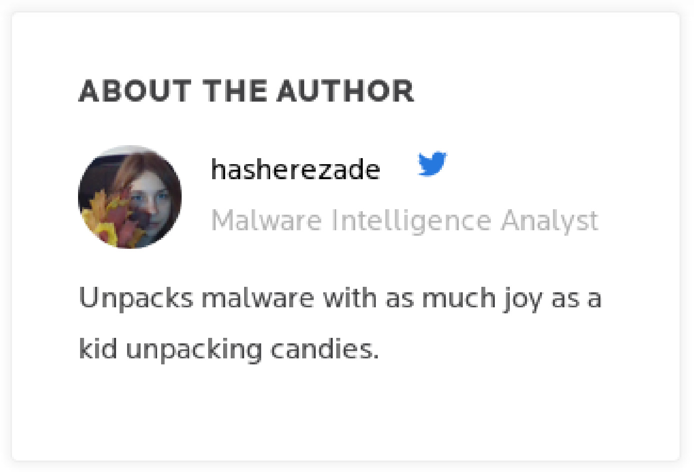
PE-sieve

AN OPENSOURCE SCANNER FOR HUNTING AND UNPACKING MALWARE

#whoami

HASHEREZADE.NET

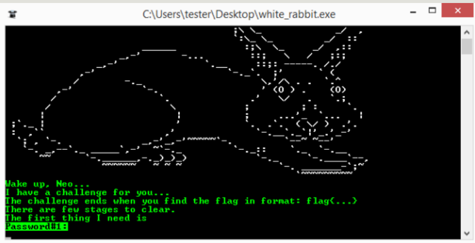
- Malware intelligence Analyst, technical blogger at Malwarebytes
- open source & free software developer (PE-bear, PE-sieve, and many others)
- writer/solver of crackmes
- wrote some ransomware decryptors
- makes videos related to malware analysis
- wrote a chapter to a book about RE


ABOUT THE AUTHOR

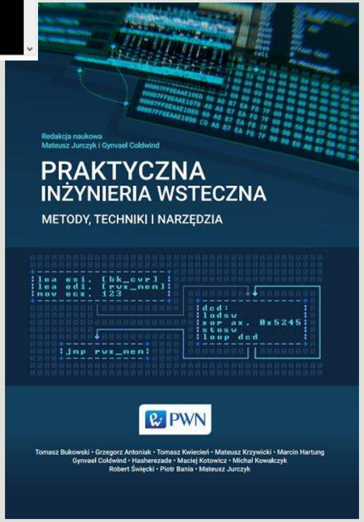
hasherezade

Malware Intelligence Analyst

Unpacks malware with as much joy as a kid unpacking candies.



```
Wake up, Neo...
I have a challenge for you...
The challenge ends when you find the flag in format: flag{...}
There are few stages to clear.
The first thing I need is
Password#1:
```


Redakcja naukowa
Mateusz Jurczyk i Gynvael Coldwind

PRAKTYCZNA INŻYNIERIA WSTECZNA

METODY, TECHNIKI I NARZĘDZIA

Tomasz Bukowski • Grzegorz Antoniak • Tomasz Kwiecień • Mateusz Krzywicki • Marcin Hartung
Gynvael Coldwind • Hasherezade • Maciej Kotowicz • Michał Kowalczyk
Robert Święcki • Piotr Bania • Mateusz Jurczyk

