()
{
  int v1; // [esp-8h] [ebp-8h]
  int v2; // [esp-4h] [ebp-4h]
  int savedregs; // [esp+0h] [ebp+0h]

  sub_401030(5);
  return ((int (__stdcall *)(int, int, int, int))loc_401066)(5, v1, v2, savedregs);
}
```

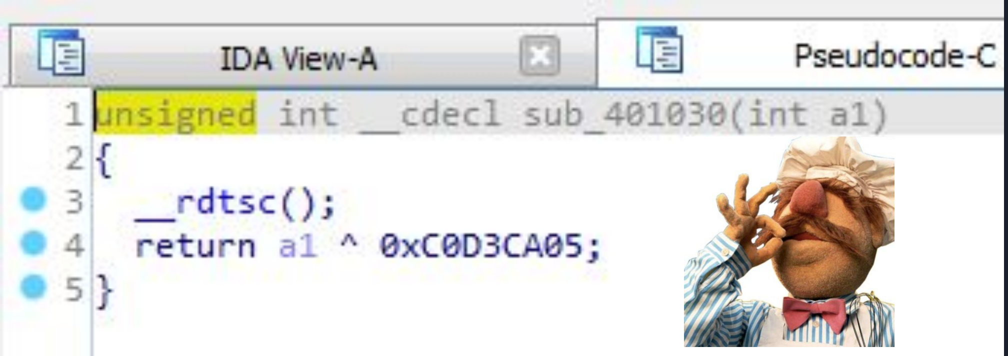
Let’s give IDA some love and re-define the data as code and create a function



```
int start_0()
{
  sub_401030(5);
  return sub_401066(5);
}
```

Anti-analysis - IDA Hex-Rays decompiler

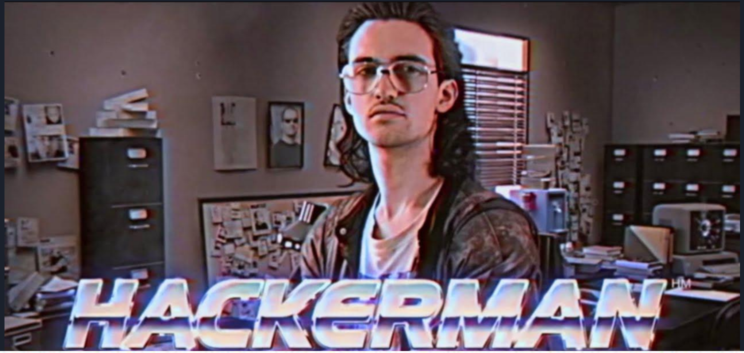
```
secret_algo PROC
          push ebp
          mov ebp, esp
          rdtsc
          cmp edx, 0
          ja real_code
          jmp dword ptr [switch_table + edx * type dword]
case0:
          mov eax, 0
          jmp real_code
case1:
          mov eax, 1
          jmp real_code
real_code:
          ; start secret algo code
          mov edx, dword ptr [ebp+8]
          xor edx, 0C0D3CA05h
          mov eax, edx
          mov esp, ebp
          pop ebp
          ret
switch_table:
          dword case0
          dword case1
secret_algo ENDP
```

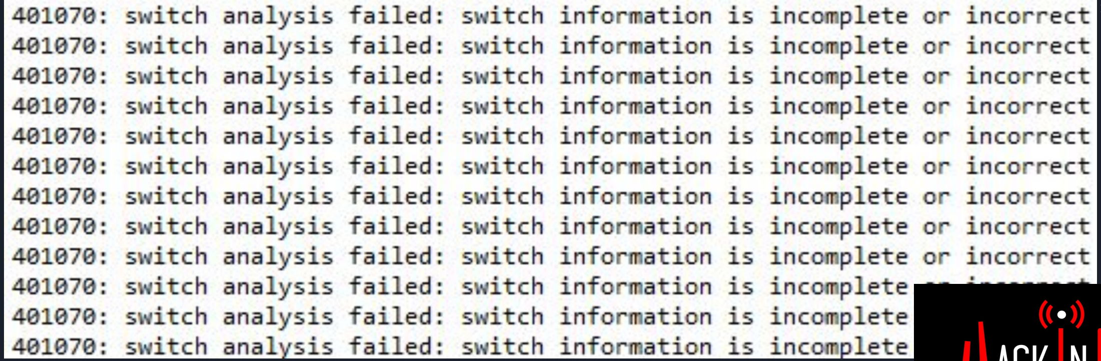


```
unsigned int __cdecl sub_401030(int a1)
{
  __rdtsc();
  return a1 ^ 0xC0D3CA05;
}
```

Anti-analysis - IDA Hex-Rays decompiler

```
secret_algo_obf PROC
        push ebp
        mov ebp, esp
        rdtsc
        cmp edx, 0
        ja real_code
        jmp dword ptr [real_code + edx * type dword]
case0:
        mov eax, 0
        jmp real_code
case1:
        mov eax, 1
        jmp real_code
real_code:
        ; start secret algo code
        mov edx, dword ptr [ebp+8]
        xor edx, 0C0D3CA05h
        mov eax, edx
        mov esp, ebp
        pop ebp
        ret
secret_algo_obf ENDP
```



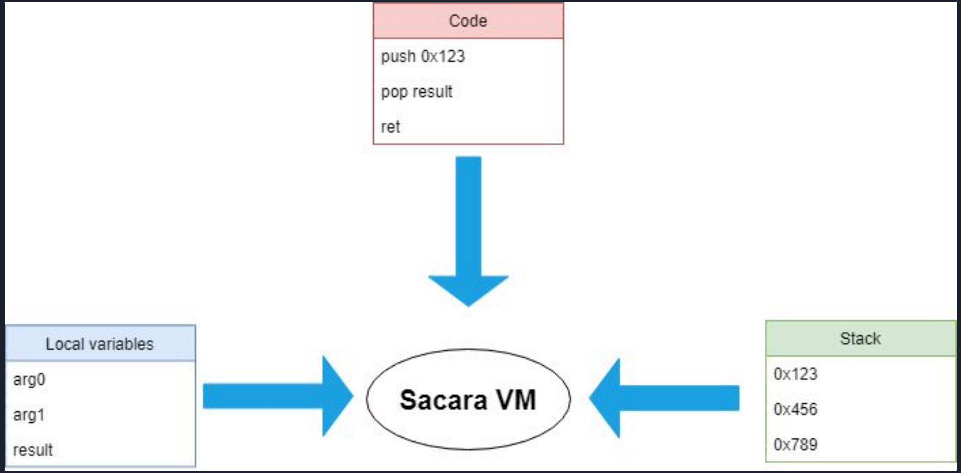


