<MALWARE ANALYSIS>

Joyce Yeo

INTERNSHIP STRUCTURE

Research Project

> Weekly 30-min meetings

> Mostly independent work

> Presentation to CSL (40+)

INTRODUCTION

Examining
anti-analysis techniques
used by malware

(nb: not anti-detection)

Project Scope

- Anti-debugging
- Anti-VM

- Packer detection techniques

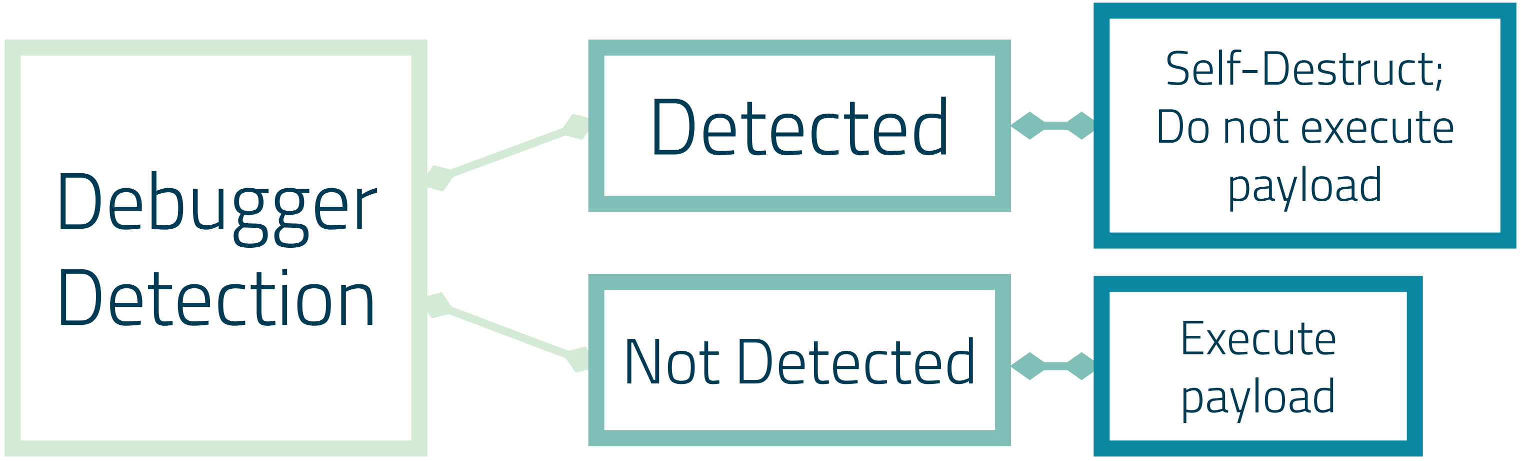
ANTI-DEBUGGING

Techniques to thwart debuggers

Anti-Debugging Techniques

$$\begin{tikzcd}\boxed{\text{Debugger Detection}} \arrow[r, leftrightarrow] & \boxed{\text{Anti-Attaching}}\end{tikzcd}$$

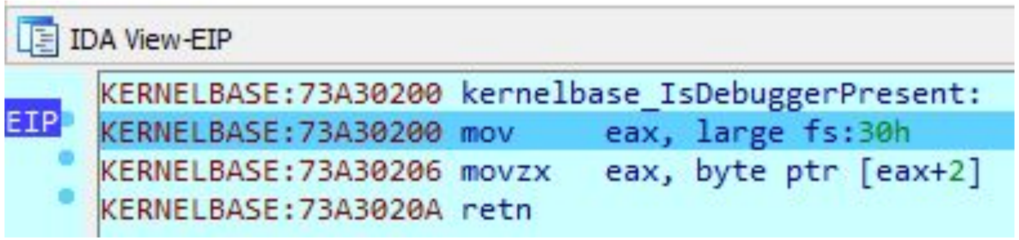
Debugger-detection Workflow



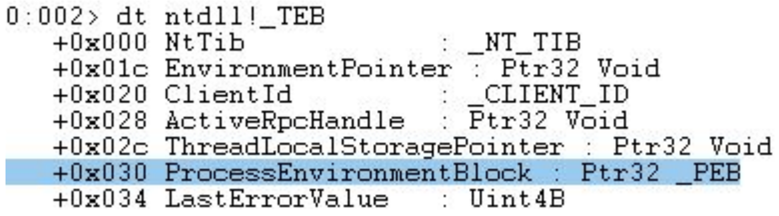
Debugger-detection: API-based

- IsDebuggerPresent
  - Retrieves BeingDebugged flag in PEB

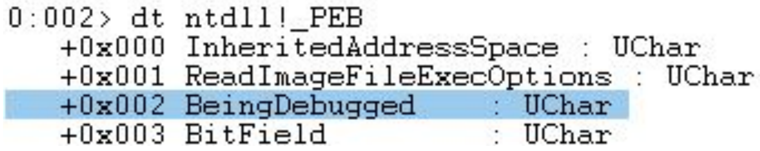
Debugger-detection: API-based



```
KERNELBASE:73A30200 kernelbase_IsDebuggerPresent:
KERNELBASE:73A30200 mov     eax, large fs:30h
KERNELBASE:73A30206 movzx   eax, byte ptr [eax+2]
KERNELBASE:73A3020A retn
```