agenda

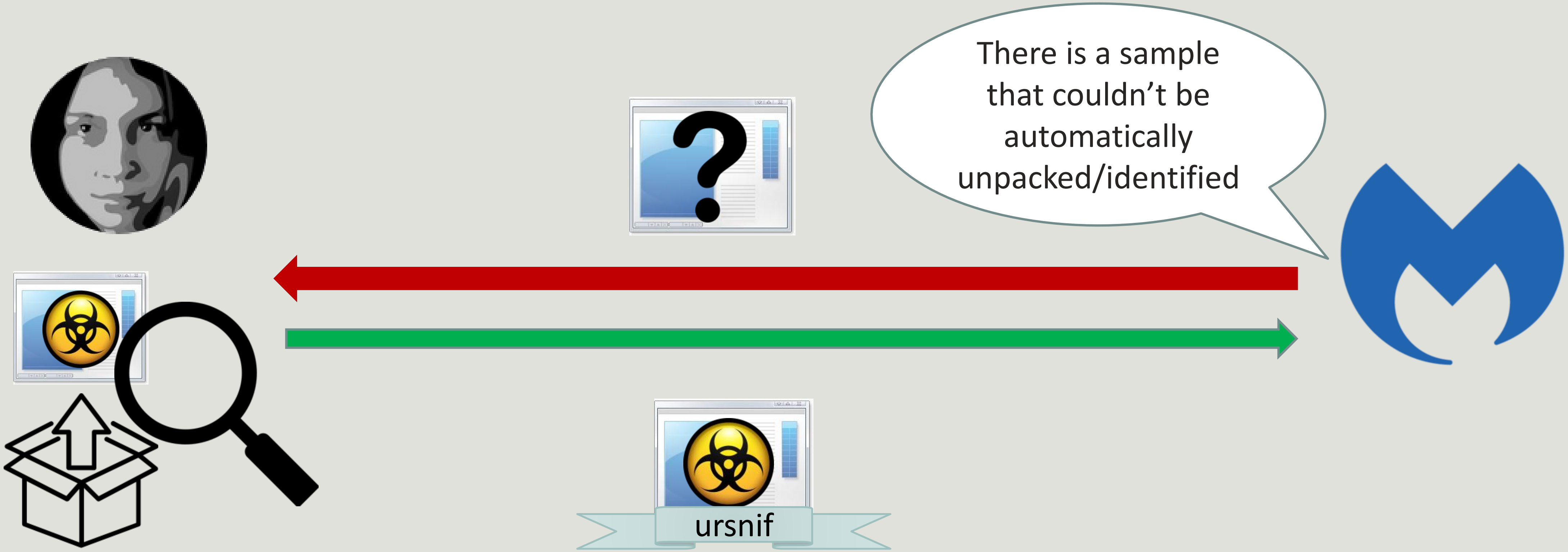
1. PE-sieve – brief history
2. Capabilities & usecases
3. Various approaches to finding code implants
4. PE-sieve implementation details

PE-sieve – brief history

HOW IT ALL STARTED AND WHERE WE ARE TODAY

Why I made PE-sieve?

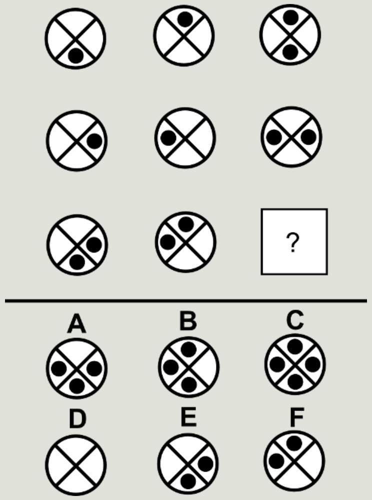
Part of my work is about unpacking unidentified samples...



There is a sample that couldn’t be automatically unpacked/identified

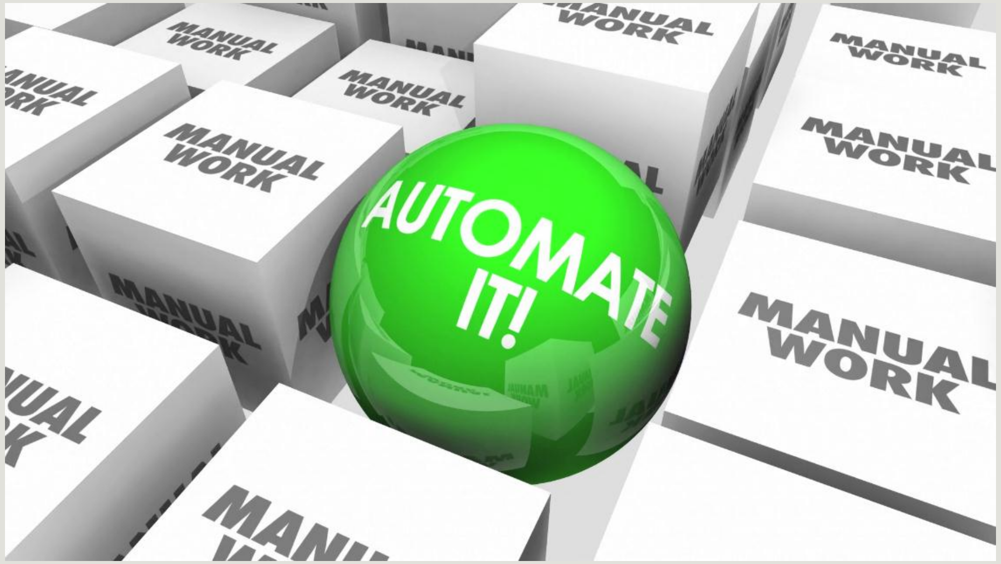
Why I made PE-sieve?