```
401070: switch analysis failed: switch information is incomplete or incorrect
401070: switch analysis failed: switch information is incomplete or incorrect
401070: switch analysis failed: switch information is incomplete or incorrect
401070: switch analysis failed: switch information is incomplete or incorrect
401070: switch analysis failed: switch information is incomplete or incorrect
401070: switch analysis failed: switch information is incomplete or incorrect
401070: switch analysis failed: switch information is incomplete or incorrect
401070: switch analysis failed: switch information is incomplete or incorrect
401070: switch analysis failed: switch information is incomplete or incorrect
401070: switch analysis failed: switch information is incomplete
401070: switch analysis failed: switch information is incomplete
401070: switch analysis failed: switch information is incomplete
```
*

* This problem is well know to IDA devs: https://www.hex-rays.com/products/decompiler/manual/failures.shtml#06

VM based obfuscation

- One of the most difficult task in Reverse Engineering is to understand how the underline computer architecture works (instruction set, calling convention, memory layout, compiler characteristics, used Libs, …)
- We are very used to INTEL arch on Windows OS, but what about a new unknown architecture? This is the basic concept of VM base protection
- A personal experiment, Sacara: https://github.com/enkomio/sacara



Code
push 0x123
pop result
ret

Local variables
arg0
arg1
result

Sacara VM

Stack
0x123
0x456
0x789

VM based obfuscation

Example: decrypt a buffer

Src: https://github.com/enkomio/sacara/blob/master/Src/Examples/LoadEncryptedAssembly/Encryption.cs

```
proc main
   push buffer
   push buffer_length
   push key
   push key_length
   push 4
   push de_encrypt
   call
   halt
endp
```

```
/*
This method accept:
1 - the length of the password
2 - a pointer to the password to use
3 - the lengh of the buffer
4 - a pointer to the buffer
*/
proc de_encrypt
   pop key_length
   pop key
   pop buffer_length
   pop buffer
   push 0
   pop buffer_index
   push 0
   pop key_index
   push 0
   pop buffer_char
   push 0
   pop key_char
```

```
encryption_loop:
   /* read the character from the buffer */
   push buffer_index
   push buffer
   add
   nread
   pop buffer_char
   /* read the character from the key buffer */
   push key_index
   push key
   add
   nread
   pop key_char
   /* do XOR and save the result on the stack */
   push key_char
   push buffer_char
   xor
   /* write back the result */
   push buffer_index
   push buffer
   add
   nwrite
```

```
   /* increase counter */
   push 1
   push key_index
   add
   pop key_index
   push 1
   push buffer_index
   add
   pop buffer_index
   /* check if I have [illegible]
   push key[illegible]
   p[illegible]_index
   cmp
   push check_for_comp
   jumpifl
round_key:
   push 0
   pop key_index
```

```
check_for_completation:
   push buffer_length
   push buffer_index
```

```
/* do XOR and save the result on the stack */
push key_char
push buffer_char
xor
```

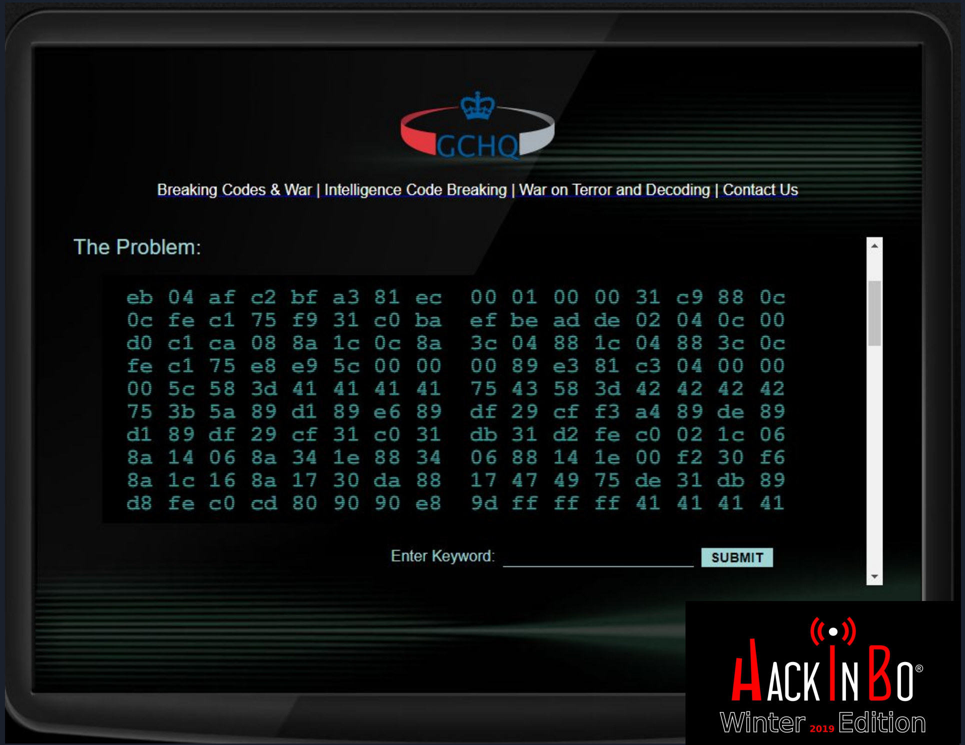
VS