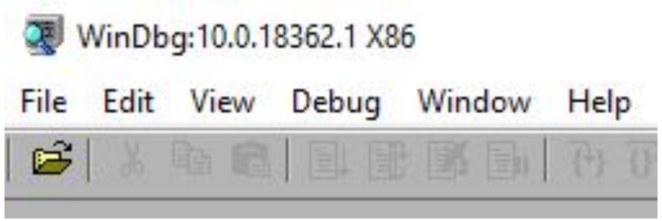
```
0:002> dt ntdll!_TEB
   +0x000 NtTib                 : _NT_TIB
   +0x01c EnvironmentPointer    : Ptr32 Void
   +0x020 ClientId              : _CLIENT_ID
   +0x028 ActiveRpcHandle       : Ptr32 Void
   +0x02c ThreadLocalStoragePointer : Ptr32 Void
   +0x030 ProcessEnvironmentBlock : Ptr32 _PEB
   +0x034 LastErrorValue        : Uint4B
```

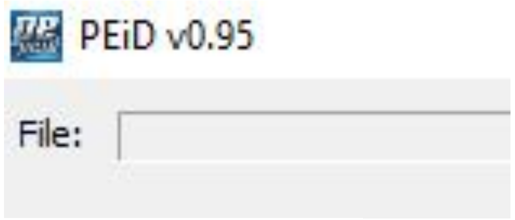


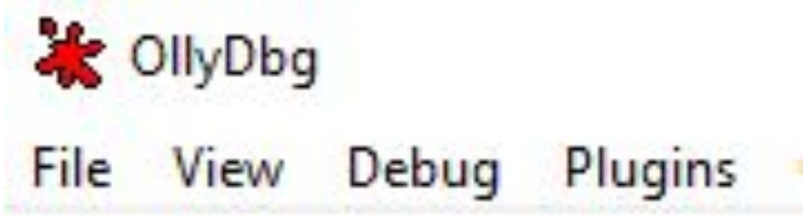
```
0:002> dt ntdll!_PEB
   +0x000 InheritedAddressSpace : UChar
   +0x001 ReadImageFileExecOptions : UChar
   +0x002 BeingDebugged         : UChar
   +0x003 BitField              : UChar
```

Debugger-detection: API-based

- FindWindow
  - Checks for handles by name

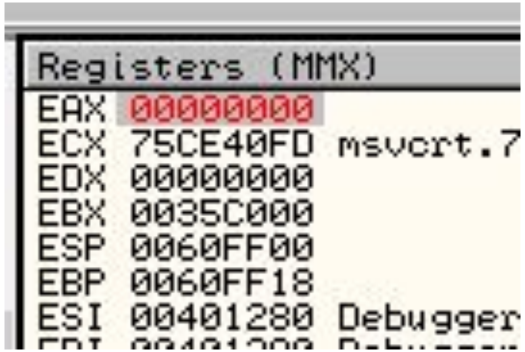




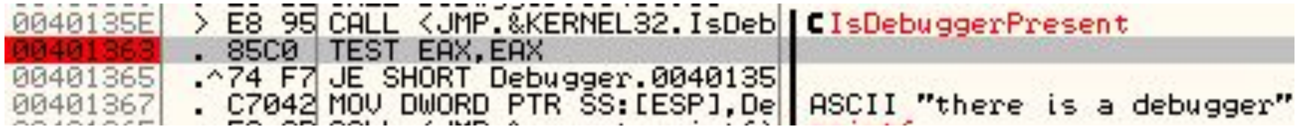


Debugger-detection: API-based

- Easy for analysts to patch
  - Modifying the return value of the function



```
Registers (MMX)
EAX 00000000
ECX 75CE40FD msvcrt.7
EDX 00000000
EBX 0035C000
ESP 0060FF00
EBP 0060FF18
ESI 00401280 Debugger
```



```
0040135E  > E8 95 CALL <JMP.&KERNEL32.IsDeb  CIsDebuggerPresent
00401363  . 85C0  TEST EAX,EAX
00401365  .^74 F7 JE SHORT Debugger.0040135
00401367  . C7042 MOV DWORD PTR SS:[ESP],De  ASCII "there is a debugger"
```