- When I started, I used to unpack samples manually
- Over the years, I learned a lot about how the malware unpacks itself in the memory, and saw the patterns



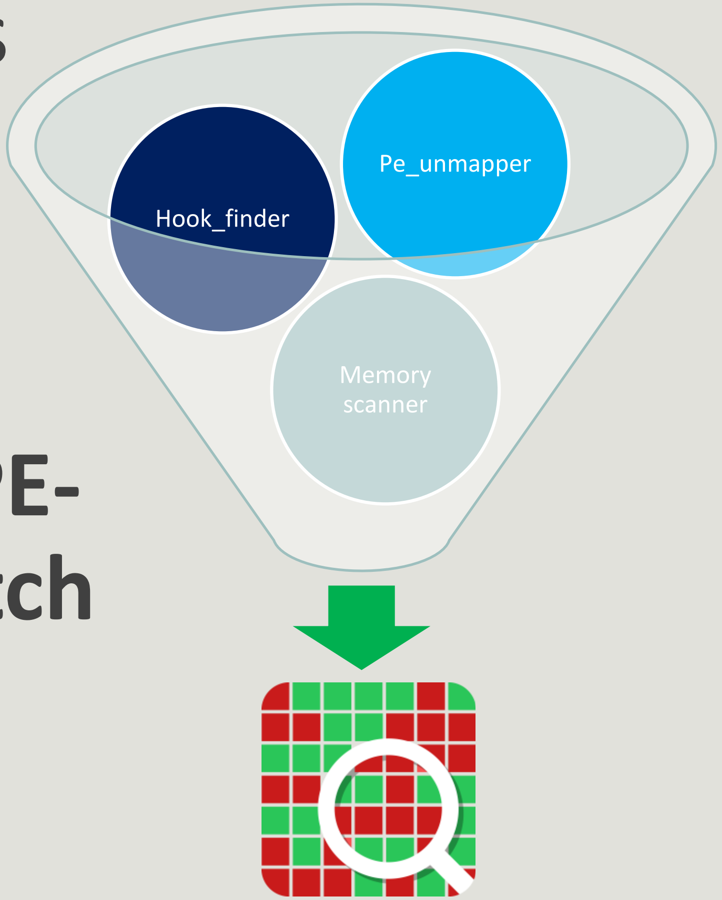
Why I made PE-sieve?

- I am originaly a programmer, so I put my experience into action by automating daily tasks