```
xor eax, ebx
```

- Encrypted Opcode
- Anti-tampering
- ...

Reverse Engineering != Reading Assembly

- Doing Reverse Engineering doesn’t always imply to read Assembly
- Sometimes it is easier to just try to get rid of the data by looking for patterns
- Some interesting links:
  - https://www.canyoucrackit.co.uk/codeexplained.html
  - http://blog.pi3.com.pl/?p=213
- If you want a more fresh challenge and you like more NSA, here is another one:
  - https://codebreaker.ltsnet.net/challenge



Breaking Codes & War | Intelligence Code Breaking | War on Terror and Decoding | Contact Us

The Problem:

```
eb 04 af c2 bf a3 81 ec  00 01 00 00 31 c9 88 0c
0c fe c1 75 f9 31 c0 ba  ef be ad de 02 04 0c 00
d0 c1 ca 08 8a 1c 0c 8a  3c 04 88 1c 04 88 3c 0c
fe c1 75 e8 e9 5c 00 00  00 89 e3 81 c3 04 00 00
00 5c 58 3d 41 41 41 41  75 43 58 3d 42 42 42 42
75 3b 5a 89 d1 89 e6 89  df 29 cf f3 a4 89 de 89
d1 89 df 29 cf 31 c0 31  db 31 d2 fe c0 02 1c 06
8a 14 06 8a 34 1e 88 34  06 88 14 1e 00 f2 30 f6
8a 1c 16 8a 17 30 da 88  17 47 49 75 de 31 db 89
d8 fe c0 cd 80 90 90 e8  9d ff ff ff 41 41 41 41
```

Enter Keyword:

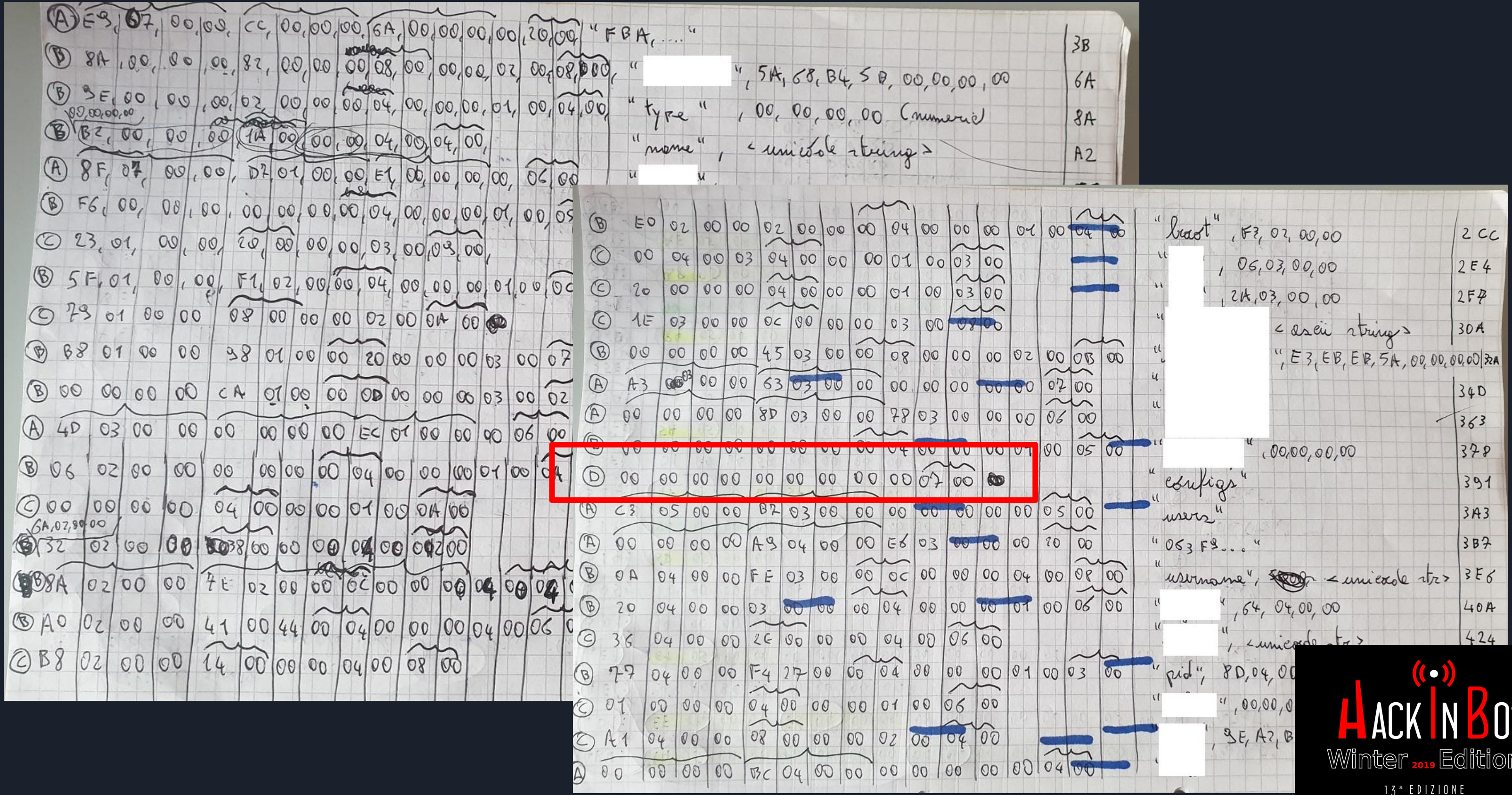
Reverse Engineering != Reading Assembly

- A real world case
  - File containing information about compromised computers
  - Malware written in C++, the code that read and update the file wasn’t easy to understand and difficult to trigger
  - File seems to be in plain text (no encryption)

- Initial bytes