Debugger-detection: Exception-based

- OutputDebugString

Debugger present:
No error raised

Debugger absent:
Error raised

- `setLastError();`
- `OutputDebugString("");`
- `checkLastError();`

Debugger-detection: Exception-based

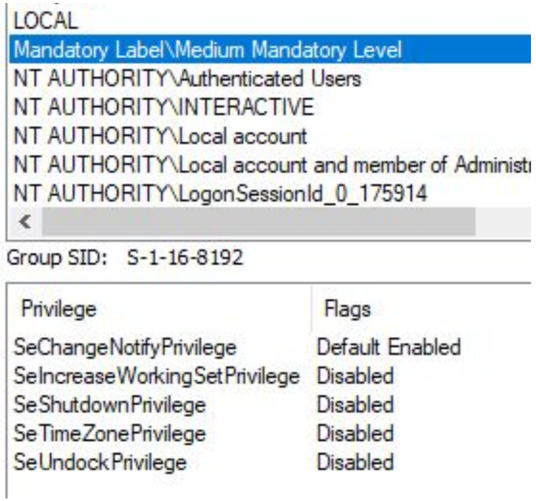
- Only works for Windows XP
  - Change in implementation of OutputDebugString
  - Error no longer raised

- False positives in newer OS
  - premature termination

Debugger-detection: Privilege-based

- SeDebugPrivilege
  - Inspect and adjust the memory of other processes and threads, regardless of security descriptors
  - Required by debuggers

Debugger-detection: Privilege-based

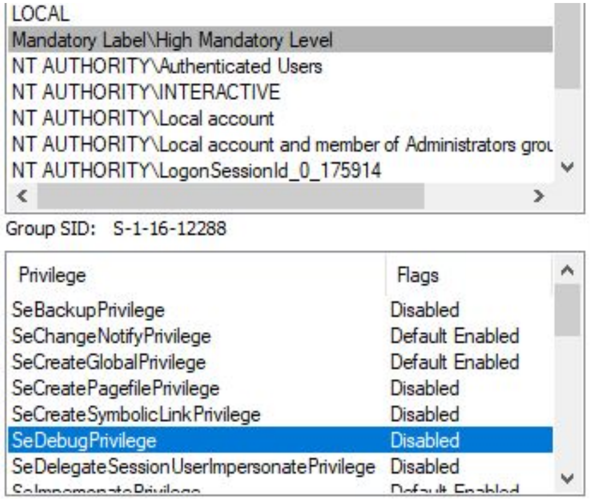


LOCAL
Mandatory Label\Medium Mandatory Level
NT AUTHORITY\Authenticated Users
NT AUTHORITY\INTERACTIVE
NT AUTHORITY\Local account
NT AUTHORITY\Local account and member of Administ
NT AUTHORITY\LogonSessionId_0_175914

Group SID: S-1-16-8192

Privilege	Flags
SeChangeNotifyPrivilege	Default Enabled
SeIncreaseWorkingSetPrivilege	Disabled
SeShutdownPrivilege	Disabled
SeTimeZonePrivilege	Disabled
SeUndockPrivilege	Disabled

Medium Integrity (default)



LOCAL
Mandatory Label\High Mandatory Level
NT AUTHORITY\Authenticated Users
NT AUTHORITY\INTERACTIVE
NT AUTHORITY\Local account
NT AUTHORITY\Local account and member of Administrators grou
NT AUTHORITY\LogonSessionId_0_175914

Group SID: S-1-16-12288

Privilege	Flags
SeBackupPrivilege	Disabled
SeChangeNotifyPrivilege	Default Enabled
SeCreateGlobalPrivilege	Default Enabled
SeCreatePagefilePrivilege	Disabled
SeCreateSymbolicLinkPrivilege	Disabled
SeDebugPrivilege	Disabled
SeDelegateSessionUserImpersonatePrivilege	Disabled
SeImpersonatePrivilege	Default Enabled

High Integrity (elevated)

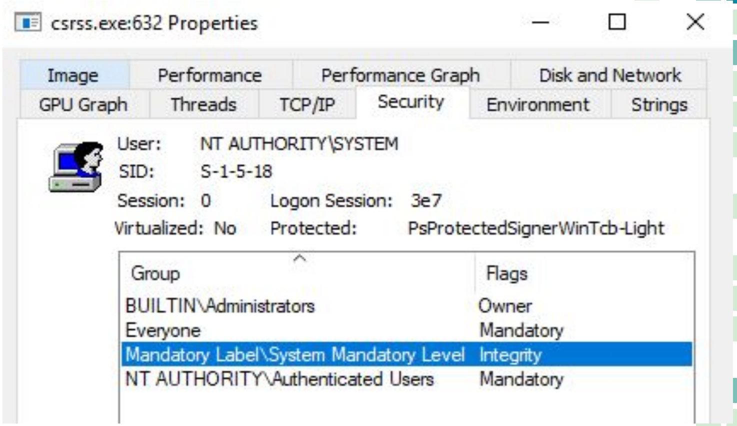
Debugger-detection: Privilege-based

- Explicit: Read SeDebugPrivilege
- Implicit check via OpenCsrss (Client/ Server Runtime Subsystem)
  - Similar to exception-based

