Teoria e Prática de Ataques de Segurança

2025/2026

André Baptista
andre.baptista@fc.up.pt

Miguel Regala
miguel.regala@fc.up.pt

https://tpas.alunos.dcc.fc.up.pt

FACULDADE DE CIÊNCIAS
UNIVERSIDADE DO PORTO

Class 7

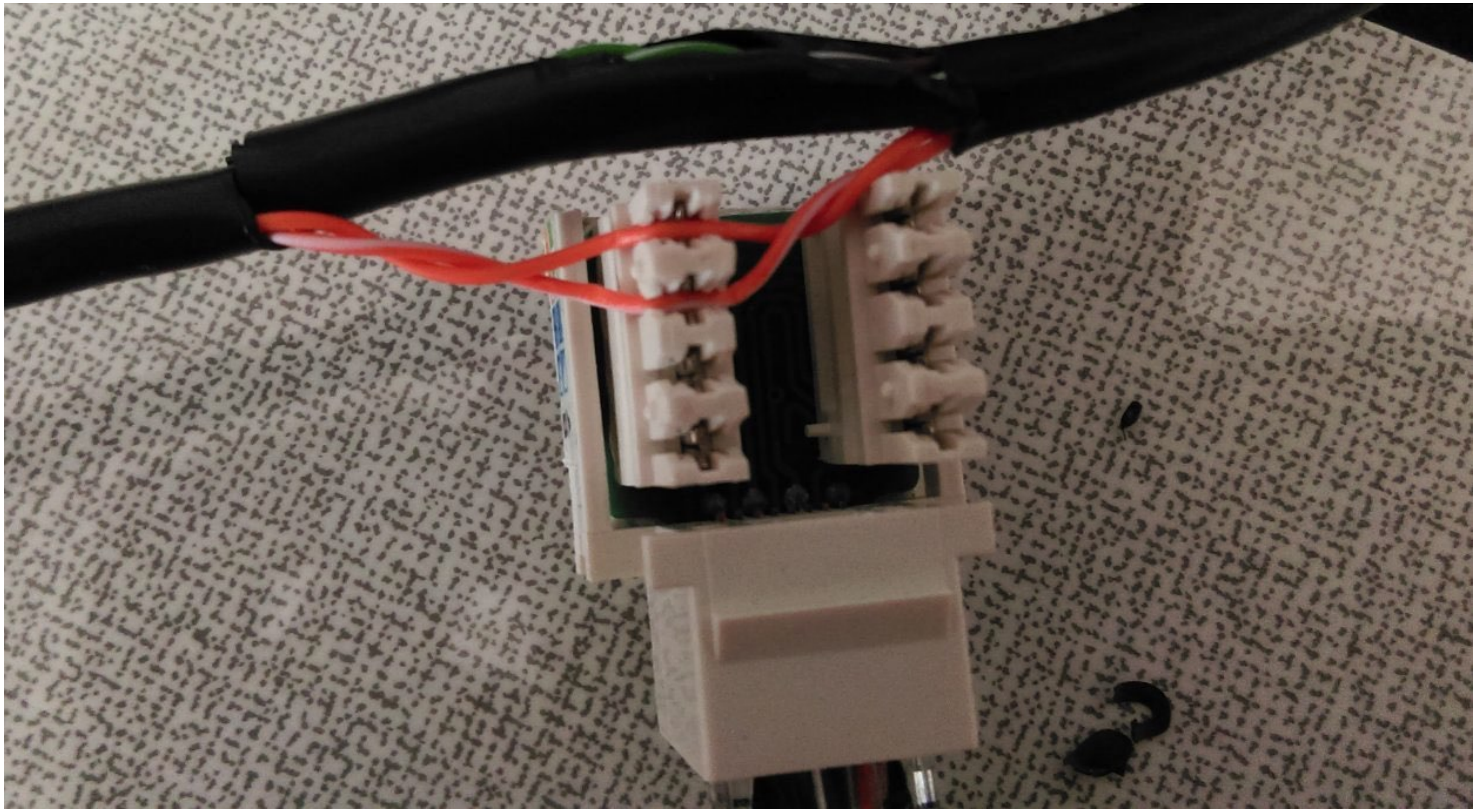
Part 1: Advanced Sniffing

FACULDADE DE CIÊNCIAS
UNIVERSIDADE DO PORTO

Wiretapping

- Wiretapping is about listening to a communication channel used by entities exchanging information.
- Attackers usually connect a device to sniff the communication channel in order to collect sensitive information (passive).
- Active wiretapping allows attackers to inject and tamper data.
- Wiretapping has been used by many countries, as a way to monitor critical infrastructures (national security).

Wiretapping



From: lastbreach.com

Packet Sniffing

- Packet Sniffing is about capturing packets generated by legitimate entities within a network or communication channel, physical or over-the-air.
- Allows attackers to observe communications (non-encrypted/plain text is easy to retrieve)
- We only need access to a communication channel, network interface etc:
  - Unprotected WiFi networks are easy
  - On password-protected wireless networks it's necessary to have network access (cracking the key or password)
  - Physical networks: wiretapping or ethernet port access

Active Sniffing

- We can use multiple techniques to make active sniffing more efficient:
  - MAC flooding: flood a switch with new MAC addresses so that filtering is not done in an efficient manner.
  - Rogue DHCP: act as a DHCP server and broadcast a malicious router/gateway address.
  - DNS poisoning: redirect machines to malicious proxies
  - ARP poisoning: spoof the IP address of the victim and associate it with our MAC address

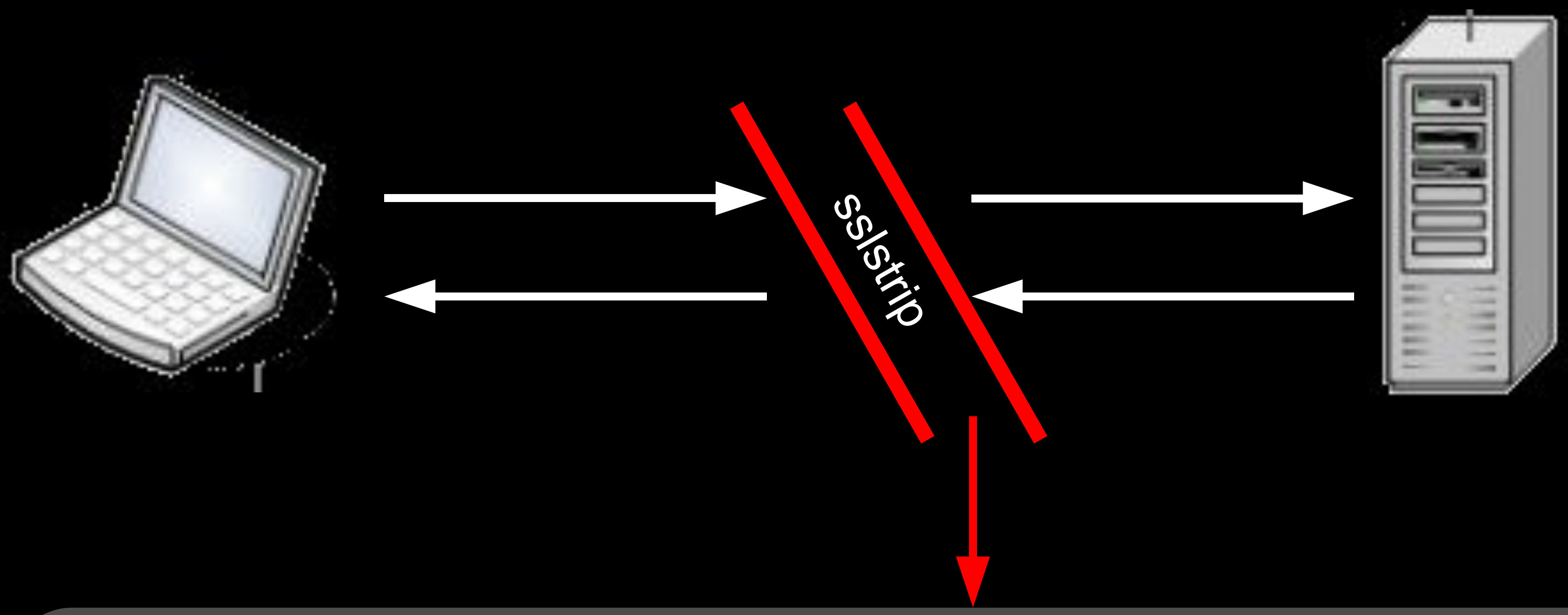
Vulnerable protocols

- All protocols lacking encryption (or using weak encryption) to protect sensitive information are vulnerable:
    - Telnet
    - rlogin
    - HTTP
    - SNMP
    - POP
    - IMAP