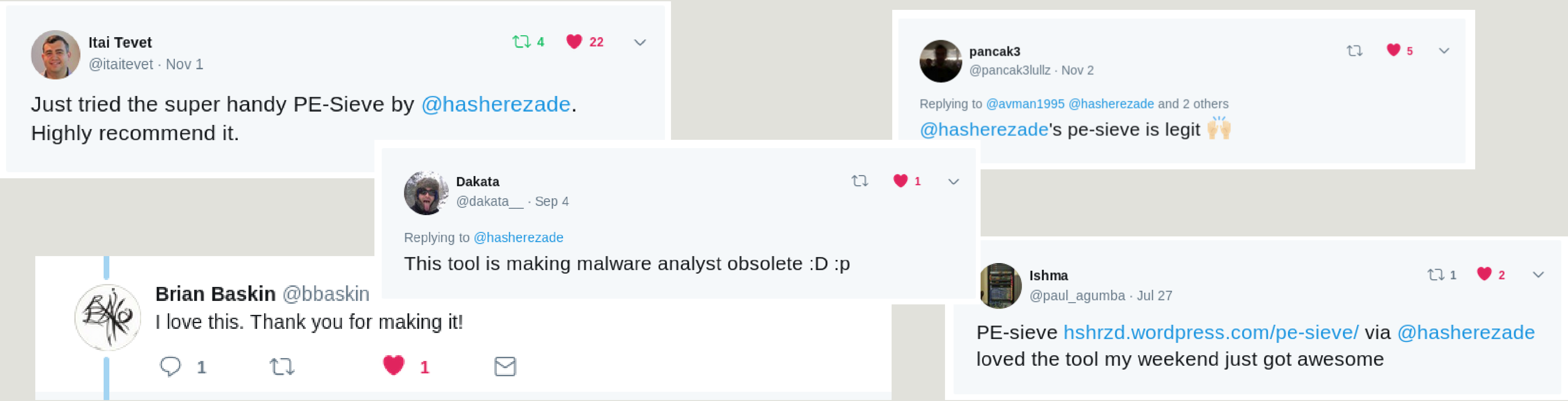
Why I made PE-sieve?

- I collected many small, simple tools for particular tasks (i.e. pe_unmapper, hook_finder)
- Around Christmas 2017 I combined them, creating the first version of **PE-sieve:** a **dynamic unpacker and patch finder**



Why I made PE-sieve?

- I use it every day, and keep improving it
- Other malware researchers also liked it...



Itai Tevet @itaitevet · Nov 1

4 22

Just tried the super handy PE-Sieve by @hasherezade. Highly recommend it.

pancak3 @pancak3lullz · Nov 2

5

Replying to @avman1995 @hasherezade and 2 others

@hasherezade's pe-sieve is legit 🙌

Dakata @dakata__ · Sep 4

1

Replying to @hasherezade

This tool is making malware analyst obsolete :D :p

Brian Baskin @bbaskin

I love this. Thank you for making it!

1 1

Ishma @paul_agumba · Jul 27

1 2

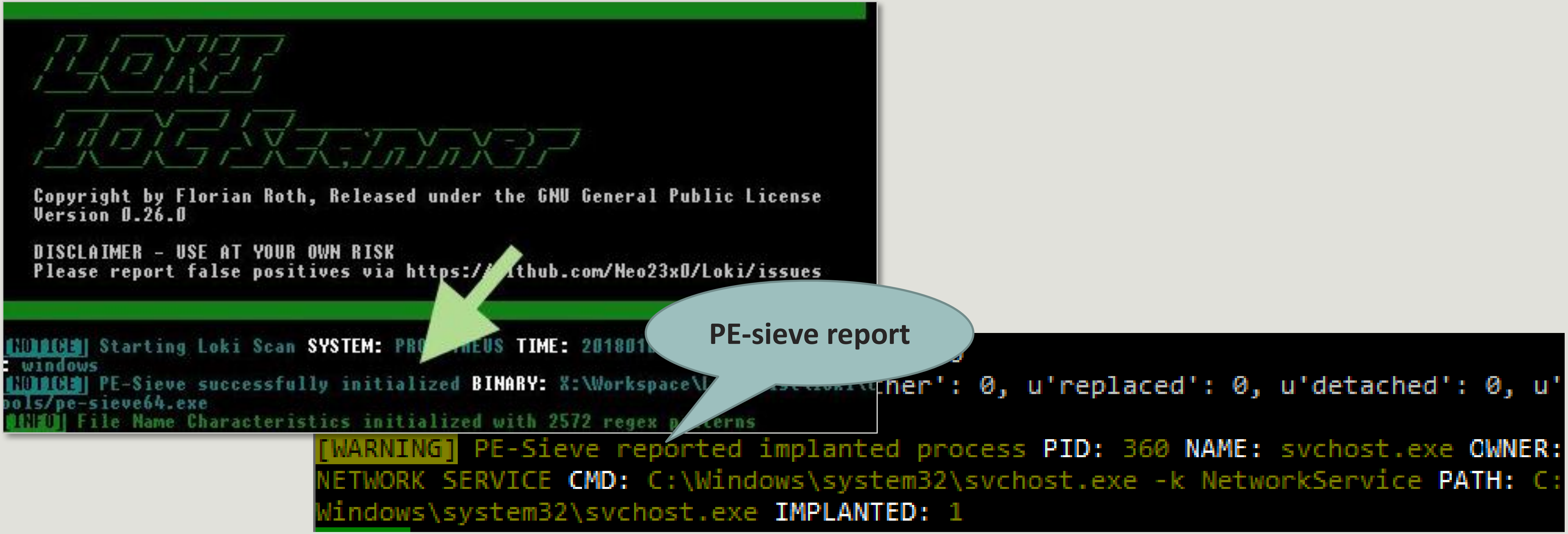
PE-sieve hshrzd.wordpress.com/pe-sieve/ via @hasherezade loved the tool my weekend just got awesome