```
0x73 0x65 0x63 0x72 0x03 0x00 0x00 0x00 0x18 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x27 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x3B 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x05 0x00 0x73 0x34 0x74 0x61 0x6E 0x02 0x01 0x00 0x00 0xB3
0x00 0x00 0x00 0x6A 0x00 0x00 0x00 0x00 0x20 0x00 0x68 0x74 0x74 0x70 0x73 0x3A
0x2F 0x2F 0x77 0x77 0x77 0x2E 0x74 0x61 0x69 0x70 0x61 0x6E 0x73 0x65 0x63 0x2E
0x63 0x6F 0x6D 0x2F 0x69 0x6E 0x64 0x65 0x78 0x20 0x8A 0x00 0x00 0x00 0x82 0x00
0x00 0x00 0x08 0x00 0x00 0x00 0x02 0x00 0x08 0x00 0x70 0x72 0x6F 0x65 0x64 0x69
0x74 0x69 0xF5 0x7B 0xE9 0x5B 0x00 0x00 0x00 0x00 0x9E 0x00 0x00 0x00 0x02 0x00
0x00 0x00 0x04 0x00 0x00 0x00 0x01 0x00 0x04 0x00 0x63 0x6F 0x6F 0x6C 0xC6 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x04 0x00 0x00 0x00 0x01 0x00 0x05 0x00 0x69 0x74
0x61 0x6C 0x79 0x2A 0x07 0x00 0x00 0x00 0x00 0x00 0x00 0xCB 0x04 0x00 0x00 0x00
0x04 0x00 0x69 0x6E 0x66 0x6F 0xE6 0x00 0x00 0x00 0xD8 0x00 0x00 0x00 0x0F 0x00
0x00 0x00 0x03 0x00 0x02 0x00 0x48 0x61 0x63 0x6B 0x49 0x6
0x39 0x31 0x31 0x30 0x31 0x39 0xEE 0x07 0x00 0x00 0xFA 0x0
```

Reverse Engineering != Reading Assembly

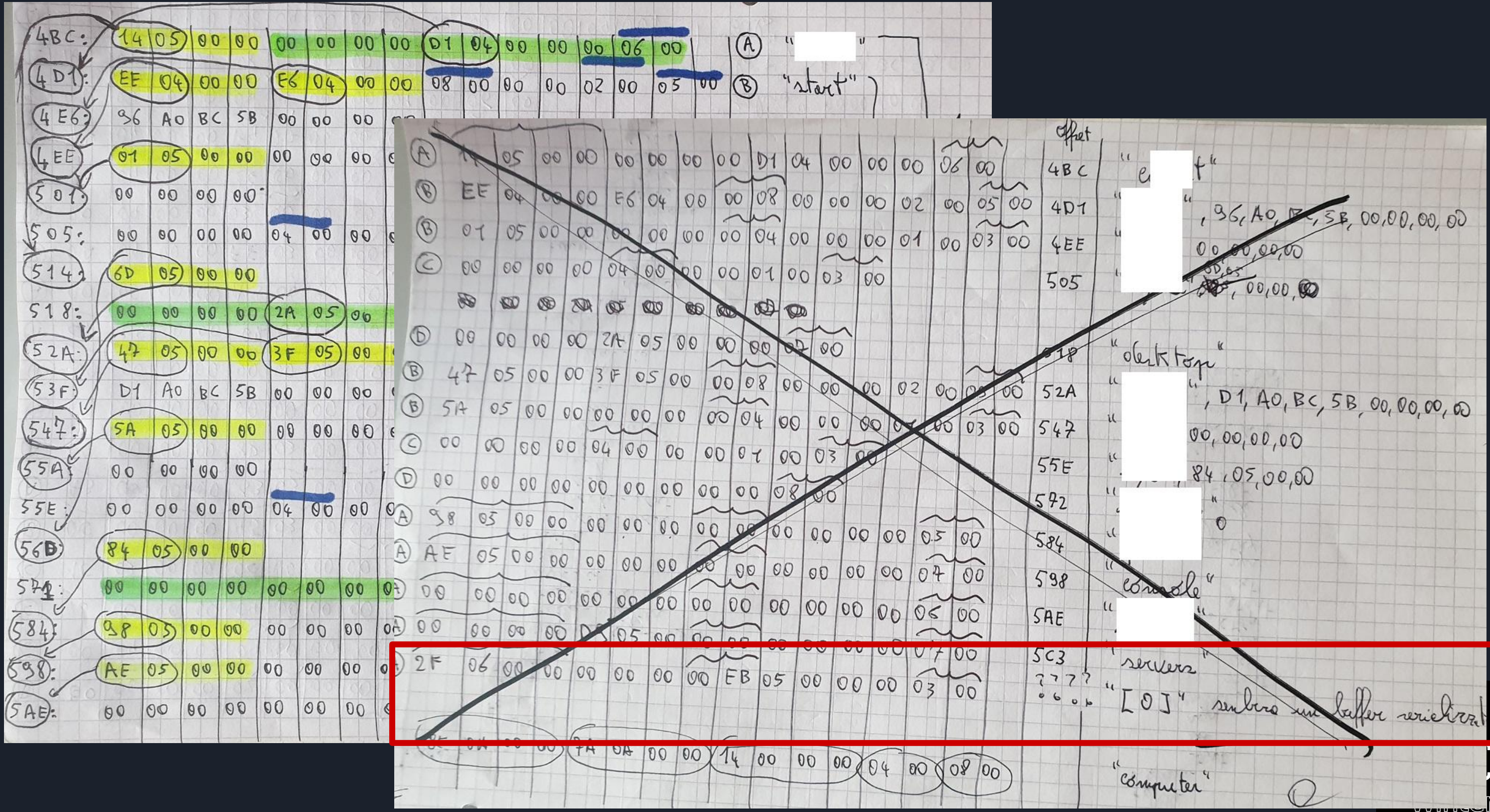