SSL Strip

- When we visit an URL manually, nobody writes in the address bar https://domain.com
- SSL is usually enforced by clicking links or location redirects.
- SSL Strip is about attacking an HTTP connection, usually through ARP spoofing (our MAC address as the router MAC address).
- Nowadays, some browsers force HTTPS by default for websites we have already visited. HTTPS Everywhere extension is helpful to increase our security.
- Mitigation: HSTS
- https://github.com/moxie0/sslstrip

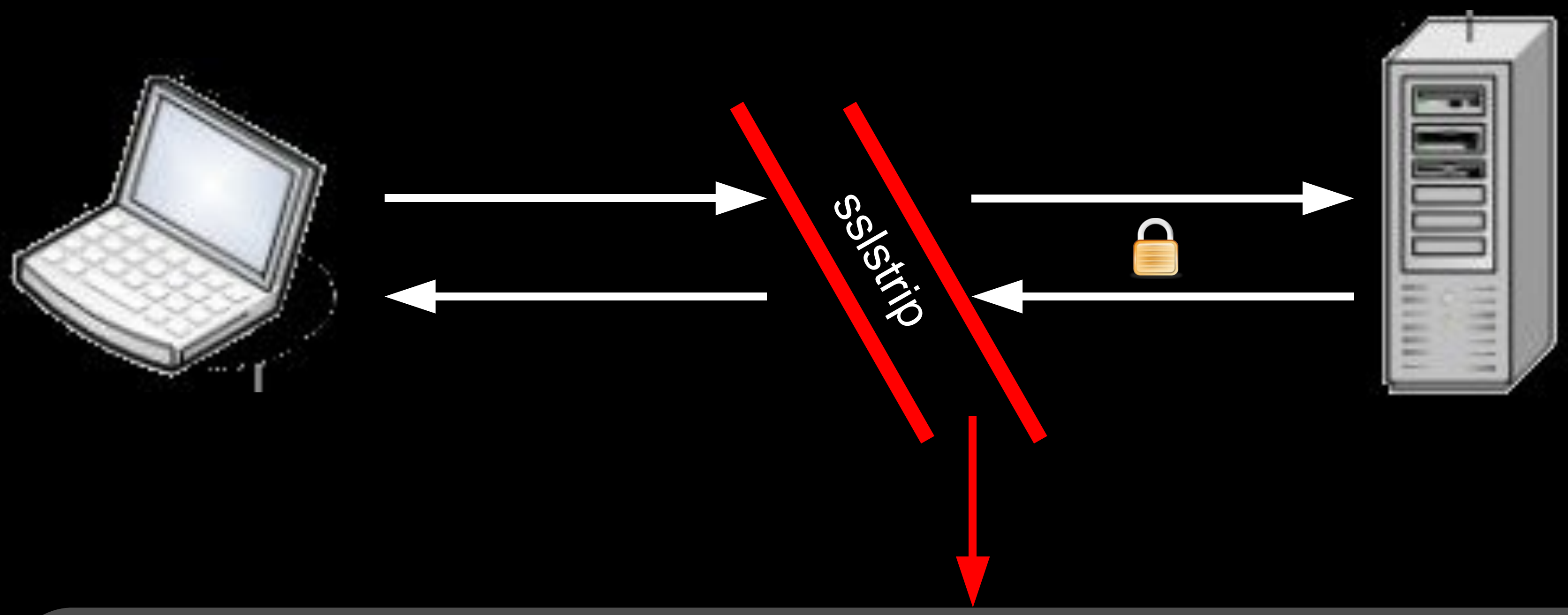
A First Cut Recipe: sslstrip



Watch HTTP traffic go by.
Switch <a href=”https://...”> to <a href=”http://...”> and keep a map of what's changed.
Switch Location: https://... to Location: http://... and keep a map of what's changed.

From: New Tricks For Defeating SSL In Practice, Moxie Marlinspike

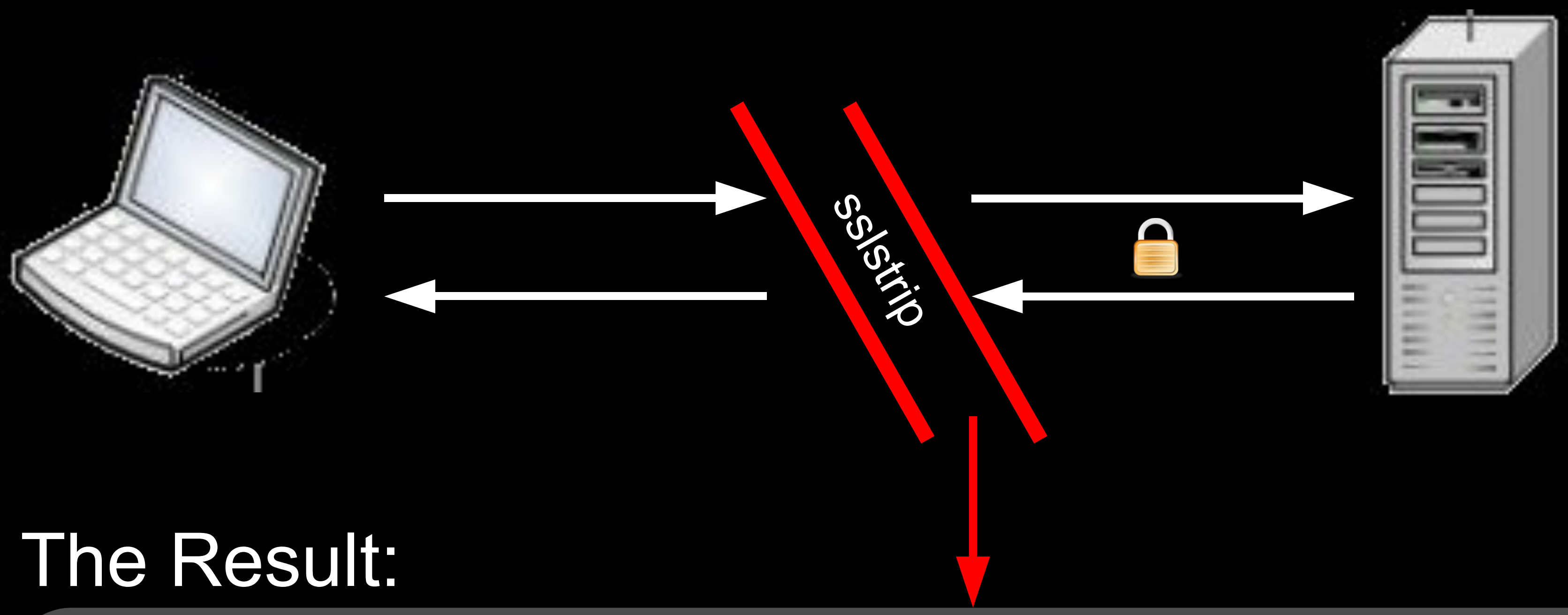
A First Cut Recipe: sslstrip



Watch HTTP traffic go by.
When we see an HTTP request for a URL that we've stripped, proxy that out as HTTPS to the server.
Watch the HTTPS traffic go by, log everything if we want, and keep a map of the relative links, CSS links, and JavaScript links that go by.