PE-sieve in other projects

- PE-sieve is a light-weight component
- Can be used as a standalone application, or as DLL
- Became a base for my other projects:
  - Hollows Hunter (https://github.com/hasherezade/hollows_hunter)
  - MalUnpack (https://github.com/hasherezade/mal_unpack)

PE-sieve in other projects

- Adapted in **LOKI** scanner (https://github.com/Neo23x0/Loki)



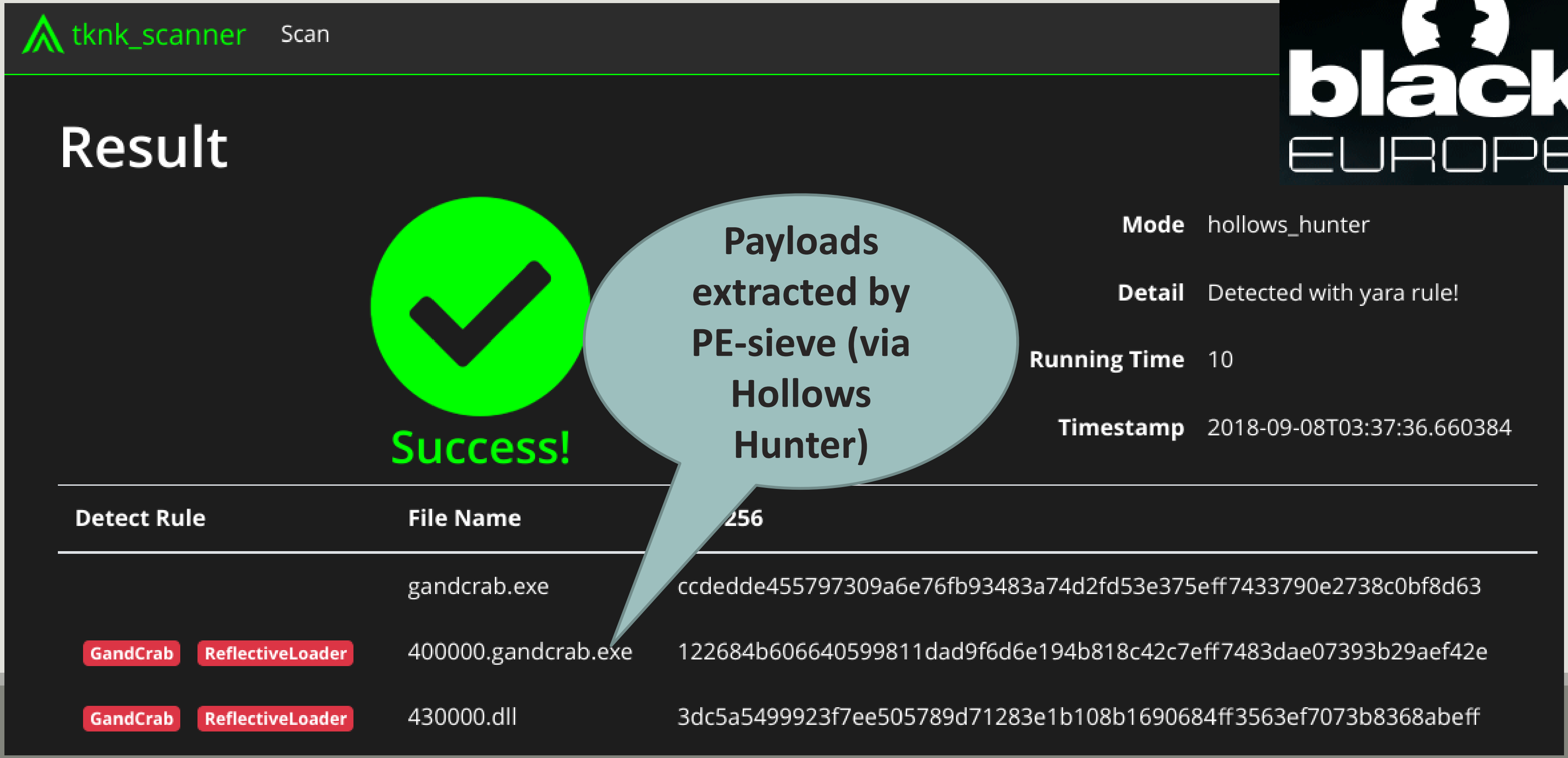
```
Copyright by Florian Roth, Released under the GNU General Public License
Version 0.26.0

DISCLAIMER - USE AT YOUR OWN RISK
Please report false positives via https:/ ithub.com/Neo23x0/Loki/issues

[NOTICE] Starting Loki Scan SYSTEM: PR   HEUS TIME: 201801
: windows
[NOTICE] PE-Sieve successfully initialized BINARY: X:\Workspace\
pols/pe-sieve64.exe
[INFO] File Name Characteristics initialized with 2572 regex p  terns
```

```
her': 0, u'replaced': 0, u'detached': 0, u'

[WARNING] PE-Sieve reported implanted process PID: 360 NAME: svchost.exe OWNER:
NETWORK SERVICE CMD: C:\Windows\system32\svchost.exe -k NetworkService PATH: C:
Windows\system32\svchost.exe IMPLANTED: 1
```