```
A E9, 07, 00, 00, CC, 00, 00, 00, 6A, 00, 00, 00, 00, 20, 00  "FBA, ... "                    3B
B 8A, 00, 00, 00, 82, 00, 00, 00, 08, 00, 00, 00, 02, 00, 08, 00  "[redacted]", 5A, 68, B4, 5B, 00, 00, 00, 00   6A
B 3E, 00, 00, 00, 02, 00, 00, 00, 04, 00, 00, 00, 01, 00, 04, 00  "type", 00, 00, 00, 00 (numeric)   8A
  00,00,00,00
B B2, 00, 00, 00, 1A, 00, 00, 00, 04, 00, 04, 00  "nome", <unicode string>                     A2
A 8F, 07, 00, 00, D7, 01, 00, 00, E1, 00, 00, 00, 00, 06, 00  "[redacted]"
B F6, 00, 00, 00, 00, 00, 00, 00, 04, 00, 00, 00, 01, 00, 05
C 23, 01, 00, 00, 20, 00, 00, 00, 03, 00, 03, 00
B 5F, 01, 00, 00, F1, 02, 00, 00, 04, 00, 00, 00, 01, 00, 0C
C 79  01  00  00  08  00  00  00  02  00  0A  00  00
B B8  01  00  00  98  01  00  00  20  00  00  00  03  00  07
B 00  00  00  00  CA  01  00  00  0D  00  00  00  03  00  02
A 4D  03  00  00  00  00  00  00  EC  01  00  00  00  06  00
B 06  02  00  00  00  00  00  00  04  00  00  00  01  00  04
C 00  00  00  00  04  00  00  00  01  00  0A  00
  6A,07,80,00
B 32  02  00  00  38  00  00  00  04  00  02  00
B 8A  02  00  00  7E  02  00  00  0C  00  00  00  04  00  04
B A0  02  00  00  41  00  44  00  04  00  00  00  04  00  06
C B8  02  00  00  14  00  00  00  04  00  08  00
```

```
B E0 02 00 00 02 00 00 00 04 00 00 00 01 00 04 00  "boot", F3, 02, 00, 00            2CC
C 00 04 00 03 04 00 00 00 01 00 03 00              "[redacted]", 06, 03, 00, 00      2E4
C 20 00 00 00 04 00 00 00 01 00 03 00              "[redacted]", 2A, 03, 00, 00      2F7
C 1E 03 00 00 0C 00 00 00 03 00 07 00              "[redacted]" <ascii string>      30A
B 00 00 00 00 45 03 00 00 08 00 00 00 02 00 0B 00  "[redacted]", E3, EB, EB, 5A, 00, 00, 00, 00 | 32A
A A3 03 00 00 63 03 00 00 00 00 00 00 07 00        "                                 34D
A 00 00 00 00 8D 03 00 00 78 03 00 00 00 06 00     "                                 363
  00 00 00 00 00 00 00 00 04 00 00 00 01 00 05 00  "[redacted]", 00, 00, 00, 00      378
D 00 00 00 00 00 00 00 00 00 07 00 00              "configs"                         391
A C3 05 00 00 B2 03 00 00 00 00 00 00 05 00        "users"                           3A3
A 00 00 00 00 A9 04 00 00 E6 03 00 00 20 00        "053F9..."                        3B7
B 0A 04 00 00 FE 03 00 00 0C 00 00 00 04 00 08 00  "username", <unicode str>         3E6
B 20 04 00 00 03 00 00 00 04 00 00 00 01 00 06 00  "[redacted]", 64, 04, 00, 00      40A
C 36 04 00 00 2E 00 00 00 04 00 06 00              "[redacted]", <unicode str>       424
B 77 04 00 00 F4 27 00 00 04 00 00 00 01 00 03 00  "pid", 8D, 04, 00
C 01 00 00 00 04 00 00 00 01 00 06 00              "[redacted]", 00, 00, 0
C A1 04 00 00 08 00 00 00 02 00 04 00              "[redacted]", 9E, A2, B
A 00 00 00 00 BC 04 00 00 00 00 00 00 04 00
```

HackInBo® Winter 2019 Edition
13ª EDIZIONE

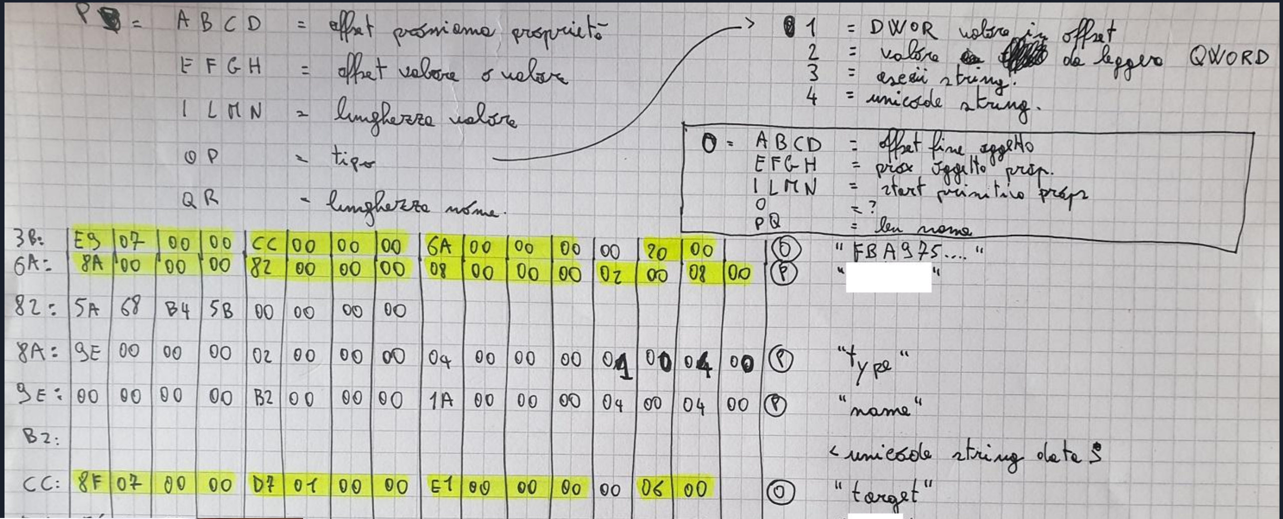
Reverse Engineering != Reading Assembly