Debugger-detection: Privilege-based

- Only works for Windows XP and 7
  - Introduction of Windows protected processes

- False negatives (Win10)
  - failed anti-debugging



csrss.exe:632 Properties

Image | Performance | Performance Graph | Disk and Network | GPU Graph | Threads | TCP/IP | Security | Environment | Strings

User: NT AUTHORITY\SYSTEM
SID: S-1-5-18
Session: 0 Logon Session: 3e7
Virtualized: No Protected: PsProtectedSignerWinTcb-Light

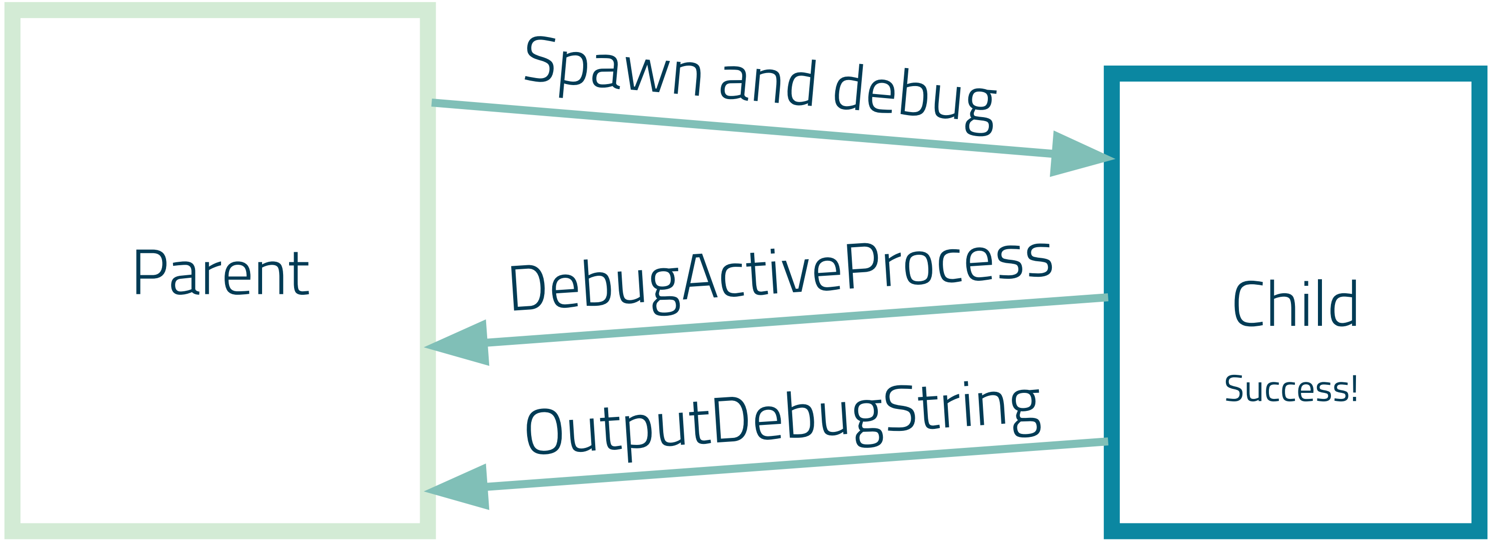
Group	Flags
BUILTIN\Administrators	Owner
Everyone	Mandatory
Mandatory Label\System Mandatory Level	Integrity
NT AUTHORITY\Authenticated Users	Mandatory

Anti-Attaching: Self-debugging

- Only one process can do **invasive debugging**
  - Able to suspend threads, access memory

- Unable to debug same process
  - Cannot suspend your own threads
  - Spawn a debugger