From: New Tricks For Defeating SSL In Practice, Moxie Marlinspike

A First Cut Recipe: sslstrip



The Result:

The server never knows the difference. Everything looks secure on their end.
The client doesn't display any of the disastrous warnings that we want to avoid.
We see all the traffic.

From: New Tricks For Defeating SSL In Practice, Moxie Marlinspike

Tools

- There’s special hardware for sniffing with high rates in terms of packet sniffing.
- Some tools are used for technical support teams that build and manage network infrastructures.
- Software-based tools: Wireshark, Bettercap, **`tcpdump`**, etc

MAC Flooding

- Nowadays, almost every network is *switched.*
- A switch contains a table that stores what MAC addresses are in each network segment.
- This information is used to optimize packet transmission.
- By injecting messages with new MAC addresses, these tables can be modified and the real addresses may be removed or changed.
- Switches will start retransmitting packets, allowing for a more efficient sniffing.

Rogue DHCP

- DHCP protocol works like this:
  - A client broadcasts the need for an IP address
  - The server answers with an IP address offer (we can have multiple servers with different offers)
  - Clients can pick one of the available offers
  - The server will answer with an acknowledgment
- A rogue DHCP server can persuade a client that the router/gateway has an attacker-controlled IP address, allowing for MITM attacks.

ARP Poisoning

- ARP protocol allows translating IP addresses to MAC addresses within a local network.
- When a machine doesn't know the MAC address of another machine it wants to communicate with, an ARP request is performed.
- The target machine will answer with its IP and MAC addresses.
- We can flood the network with ARP responses that makes our MAC address match a given set of IPs we want to attack.
- This allows MITM attacks.
- Tool: **`arpspoof`** (UNIX)

MAC/IRDP Spoofing

- When sniffing reveals the MAC address of a given machine, we can try to steal it.
- To make this work, we can configure our network interface to use this address:

```
macchanger -r INTERFACE --mac=XX.XX.XX.XX.XX.XX
```

- Our interface will start transmitting and receiving information like if it was the target.
- ICMP Router Discovery Protocol (IRDP) allows the discovery of a router within a network.
- If we broadcast a fake router, we can make other machines in the network using a new gateway controlled by us.

DNS Poisoning/Spoofing

- Simple concept:
  - We make the target use a fake, attacker-controlled DNS server
  - We can broadcast fake IP addresses that will resolve domains to attacker-controlled machines.
  - We can trick users by showing services that look legitimate (phishing, etc)
  - We can then extract sensitive information e.g. passwords
- We can use ARP poisoning to steal the legitimate DNS server IP address.

Class 7

Part 2: Malware Analysis

FACULDADE DE CIÊNCIAS
UNIVERSIDADE DO PORTO

Malware Analysis

- Motivation:
    - Attribution (tracking the attackers)
    - Understanding the complexity of an attack
    - Creating signatures and patterns for detection
    - Understanding how to remove / fix / revert the infection

Secure environment

- We don’t want to run the malware on our machines.
- “Old-school” method: Creating a physical space with multiple machines and an isolated network.
- Recent methods rely on virtualization:
  - VMware
  - Parallels
  - Microsoft Virtual PC
  - VirtualBox
  - UTM
- We can create disk snapshots before running the malware for analysis.

Secure environment

https://www.youtube.com/watch?v=NDuWcGn5hTQ

Secure environment

- Possible solutions:
    - Virtualization + "old-school" method (dedicated machines on a special physical space) and sandboxes.
    - E.g. https://cuckoosandbox.org/ - now hatching.io
    - Running malware inside a VM on a different platform (e.g. using VMware on Linux/OSX to analyze malware for Windows)
    - Save and revert snapshots as you perform analysis

Secure Environment

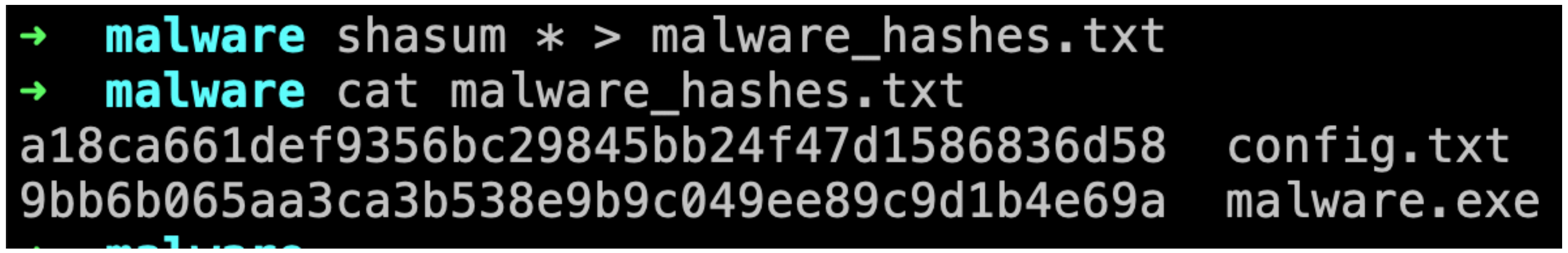
- If we allow Internet access:
  - We can eventually get to a battle for control of our VM with a real attacker (human or bot)
  - Our IP address can be prone to attacks (we can use TOR / VPN or other anonymization techniques)
- It's dangerous to allow VM access to a local network (automated exploits for spreading and poisoning other machines, sniffing, MITM attacks, and further exploitation).
- However, malware can behave differently if the network access is blocked or a sandbox is detected.

Static vs dynamic analysis

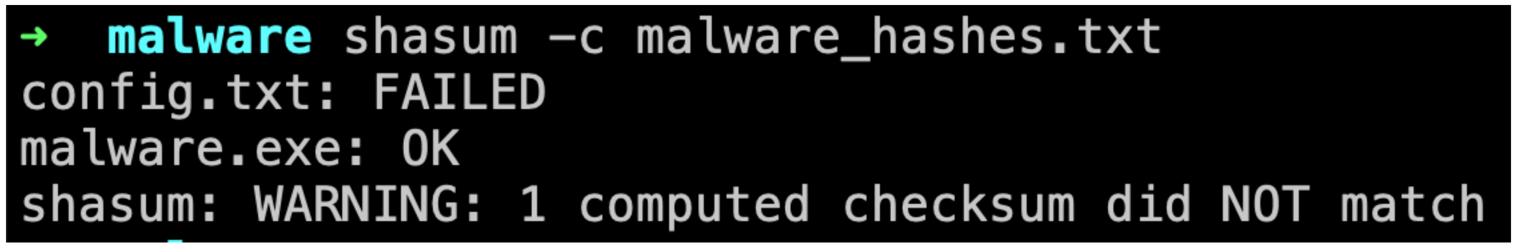
- **Static analysis:**
    - Malware is not executed.
    - We can identify packers, obfuscators, cryptors, and programming languages.
    - Identify dead code.
    - We should use disassemblers, decompilers, etc
- **Dynamic analysis:**
    - Observing and controlling malware in runtime
    - We should use debuggers, proxies, traffic analysis tools, identify C&C servers, changes in the filesystem (read/write/delete), Windows registry
    - We can observe the process memory to identify strings and keys
- Best solution: Use both analysis techniques

Static analysis

- Since we are not running the malware, we don't need to set up a secure environment.
- First step:
    - Store malware hashes
    - After performing dynamic analysis, observe if the hashes changed (e.g. mutations or configuration changes)



```
➜  malware shasum * > malware_hashes.txt
➜  malware cat malware_hashes.txt
a18ca661def9356bc29845bb24f47d1586836d58  config.txt
9bb6b065aa3ca3b538e9b9c049ee89c9d1b4e69a  malware.exe
```



```
➜  malware shasum -c malware_hashes.txt
config.txt: FAILED
malware.exe: OK
shasum: WARNING: 1 computed checksum did NOT match
```

Packers, *cryptors* and *protectors*

- Malware is usually protected against detection and reverse engineering techniques. Attackers tend to add code (not necessarily malicious) to hide the malware behaviour and make researching tasks harder.

- **Packers**

  - Use of compression.