```
4BC: 14 05 00 00 00 00 00 00 D1 04 00 00 00 06 00      (A) "[redacted]"
4D1: EE 04 00 00 E6 04 00 00 08 00 00 00 02 00 05 00   (B) "start"
4E6: 96 A0 BC 5B 00 00 00
4EE: 01 05 00 00 00 00 00
501: 00 00 00 00
505: 00 00 00 00 04 00 00
514: 6D 05 00 00
518: 00 00 00 00 2A 05 00
52A: 47 05 00 00 3F 05 00
53F: D1 A0 BC 5B 00 00 00
547: 5A 05 00 00 00 00 00
55A: 00 00 00 00
55E: 00 00 00 00 04 00 00
56D: 84 05 00 00
571: 00 00 00 00 00 00 00 0
584: 98 05 00 00 00 00 00
598: AE 05 00 00 00 00 00
5AE: 00 00 00 00 00 00 00
```

```
                                                          offset
(A) 14 05 00 00 00 00 00 00 D1 04 00 00 00 06 00          4BC   "e[redacted]t"
(B) EE 04 00 00 E6 04 00 00 08 00 00 00 02 00 05 00       4D1   "[redacted]", 96, A0, BC, 5B, 00,00,00,00
(B) 01 05 00 00 00 00 00 00 04 00 00 00 01 00 03 00       4EE   00,00,00,00
(C) 00 00 00 00 04 00 00 00 01 00 03 00                   505   0D,05, 00,00,00
(D) 00 00 00 00 2A 05 00 00 00 07 00                      518   "desktop"
(B) 47 05 00 00 3F 05 00 00 08 00 00 00 02 00 03 00       52A   "[redacted]", D1, A0, BC, 5B, 00,00,00,00
(B) 5A 05 00 00 00 00 00 00 04 00 00 00 07 00 03 00       547   00,00,00,00
(C) 00 00 00 00 04 00 00 00 01 00 03 00                   55E   84, 05,00,00
(D) 00 00 00 00 00 00 00 00 00 08 00                      572   "[redacted]" 0
(A) 98 05 00 00 00 00 00 00 00 00 00 00 00 05 00          584   "[redacted]"
(A) AE 05 00 00 00 00 00 00 00 00 00 00 00 07 00          598   "console"
(A) 00 00 00 00 00 00 00 00 00 00 00 00 00 06 00          5AE   "[redacted]"
(A) 00 00 00 00 D 05 00 00 00 00 00 00 00 07 00           5C3   "servers"
(A) 2F 06 00 00 00 00 00 00 EB 05 00 00 00 03 00          ????  "[O]" sembra un buffer serializz
    05 04 00 00 7A 0A 00 00 14 00 00 00 04 00 08 00             "computer"
```

Reverse Engineering != Reading Assembly



P = A B C D = offset prossima proprietà
E F G H = offset valore o valore
I L M N = lunghezza valore
O P = tipo
Q R = lunghezza nome

1 = DWOR valore in offset
2 = valore da leggere QWORD
3 = ascii string.
4 = unicode string.

O = ABCD = offset fine oggetto
EFGH = prox oggetto prop.
ILMN = start primitivo props
O = ?
PQ = len nome

3B:	E9	07	00	00	CC	00	00	00	6A	00	00	00	00	20	00		Ⓢ	"FBA975...."
6A:	8A	00	00	00	82	00	00	00	08	00	00	00	02	00	08	00	Ⓟ	" [redacted] "
82:	5A	68	B4	5B	00	00	00	00										
8A:	9E	00	00	00	02	00	00	00	04	00	00	00	01	00	04	00	Ⓟ	"type"
9E:	00	00	00	00	B2	00	00	00	1A	00	00	00	04	00	04	00	Ⓟ	"name"
B2:																		< unicode string date $
CC:	8F	07	00	00	D7	01	00	00	E1	00	00	00	00	06	00		Ⓞ	"target"
						0	00	00	04	00	00	00	01	00	05	00	Ⓟ	" [redacted] "
						0	00	00	04	00	00	00	01	00	04	00	Ⓟ	" [redacted] "
						1	00	00	20	00	00	00	03	00	03	00	Ⓟ	" [redacted] "
																		< ascii string >

HackInBo® Winter 2019 Edition
13ª EDIZIONE

Sojobo a B2R2 emulator

- Sojobo emulates the B2R2 IR in order to provide an environment where you can emulate the execution of a binary. You can download it from:

★ Star https://github.com/enkomio/Sojobo

- At the current state it supports:
  - Intel architecture X86 32 bit
  - Window Process
  - A limited API set
- Tengu is a command line debugger like tool based on Sojobo
  - Same command switches as **windbg**
  - It allows to save snapshot
  - It emulates main Windows functions

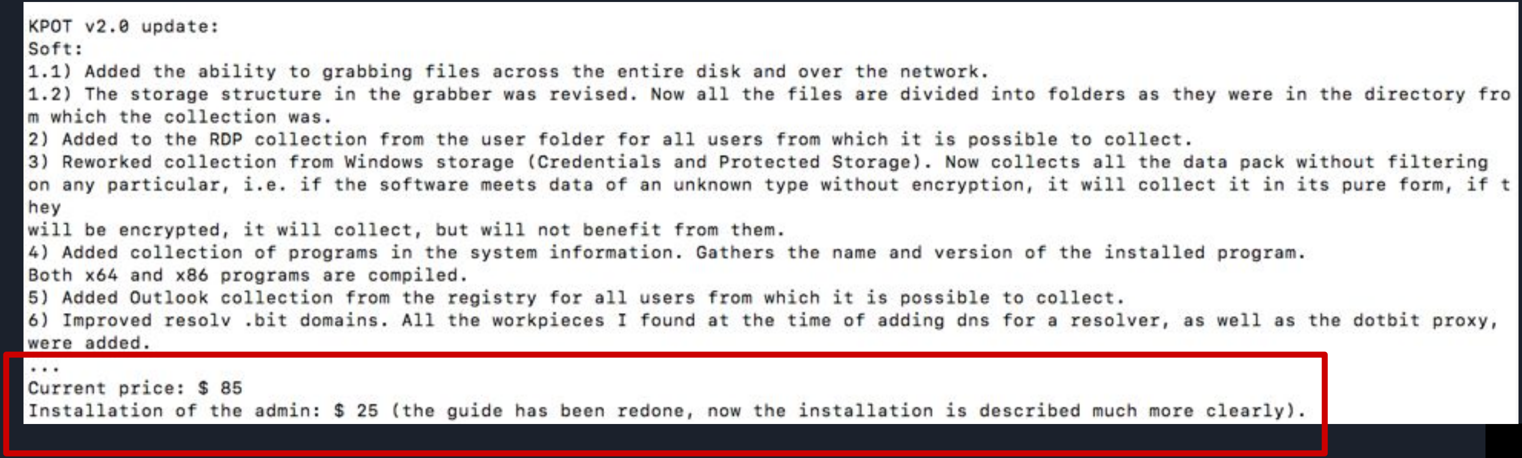
HACK IN BO®
Winter 2019 Edition
13ª EDIZIONE

Sojobo a B2R2 emulator