Anti-Attaching: Self-debugging



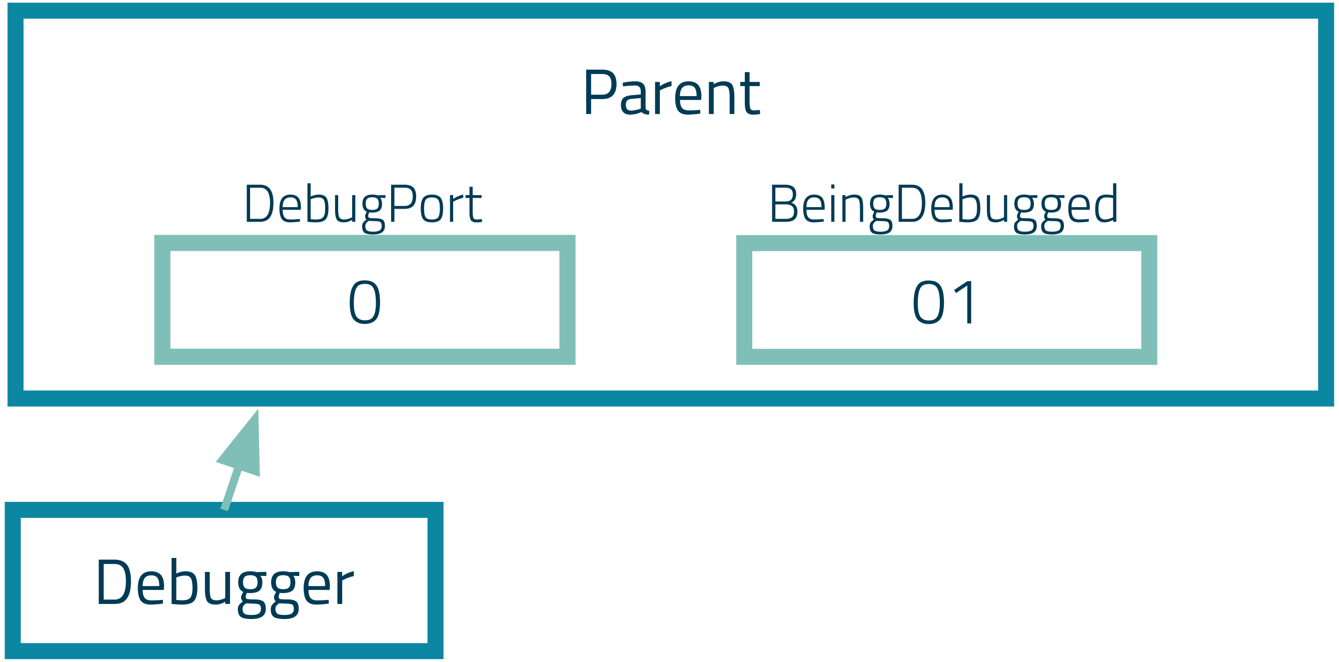
Anti-Attaching: Self-debugging

- Debugger can be detached

- User-mode: BeingDebugged flag in PEB
- Kernel-mode: DebugPort in EPROCESS
  - Contains DebugObject handle

- Set the DebugPort in EPROCESS to 0

Anti-Attaching: Self-debugging



ANTI-VM

Techniques to detect virtualisation

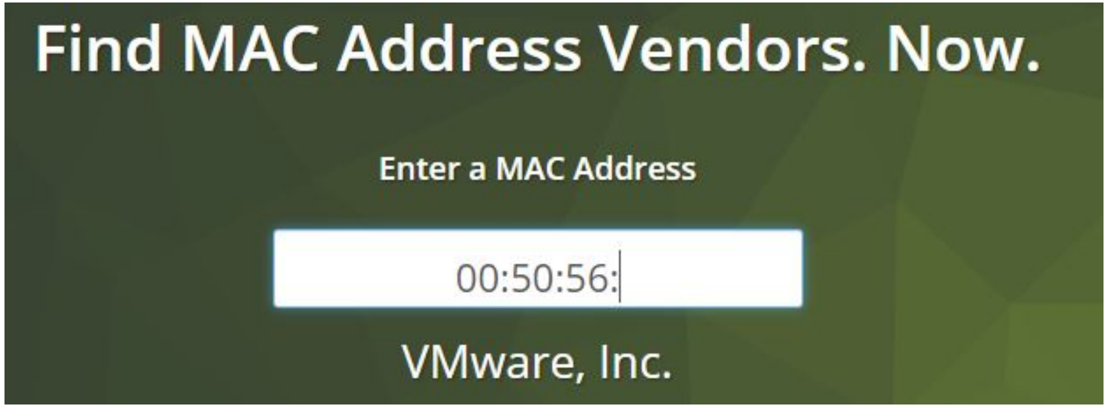
Anti-VM Techniques

- Registry Query
  - HKLM\\SYSTEM\\CurrentControlSet\\Control\\VirtualDeviceDrivers

- VM-specific, but works against common VMs (VMWare, VirtualBox)

Anti-VM Techniques

- MAC Address (NetworkAddress) in registry
  - VMWare machines 00:50:56 :XX:YY:ZZ
  - Organisationally-Unique Identifiers (OUIs)



Find MAC Address Vendors. Now.

Enter a MAC Address

00:50:56:

VMware, Inc.

PACKER DETECTION

UPX Detection

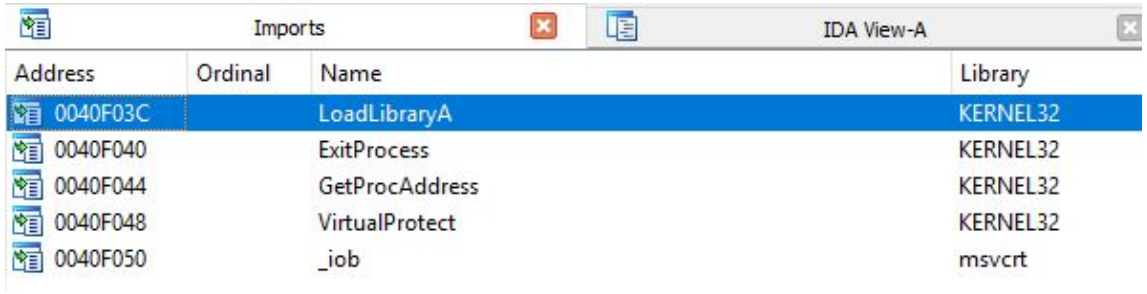
Packer Detection

- Is the file packed?
  - Import table and entropy

- What is the file packed with?
  - Byte string matching
  - Attempt to unpack

Is the file packed?