  - *Runtime extractors*: when a program is executed by unpacking the legitimate binary to executable memory and jumping

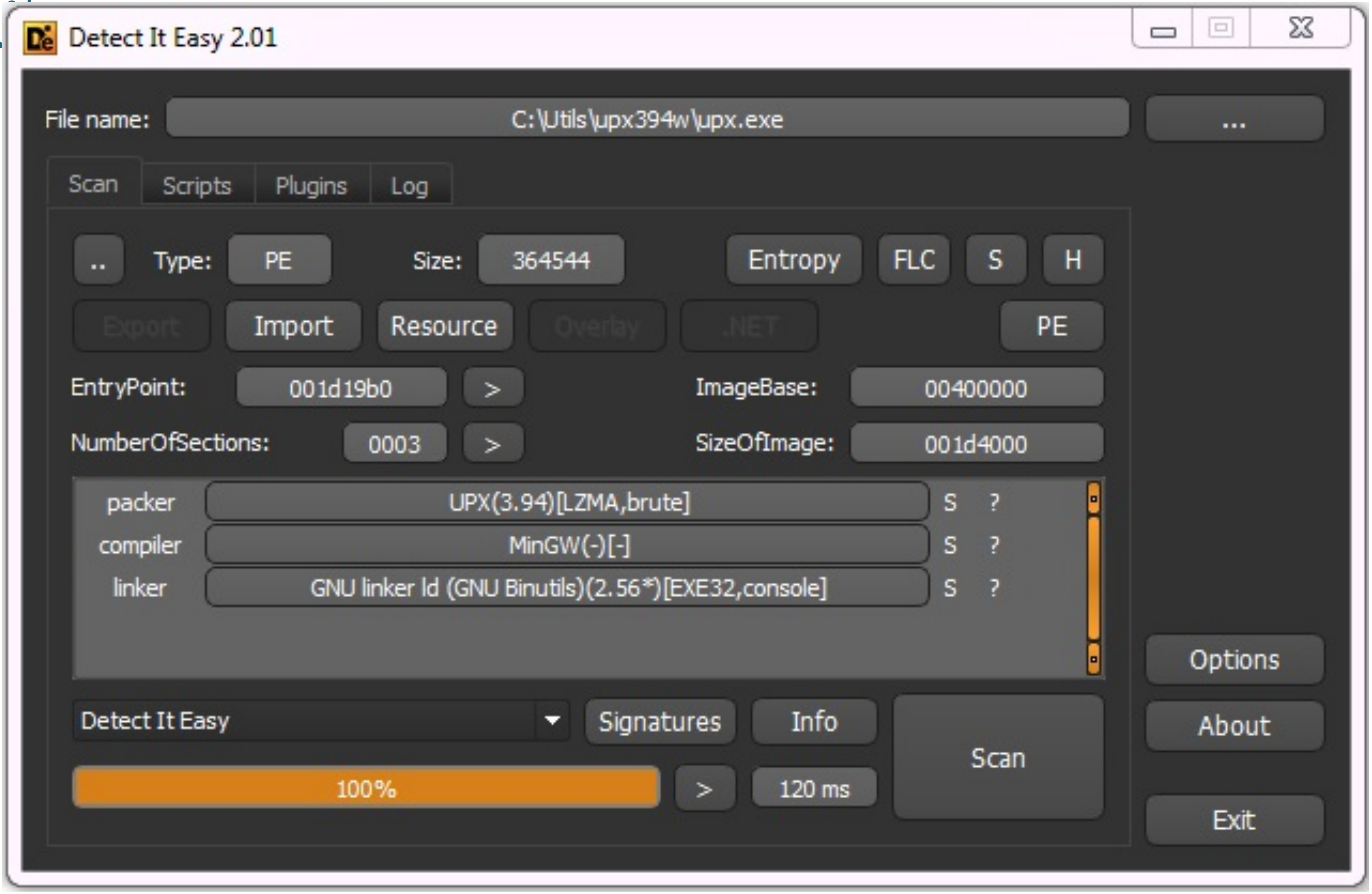
  - Allow signature bypass

Packers, *cryptors* and *protectors*

- **Cryptors**
  - Use of cryptography to decrypt code segments
  - Code obfuscation
  - FUD (Fully Undetectable) - attackers really want this
- **Protectors**
  - Packers + cryptors + anti-debugging/tampering techniques
  - Code virtualization techniques (e.g. WProtect)

Identification

- ~~https://www.packerinspector.com/ (offline)~~
- Detect It Easy
- PEstudio
- Exeinfo PE
- PEiD (deprecated, but contains more than 470 signatures for packer, cryptor and compiler detection)


Detect It Easy 2.01

File name: C:\Utils\upx394w\upx.exe

Type: PE   Size: 364544

EntryPoint: 001d19b0   ImageBase: 00400000

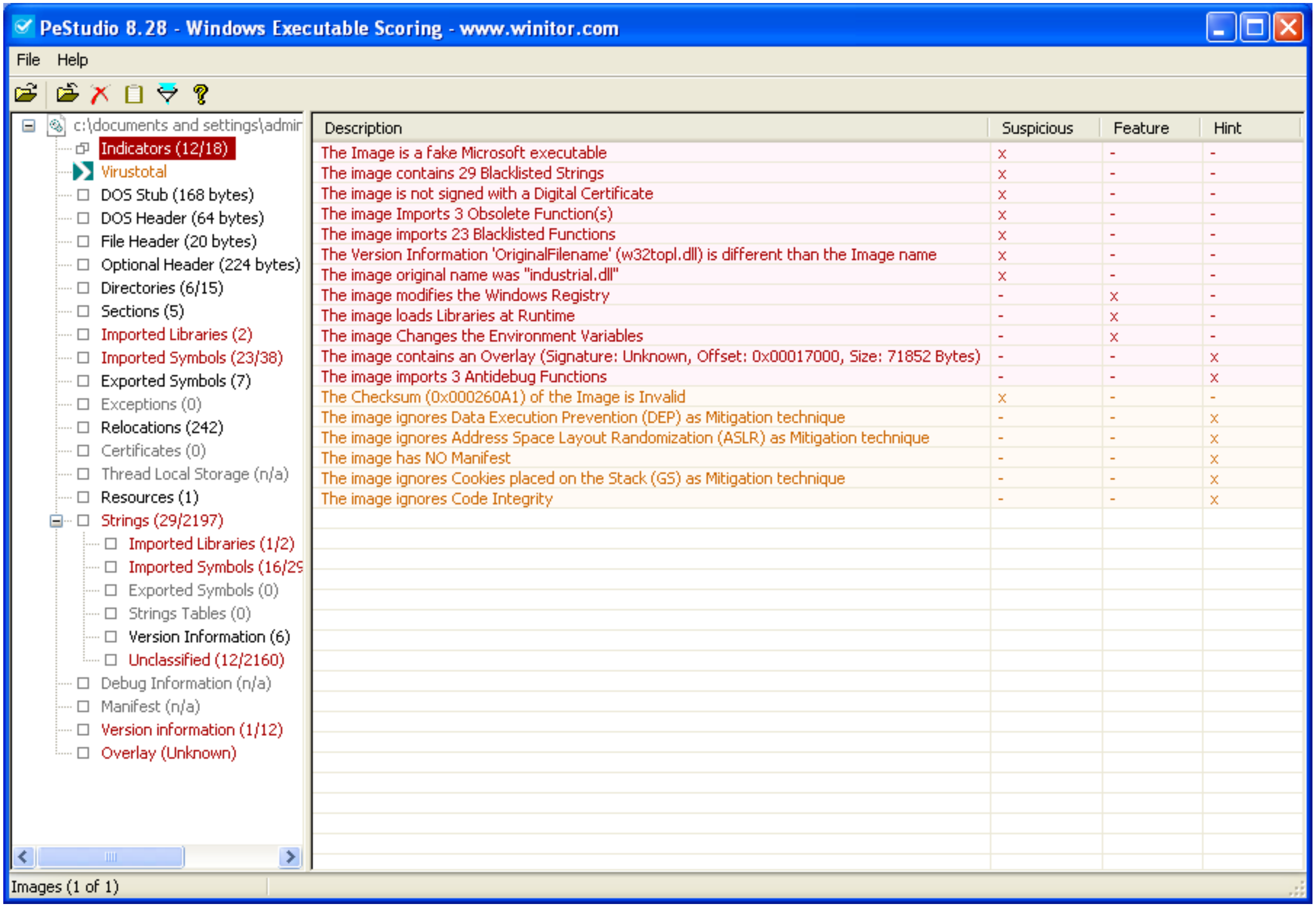
NumberOfSections: 0003   SizeOfImage: 001d4000

```
packer    UPX(3.94)[LZMA,brute]
compiler  MinGW(-)[-]
linker    GNU linker ld (GNU Binutils)(2.56*)[EXE32,console]
```