PE-sieve in other projects

- Adapted in **tknk_scanner** (https://github.com/nao-sec/tknk_scanner)

black hat EUROPE 2018



Result

Success!

Payloads extracted by PE-sieve (via Hollows Hunter)

Mode hollows_hunter

Detail Detected with yara rule!

Running Time 10

Timestamp 2018-09-08T03:37:36.660384

Detect Rule	File Name	256
	gandcrab.exe	ccdedde455797309a6e76fb93483a74d2fd53e375eff7433790e2738c0bf8d63
GandCrab ReflectiveLoader	400000.gandcrab.exe	122684b606640599811dad9f6d6e194b818c42c7eff7483dae07393b29aef42e
GandCrab ReflectiveLoader	430000.dll	3dc5a5499923f7ee505789d71283e1b108b1690684ff3563ef7073b8368abeff

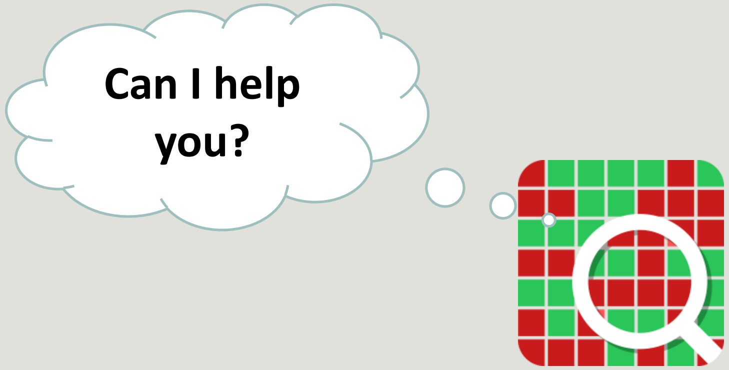
PE-sieve stole my job…

- We save a lot of time from manual sample unpacking:
  - Almost all the dumped samples allow for a **malware family identification**
  - **Majority** of the dumped payloads are suitable for **dynamic analysis** of the next stage
    - (minority doesn’t run properly and still needs manual unpacking)

Beyond unpacking...