- Import table



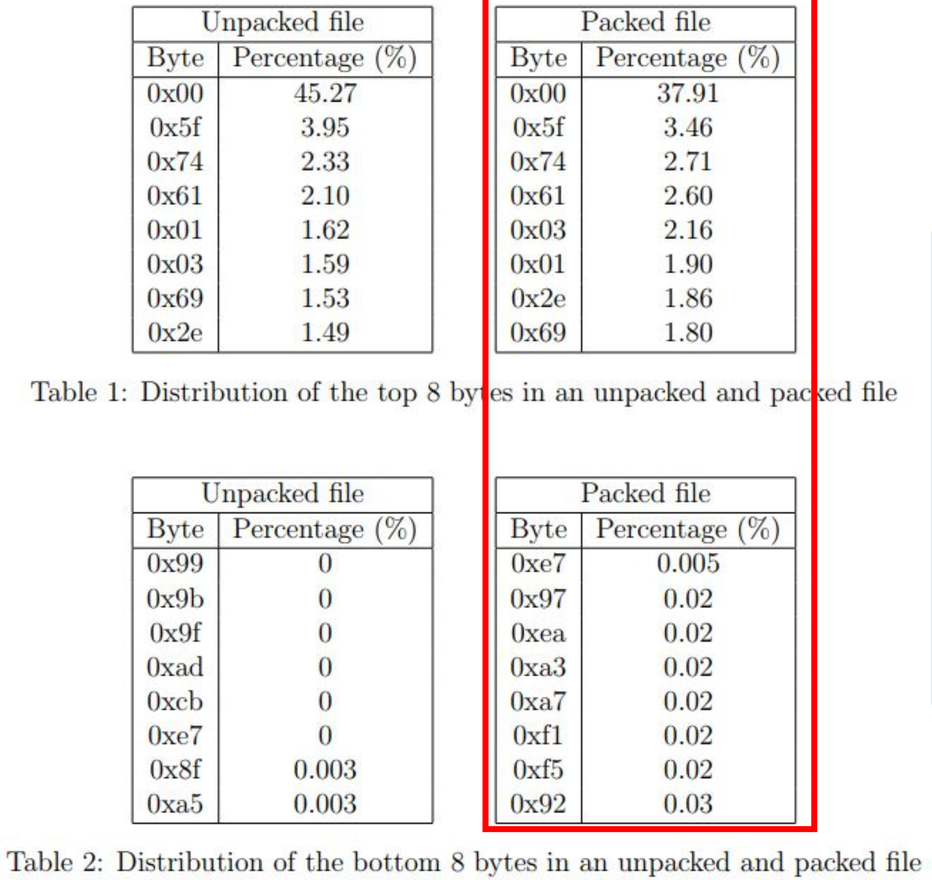
Address	Ordinal	Name	Library
0040F03C		LoadLibraryA	KERNEL32
0040F040		ExitProcess	KERNEL32
0040F044		GetProcAddress	KERNEL32
0040F048		VirtualProtect	KERNEL32
0040F050		_iob	msvcrt

  - LoadLibraryA, GetProcAddress
- Tools: IDA Pro

Is the file packed?

- Entropy
  - Byte distribution

- Tool: Detect It Easy



Unpacked file	
Byte	Percentage (%)
0x00	45.27
0x5f	3.95
0x74	2.33
0x61	2.10
0x01	1.62
0x03	1.59
0x69	1.53
0x2e	1.49

Packed file	
Byte	Percentage (%)
0x00	37.91
0x5f	3.46
0x74	2.71
0x61	2.60
0x03	2.16
0x01	1.90
0x2e	1.86
0x69	1.80

Table 1: Distribution of the top 8 bytes in an unpacked and packed file

Unpacked file	
Byte	Percentage (%)
0x99	0
0x9b	0
0x9f	0
0xad	0
0xcb	0
0xe7	0
0x8f	0.003
0xa5	0.003

Packed file	
Byte	Percentage (%)
0xe7	0.005
0x97	0.02
0xea	0.02
0xa3	0.02
0xa7	0.02
0xf1	0.02
0xf5	0.02
0x92	0.03