Detect It Easy

100%   120 ms

From: ntinfo.biz

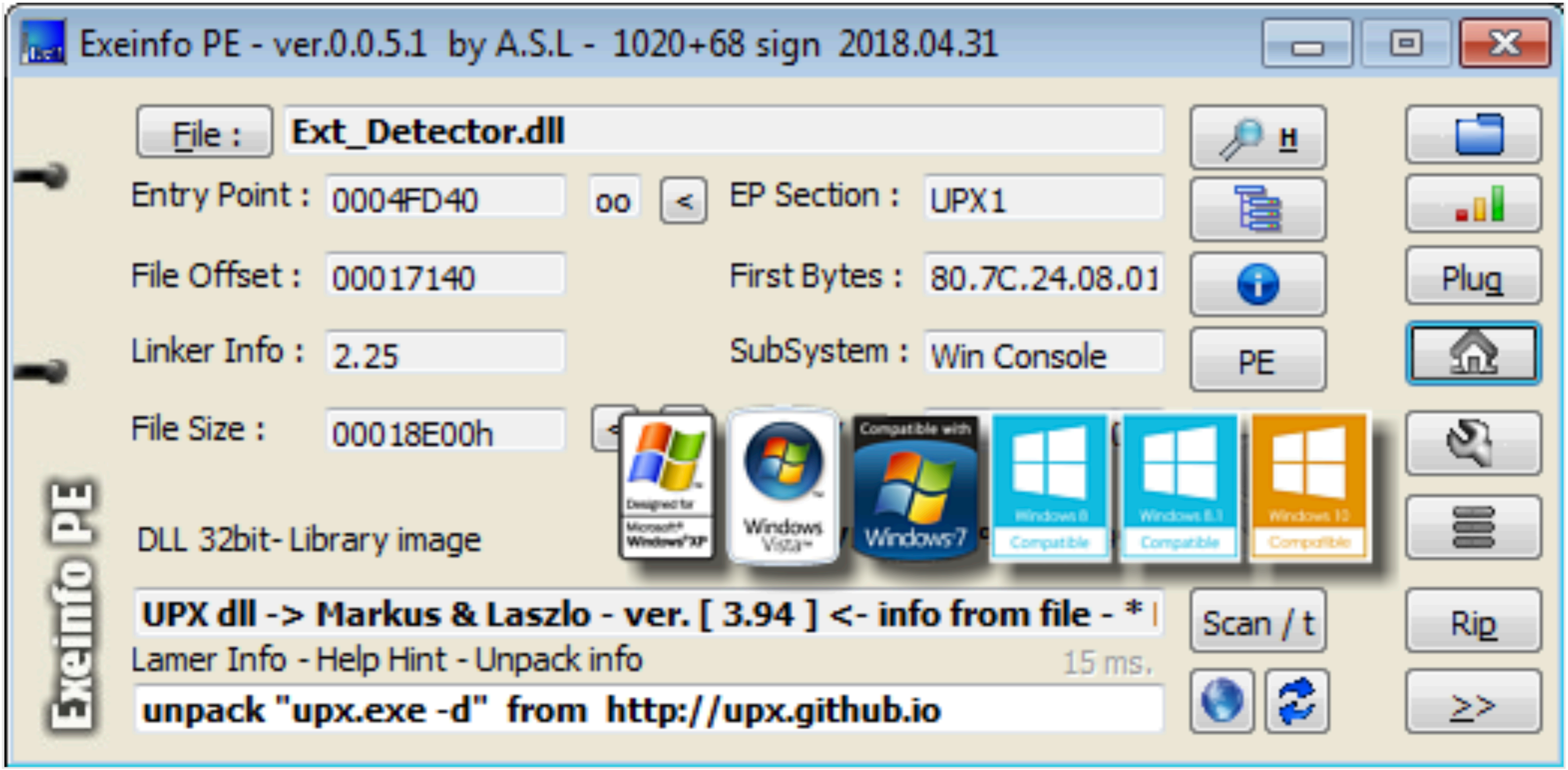

PeStudio 8.28 - Windows Executable Scoring - www.winitor.com

```
c:\documents and settings\admir
  Indicators (12/18)
  Virustotal
  DOS Stub (168 bytes)
  DOS Header (64 bytes)
  File Header (20 bytes)
  Optional Header (224 bytes)
  Directories (6/15)
  Sections (5)
  Imported Libraries (2)
  Imported Symbols (23/38)
  Exported Symbols (7)
  Exceptions (0)
  Relocations (242)
  Certificates (0)
  Thread Local Storage (n/a)
  Resources (1)
  Strings (29/2197)
    Imported Libraries (1/2)
    Imported Symbols (16/29
    Exported Symbols (0)
    Strings Tables (0)
    Version Information (6)
    Unclassified (12/2160)
  Debug Information (n/a)
  Manifest (n/a)
  Version information (1/12)
  Overlay (Unknown)
```

Description	Suspicious	Feature	Hint
The Image is a fake Microsoft executable	x	-	-
The image contains 29 Blacklisted Strings	x	-	-
The image is not signed with a Digital Certificate	x	-	-
The image Imports 3 Obsolete Function(s)	x	-	-
The image imports 23 Blacklisted Functions	x	-	-
The Version Information 'OriginalFilename' (w32topl.dll) is different than the Image name	x	-	-
The image original name was "industrial.dll"	x	-	-
The image modifies the Windows Registry	-	x	-
The image loads Libraries at Runtime	-	x	-
The image Changes the Environment Variables	-	x	-
The image contains an Overlay (Signature: Unknown, Offset: 0x00017000, Size: 71852 Bytes)	-	-	x
The image imports 3 Antidebug Functions	-	-	x
The Checksum (0x000260A1) of the Image is Invalid	x	-	-
The image ignores Data Execution Prevention (DEP) as Mitigation technique	-	-	x
The image ignores Address Space Layout Randomization (ASLR) as Mitigation technique	-	-	x
The image has NO Manifest	-	-	x
The image ignores Cookies placed on the Stack (GS) as Mitigation technique	-	-	x
The image ignores Code Integrity	-	-	x

Images (1 of 1)

From: Malwarebytes blog



File : Ext_Detector.dll

Entry Point : 0004FD40 oo <

EP Section : UPX1

File Offset : 00017140

First Bytes : 80.7C.24.08.01

Linker Info : 2.25

SubSystem : Win Console

File Size : 00018E00h

DLL 32bit- Library image

UPX dll -> Markus & Laszlo - ver. [3.94] <- info from file - *

Lamer Info - Help Hint - Unpack info 15 ms.

unpack "upx.exe -d" from http://upx.github.io

From: exeinfo.atwebpages.com

Strings

- We can look for strings in the binary or memory.
- Unicode strings? `strings -u`
- Other tools exist: bintext, `xxd`, Hex Fiend, disassemblers
- Can identify or reveal interesting information about the attackers.
- Attackers can include special strings to deceive researchers.