```
// emulate a malware and take snapshot at a given address
let sandbox = new Win32Sandbox()
let snapshotManager = new SnapshotManager(sandbox)
sandbox.Load(malwareFile)

// setup handlers
sandbox.BeforeEmulation.Add(fun proc ->
        if 0x401061 = proc.ProgramCounter.As<Int32>() then
                snapshotManager.TakeSnaphot()
)

// run the sample
sandbox.Run()
```

Case Study: KPOT v2

- KPOT v2 is an information stealer malware sold on underground forums
- A description about the malware is provided by the author



```
KPOT v2.0 update:
Soft:
1.1) Added the ability to grabbing files across the entire disk and over the network.
1.2) The storage structure in the grabber was revised. Now all the files are divided into folders as they were in the directory fro
m which the collection was.
2) Added to the RDP collection from the user folder for all users from which it is possible to collect.
3) Reworked collection from Windows storage (Credentials and Protected Storage). Now collects all the data pack without filtering
on any particular, i.e. if the software meets data of an unknown type without encryption, it will collect it in its pure form, if t
hey
will be encrypted, it will collect, but will not benefit from them.
4) Added collection of programs in the system information. Gathers the name and version of the installed program.
Both x64 and x86 programs are compiled.
5) Added Outlook collection from the registry for all users from which it is possible to collect.
6) Improved resolv .bit domains. All the workpieces I found at the time of adding dns for a resolver, as well as the dotbit proxy,
were added.
...
Current price: $ 85
Installation of the admin: $ 25 (the guide has been redone, now the installation is described much more clearly).
```

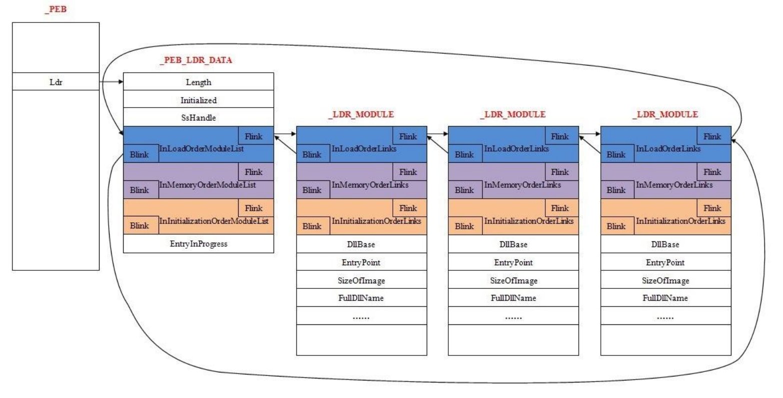
*

* Source: https://www.proofpoint.com/us/threat-insight/post/new-kpot-v20-stealer-brings-zero-persistence-and-memory-features-silently-steal

KPOT function resolution algorithm

Steps to resolve a function pointer:

1. Walk **TEB->PEB->Ldr** to get the base address for Kernel and ntdll. Resolve *LoadLibraryA* by walking Kernel32 EAT. Use *LoadLibraryA* to load the desired DLLs
2. Store the DLL base address and other info in a structure composed by the following items: *<base address, number of functions to lookup, function array>*
3. Parse PE and walk EAT. For each exported function compute the *MurmurHash* hash and search for this value in the above array. If found store the pointer.

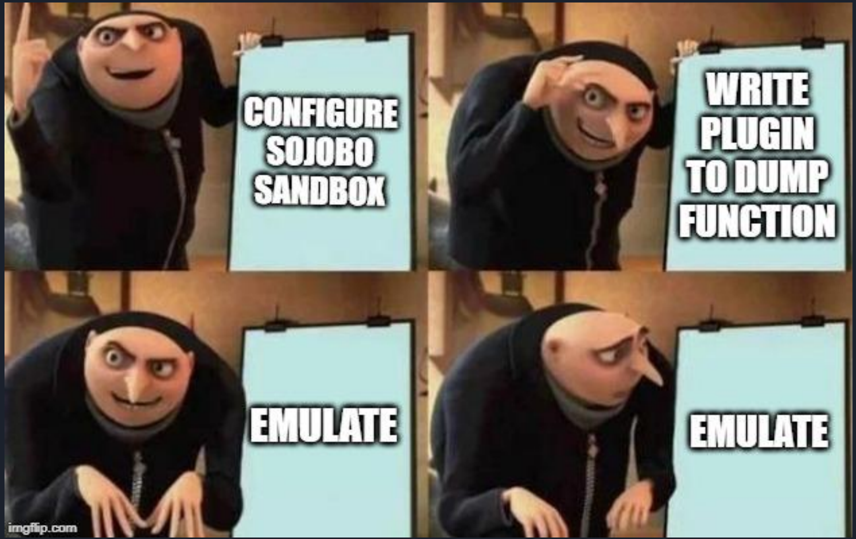