Table 2: Distribution of the bottom 8 bytes in an unpacked and packed file

Packed file has a more even byte distribution

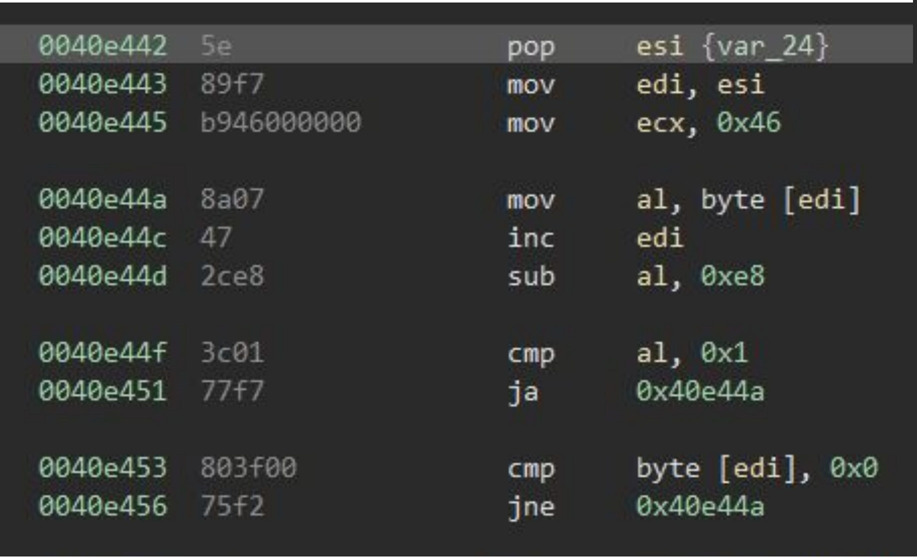
What is the file packed with?

- Byte string matching
  - Patterns in unpacking code
- Tool: PEiD
  - Closed source: external database was used

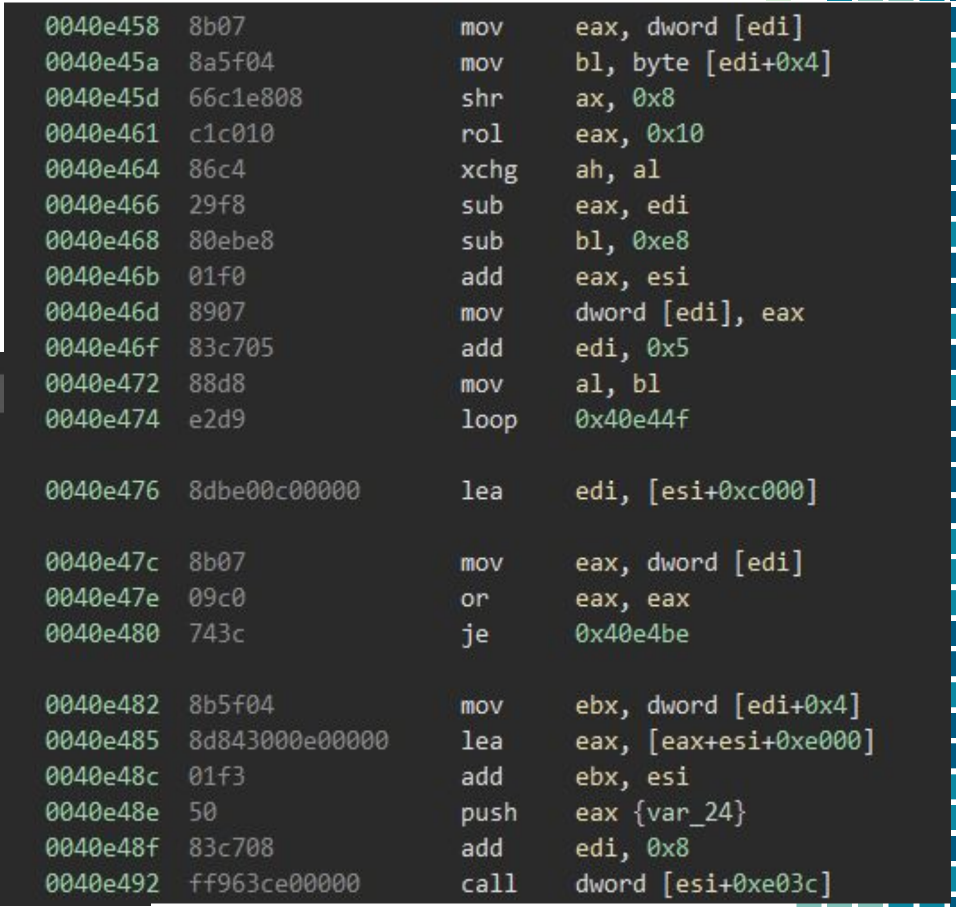
Byte String Matching

```
[UPX 2.00-3.0X -> Markus Oberhumer & Laszlo Molnar & John Reiser]

signature = 5E 89 F7 B9 ?? ?? ?? ?? 8A 07 47 2C E8 3C 01 77 F7 80 3F ?? 75 F2 8B 07 8A 5F 04 66 C1
E8 08 C1 C0 10 86 C4 29 F8 80 EB E8 01 F0 89 07 83 C7 05 88 D8 E2 D9 8D ?? ?? ?? ?? ?? 8B 07 09 C0
74 3C 8B 5F 04 8D ?? ?? ?? ?? ?? ?? 01 F3 50 83 C7 08 FF ?? ?? ?? ?? ?? 95 8A 07 47 08 C0 74 DC 89
F9 57 48 F2 AE 55 FF ?? ?? ?? ?? ?? 09 C0 74 07 89 03 83 C3 04 EB E1 FF ?? ?? ?? ?? ?? 8B
AE ?? ?? ?? ?? 8D BE 00 F0 FF FF BB 00 10 00 00 50 54 6A 04 53 57 FF D5 8D 87 ?? ?? ?? ?? 80 20 7F
80 60 28 7F 58 50 54 50 53 57 FF D5 58 61 8D 44 24 80 6A 00 39 C4 75 FA 83 EC 80 E9

ep_only = false
```