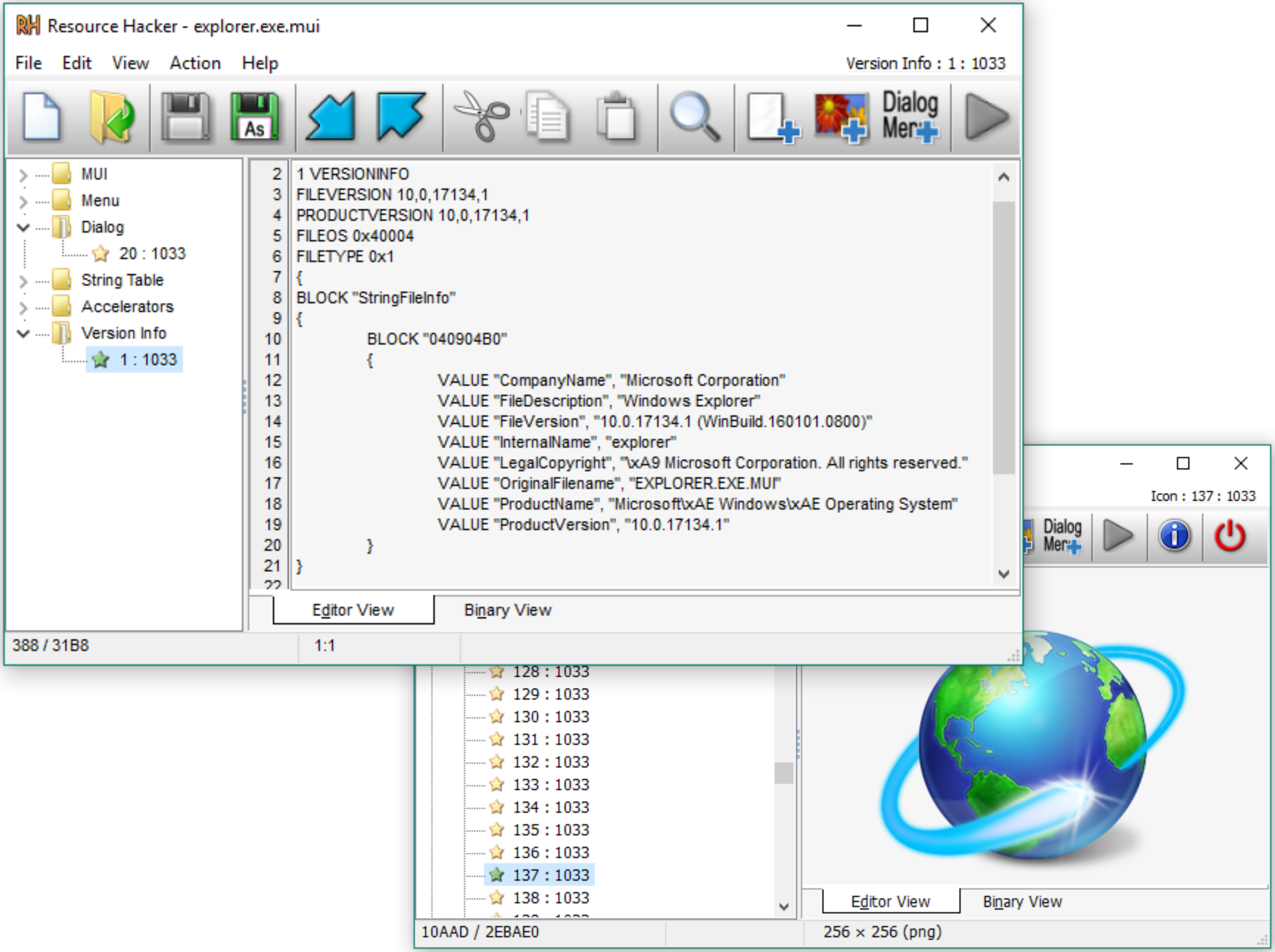
Strings

```
$ strings unknown2.exe
...
 <host> <port>
 -install <host> <port>
 -remove
EC.1
EC.2
cmd.exe
connect thread started!
...
```

From: Practical Malware Analysis, Kendall McMillan

Research

- We can obtain strings, email addresses, network information (IP addresses, hostnames, etc) and more data.
- We can search online for the malware, verify if there are any victims asking for help in forums, malware databases, news, etc.
- Sometimes we need to use translating services (Chinese, Russian, etc).

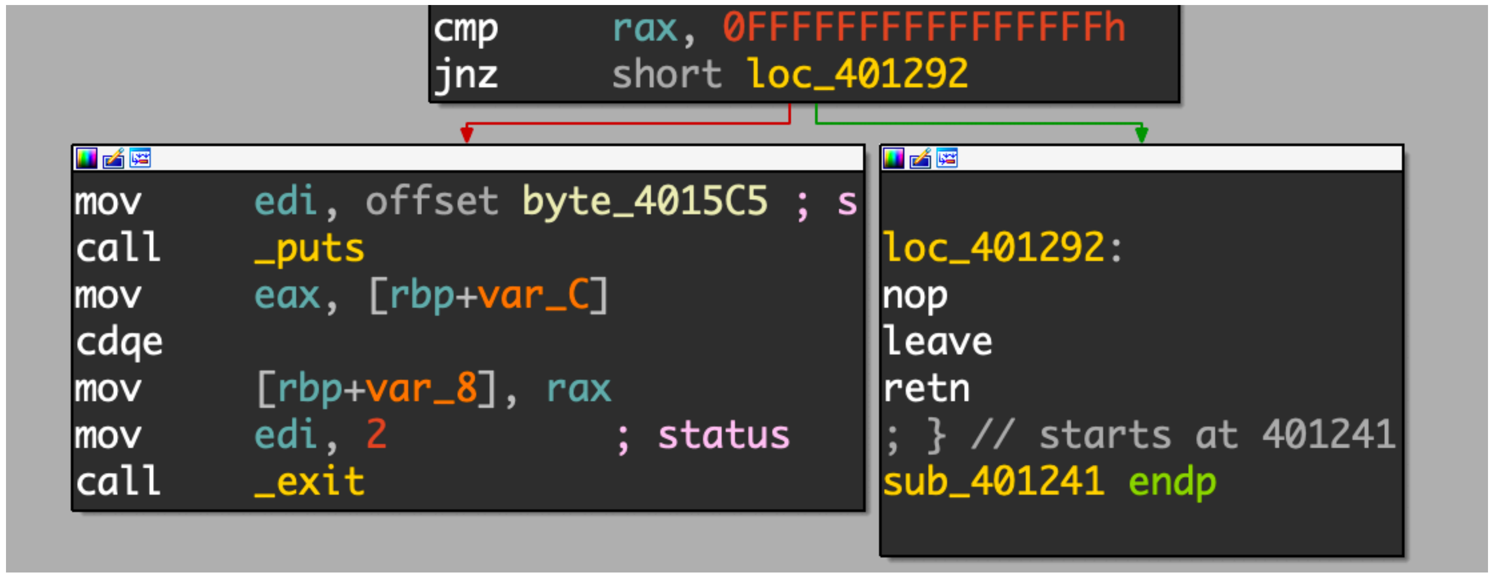


```
1 VERSIONINFO
FILEVERSION 10,0,17134,1
PRODUCTVERSION 10,0,17134,1
FILEOS 0x40004
FILETYPE 0x1
{
BLOCK "StringFileInfo"
{
		BLOCK "040904B0"
		{
				VALUE "CompanyName", "Microsoft Corporation"
				VALUE "FileDescription", "Windows Explorer"
				VALUE "FileVersion", "10.0.17134.1 (WinBuild.160101.0800)"
				VALUE "InternalName", "explorer"
				VALUE "LegalCopyright", "\xA9 Microsoft Corporation. All rights reserved."
				VALUE "OriginalFilename", "EXPLORER.EXE.MUI"
				VALUE "ProductName", "Microsoft\xAE Windows\xAE Operating System"
				VALUE "ProductVersion", "10.0.17134.1"
		}
}
```

Static analysis

- We can use disassemblers such as Ghidra, IDA Pro, Hopper, radare2, etc and specific decompilers (according the programming language identified in the sample).