```
0018FB54  3C 00 00 00 00 00 1B 75 B0 FC 18 00 0B 00 00 00  <......u°ü......
0018FB64  00 00 46 6C 08 FD 18 00 0A 00 00 00 00 00 2B 75  ..Fl.ý........+u
0018FB74  58 FD 18 00 0A 00 00 00 00 00 EA 76 38 FE 18 00  Xý........êv8þ..
0018FB84  06 00 00 00 00 00 5F 77 B8 FE 18 00 04 00 00 00  ......_w¸þ......
```

- DLL Base address
- Array of hash to search for
- Number of hash in array

Goal: We want to know which are the functions that are resolved by the malware

- Sample SHA-256 : 67f8302a2fd28d15f62d6d20d748bfe350334e5353cbdef112bd1f8231b5599d
- By knowing which are the used functions we can have a better picture of the malware functionalities. Let’s emulate the previous steps in Sojobo.



Goal: We want to know which are the functions that are resolved by the malware

At Step 1 we have the biggest problem. We need to have a valid PEB structure to correctly emulate the execution. The *Ldr* field is one of the most difficult to represent since it contains a linked list via LIST_ENTRY structure.

At lower level it is easy to manage LIST_ENTRY, but how to represent it at a high level language like F#? Possible solution:

- LIST_ENTRY can point to any kind of data, it is a nice use case for using inheritance!
  - We can’t do this if we consider LIST_ENTRY like a struct. Struct cannot be inherited by definition.
- Then consider LIST_ENTRY as a class
  - We can’t do this, since it is treated like a structure (it occupy 8 bytes in x86, since it has 2 pointers). If we define it like a class we will have a pointer during serialization (4 bytes and not 8).
- Treat it as a struct and consider the pointed object like a generic Object class
  - Goodbye deserialization → Impossible to know during deserialization which Object type we have to create
- ...

Goal: We want to know which are the functions that are resolved by the malware

- Writing Binary Analysis tools it's not an easy task :)

HACK IN BO®
Winter 2019 Edition
13ª EDIZIONE

Conclusion

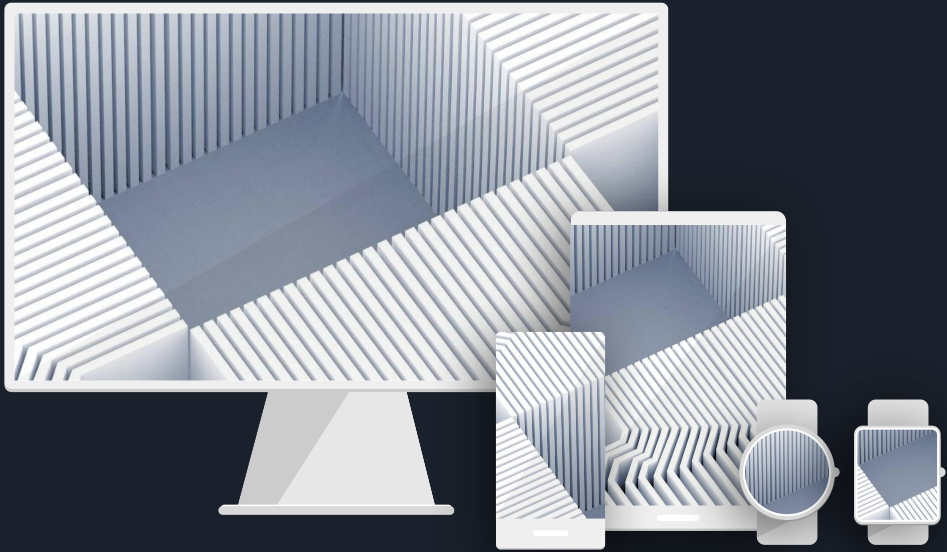
- Effective malware can be very complex
- Effective anti-analysis techniques can slower the reverse engineering process
  - Anti-VM
  - Anti-Debugging
  - VM based protection
- Some implementation choices can further slow the analysis
  - Usage of rarely used compression algorithms
  - Usage of external lib for crypto instead of relying on Windows Crypto API
- There are many tools that can help to analyze malware, not only debuggers and disassemblers :)
  - In order to be proficient with them is necessary to have some basic/medium knowledge about reverse engineering

Thank you!

Twitter: s4tan

GitHub: https://github.com/sponsors/enkomio

Contact: aparata@gmail.com



HackInBo® Winter 2019 Edition
13ª EDIZIONE