Linear Disassembly



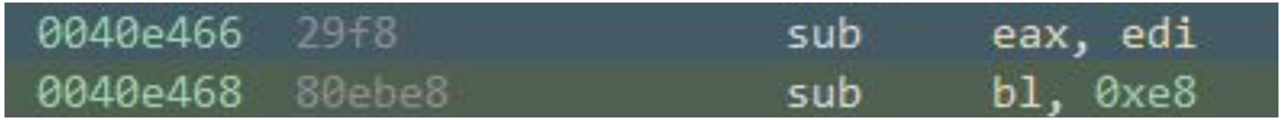
```
0040e442  5e            pop   esi {var_24}
0040e443  89f7          mov   edi, esi
0040e445  b946000000    mov   ecx, 0x46

0040e44a  8a07          mov   al, byte [edi]
0040e44c  47            inc   edi
0040e44d  2ce8          sub   al, 0xe8

0040e44f  3c01          cmp   al, 0x1
0040e451  77f7          ja    0x40e44a

0040e453  803f00        cmp   byte [edi], 0x0
0040e456  75f2          jne   0x40e44a
```



```
0040e458  8b07          mov   eax, dword [edi]
0040e45a  8a5f04        mov   bl, byte [edi+0x4]
0040e45d  66c1e808      shr   ax, 0x8
0040e461  c1c010        rol   eax, 0x10
0040e464  86c4          xchg  ah, al
0040e466  29f8          sub   eax, edi
0040e468  80ebe8        sub   bl, 0xe8
0040e46b  01f0          add   eax, esi
0040e46d  8907          mov   dword [edi], eax
0040e46f  83c705        add   edi, 0x5
0040e472  88d8          mov   al, bl
0040e474  e2d9          loop  0x40e44f

0040e476  8dbe00c00000  lea   edi, [esi+0xc000]

0040e47c  8b07          mov   eax, dword [edi]
0040e47e  09c0          or    eax, eax
0040e480  743c          je    0x40e4be

0040e482  8b5f04        mov   ebx, dword [edi+0x4]
0040e485  8d843000e00000  lea   eax, [eax+esi+0xe000]
0040e48c  01f3          add   ebx, esi
0040e48e  50            push  eax {var_24}
0040e48f  83c708        add   edi, 0x8
0040e492  ff963ce00000  call  dword [esi+0xe03c]
```

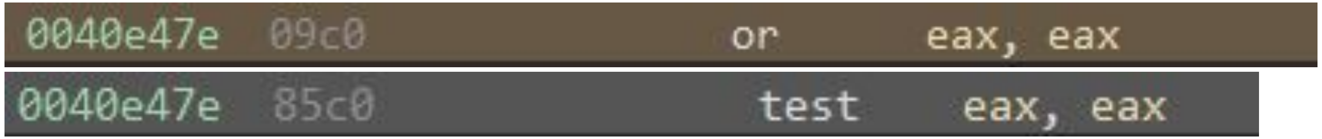
Byte String Matching

- Re-order instructions
  - Ensure no dependencies



```
0040e466  29f8        sub    eax, edi
0040e468  80ebe8      sub    bl, 0xe8
```

- Replace with equivalent instructions



```
0040e47e  09c0        or     eax, eax
0040e47e  85c0        test   eax, eax
```

Byte String Matching

- PEiD still able to detect as UPX-packed
  - Could have updated database with variants of the entries

- More complicated mechanisms to check

What is the file packed with?

- Attempt to unpack file
  - Requires specific unpacking algorithm
- UPX: Renaming sections makes files unpackable by default algorithm

Summary

Challenges and Learning

Future Work

- Create tools to automatically generate permutations of changing byte strings
- Reverse engineer PEiD to examine and evade its packer detection techniques

Difficulties Faced and Learning

- Outdated references (> 10 years ago)
  - Evolving operating systems
- Gained better understanding of Windows tools
  - Sysinternals, Windows programming, kernel debugging, packers
  - Evolving arms race in malware development
  - Exploiting implementation details

THANKS!

Any questions?