- You can find more information and examples in the reverse engineering class.



```
cmp     rax, 0FFFFFFFFFFFFFFFFh
jnz     short loc_401292
```

```
mov     edi, offset byte_4015C5 ; s
call    _puts
mov     eax, [rbp+var_C]
cdqe
mov     [rbp+var_8], rax
mov     edi, 2          ; status
call    _exit
```

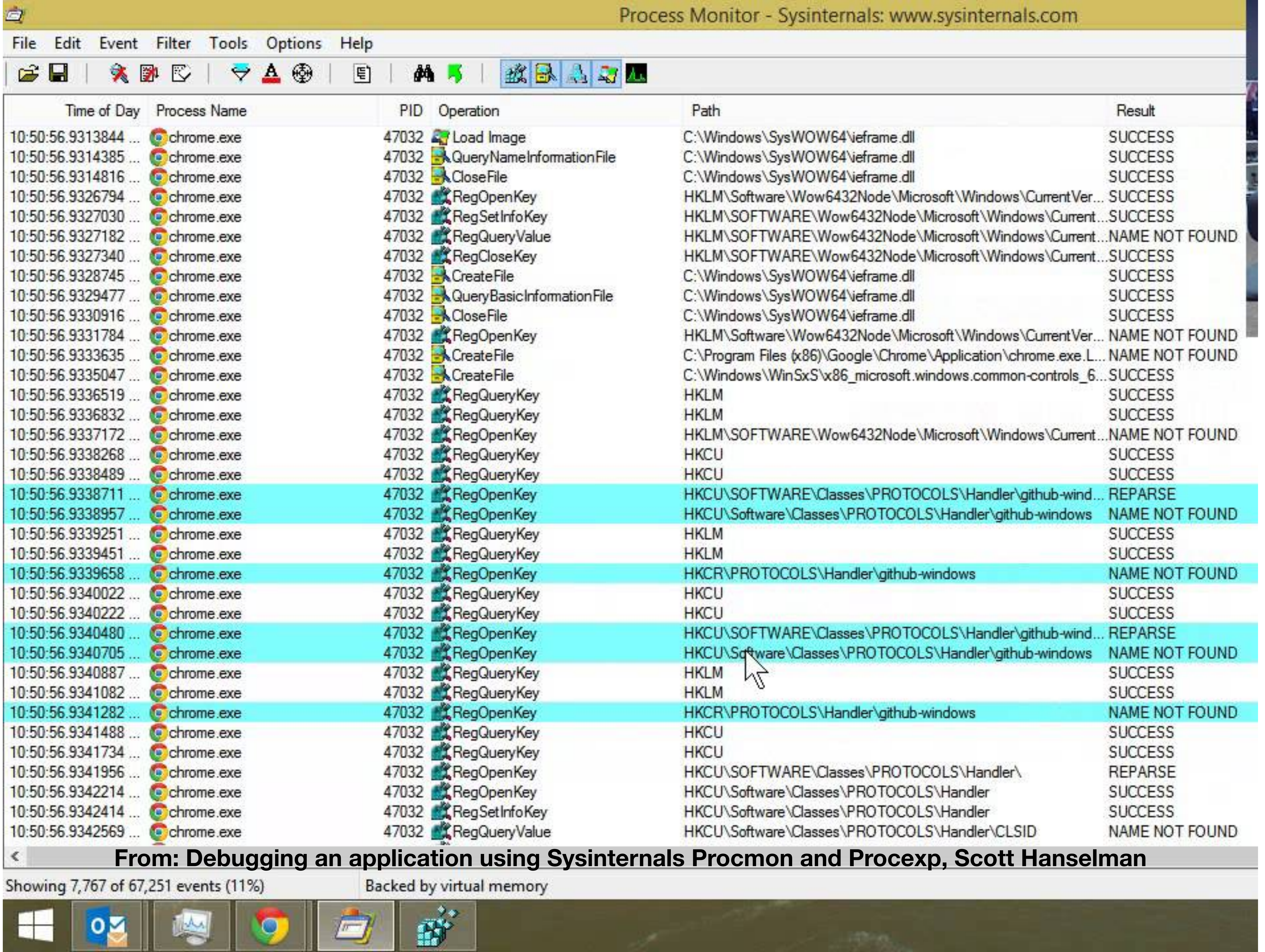
```
loc_401292:
nop
leave
retn
; } // starts at 401241
sub_401241 endp
```

Dynamic analysis

- Static analysis can reveal a lot of information, but it’s usually slow and hard.
- We should look at the malware behaviour by executing it
- Sometimes, it’s more important to understand **what** the malware is doing and not **how** it’s being done.
- Secure environment setup required.

OS and network monitoring

- We should understand if there's any activity on:
  - Files
  - Registry (Windows)
  - Processes
  - Network traffic
- Some tools:
  - Sysinternals Process Monitor
  - https://github.com/Rurik/Noriben
  - https://processhacker.sourceforge.io/
  - Wireshark
  - More info: https://github.com/rshipp/awesome-malware-analysis


Process Monitor - Sysinternals: www.sysinternals.com
File Edit Event Filter Tools Options Help

```
Time of Day         Process Name   PID    Operation                   Path                                                                  Result
10:50:56.9313844 ... chrome.exe    47032  Load Image                  C:\Windows\SysWOW64\ieframe.dll                                       SUCCESS
10:50:56.9314385 ... chrome.exe    47032  QueryNameInformationFile    C:\Windows\SysWOW64\ieframe.dll                                       SUCCESS
10:50:56.9314816 ... chrome.exe    47032  CloseFile                   C:\Windows\SysWOW64\ieframe.dll                                       SUCCESS
10:50:56.9326794 ... chrome.exe    47032  RegOpenKey                  HKLM\Software\Wow6432Node\Microsoft\Windows\CurrentVer...             SUCCESS
10:50:56.9327030 ... chrome.exe    47032  RegSetInfoKey               HKLM\SOFTWARE\Wow6432Node\Microsoft\Windows\Current...                SUCCESS
10:50:56.9327182 ... chrome.exe    47032  RegQueryValue               HKLM\SOFTWARE\Wow6432Node\Microsoft\Windows\Current...                NAME NOT FOUND
10:50:56.9327340 ... chrome.exe    47032  RegCloseKey                 HKLM\SOFTWARE\Wow6432Node\Microsoft\Windows\Current...                SUCCESS
10:50:56.9328745 ... chrome.exe    47032  CreateFile                  C:\Windows\SysWOW64\ieframe.dll                                       SUCCESS
10:50:56.9329477 ... chrome.exe    47032  QueryBasicInformationFile   C:\Windows\SysWOW64\ieframe.dll                                       SUCCESS
10:50:56.9330916 ... chrome.exe    47032  CloseFile                   C:\Windows\SysWOW64\ieframe.dll                                       SUCCESS
10:50:56.9331784 ... chrome.exe    47032  RegOpenKey                  HKLM\Software\Wow6432Node\Microsoft\Windows\CurrentVer...             NAME NOT FOUND
10:50:56.9333635 ... chrome.exe    47032  CreateFile                  C:\Program Files (x86)\Google\Chrome\Application\chrome.exe.L...      NAME NOT FOUND
10:50:56.9335047 ... chrome.exe    47032  CreateFile                  C:\Windows\WinSxS\x86_microsoft.windows.common-controls_6...          SUCCESS
10:50:56.9336519 ... chrome.exe    47032  RegQueryKey                 HKLM                                                                  SUCCESS
10:50:56.9336832 ... chrome.exe    47032  RegQueryKey                 HKLM                                                                  SUCCESS
10:50:56.9337172 ... chrome.exe    47032  RegOpenKey                  HKLM\SOFTWARE\Wow6432Node\Microsoft\Windows\Current...                NAME NOT FOUND
10:50:56.9338268 ... chrome.exe    47032  RegQueryKey                 HKCU                                                                  SUCCESS
10:50:56.9338489 ... chrome.exe    47032  RegQueryKey                 HKCU                                                                  SUCCESS
10:50:56.9338711 ... chrome.exe    47032  RegOpenKey                  HKCU\SOFTWARE\Classes\PROTOCOLS\Handler\github-wind...                REPARSE
10:50:56.9338957 ... chrome.exe    47032  RegOpenKey                  HKCU\Software\Classes\PROTOCOLS\Handler\github-windows                NAME NOT FOUND
10:50:56.9339251 ... chrome.exe    47032  RegQueryKey                 HKLM                                                                  SUCCESS
10:50:56.9339451 ... chrome.exe    47032  RegQueryKey                 HKLM                                                                  SUCCESS
10:50:56.9339658 ... chrome.exe    47032  RegOpenKey                  HKCR\PROTOCOLS\Handler\github-windows                                 NAME NOT FOUND
10:50:56.9340022 ... chrome.exe    47032  RegQueryKey                 HKCU                                                                  SUCCESS
10:50:56.9340222 ... chrome.exe    47032  RegQueryKey                 HKCU                                                                  SUCCESS
10:50:56.9340480 ... chrome.exe    47032  RegOpenKey                  HKCU\SOFTWARE\Classes\PROTOCOLS\Handler\github-wind...                REPARSE
10:50:56.9340705 ... chrome.exe    47032  RegOpenKey                  HKCU\Software\Classes\PROTOCOLS\Handler\github-windows                NAME NOT FOUND
10:50:56.9340887 ... chrome.exe    47032  RegQueryKey                 HKLM                                                                  SUCCESS
10:50:56.9341082 ... chrome.exe    47032  RegQueryKey                 HKLM                                                                  SUCCESS
10:50:56.9341282 ... chrome.exe    47032  RegOpenKey                  HKCR\PROTOCOLS\Handler\github-windows                                 NAME NOT FOUND
10:50:56.9341488 ... chrome.exe    47032  RegQueryKey                 HKCU                                                                  SUCCESS
10:50:56.9341734 ... chrome.exe    47032  RegQueryKey                 HKCU                                                                  SUCCESS
10:50:56.9341956 ... chrome.exe    47032  RegOpenKey                  HKCU\SOFTWARE\Classes\PROTOCOLS\Handler\                              REPARSE
10:50:56.9342214 ... chrome.exe    47032  RegOpenKey                  HKCU\Software\Classes\PROTOCOLS\Handler                               SUCCESS
10:50:56.9342414 ... chrome.exe    47032  RegSetInfoKey               HKCU\Software\Classes\PROTOCOLS\Handler                               SUCCESS
10:50:56.9342569 ... chrome.exe    47032  RegQueryValue               HKCU\Software\Classes\PROTOCOLS\Handler\CLSID                         NAME NOT FOUND

Showing 7,767 of 67,251 events (11%)    Backed by virtual memory
```

From: Debugging an application using Sysinternals Procmon and Procexp, Scott Hanselman

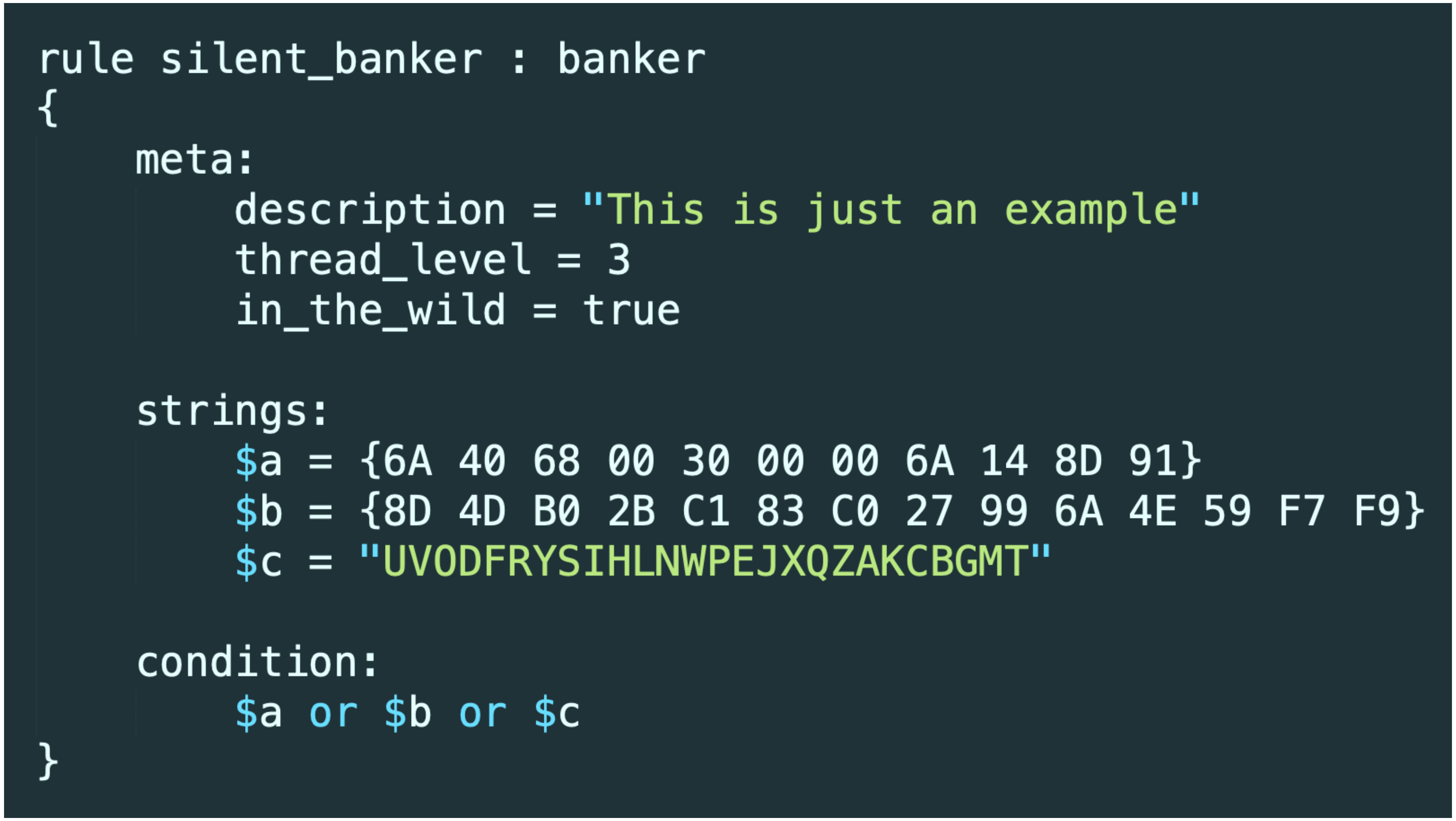
OS monitoring

- Sometimes, static analysis and OS monitoring are enough to uncover most information about the malware.
- If we want to go further: use debuggers.
- Set breakpoints, inspect memory, retrieve function arguments, return values, etc.

YARA rules

- Rules that can be used for malware detection.
- Based on patterns and signatures for known malware.
- Supported by some sandboxes (Cuckoo Sandbox, etc).
- Some IDS solutions also rely on YARA rules. Also supported by volatility (forensics).
- It’s easy to create rules for a new malware sample.

YARA rules



```
rule silent_banker : banker
{
    meta:
        description = "This is just an example"
        thread_level = 3
        in_the_wild = true

    strings:
        $a = {6A 40 68 00 30 00 00 6A 14 8D 91}
        $b = {8D 4D B0 2B C1 83 C0 27 99 6A 4E 59 F7 F9}
        $c = "UVODFRYSIHLNWPEJXQZAKCBGMT"

    condition:
        $a or $b or $c
}
```

Malware Forensics

- Memory dumps during/after malware execution are also very useful artefacts for analysis
- Recovering AES keys: **aeskeyfind**
- One of the most well-known tools is **volatility**:
  - Identifying Processes - **pslist**, **psxview**
  - Analyzing Network Connections - **netscan**, **connscan**
  - Extracting Injected Malicious Code - **malfind**
  - Revealing Rootkits and Hooking - **ssdt**, **idt**
  - Evidence of Code Injection via DLLs - **ldrmodules**, **dlllist**, **dlldump**
  - Reconstructing Command Activity - **cmdscan**, **consoles**