- **finding** what the **implanted code** is
- **reconstructing** the corrupt parts of the payload
- converting PE into a **raw format**
- pointing out where the **hooks/patches** are installed

ursnif



Capabilities & usecases

WHAT PROBLEMS CAN IT SOLVE?

PE-sieve: capabilities

- Works on a **live system**
- Focus: speed and simplicity of use
- **Passive scan**, not hooking any APIs
- Can be used post-infection
- **Generates material** ready to be analyzed: not only detection, but precise details
- Free & open source: https://github.com/hasherezade/pe-sieve
  https://github.com/hasherezade/hollows_hunter

PE-sieve: capabilities

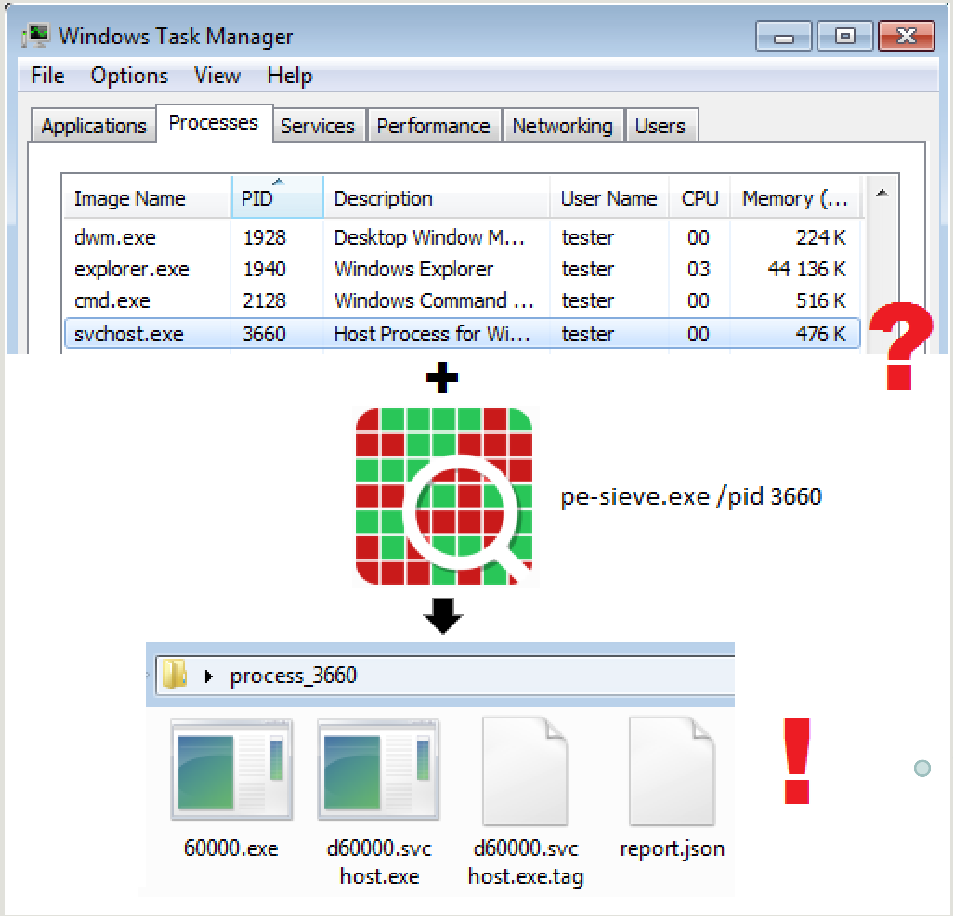

Windows Task Manager
File Options View Help
Applications Processes Services Performance Networking Users
?
+
!

Image Name	PID	Description	User Name	CPU	Memory (...
dwm.exe	1928	Desktop Window M...	tester	00	224 K
explorer.exe	1940	Windows Explorer	tester	03	44 136 K
cmd.exe	2128	Windows Command ...	tester	00	516 K
svchost.exe	3660	Host Process for Wi...	tester	00	476 K

```
pe-sieve.exe /pid 3660
```

process_3660

- 60000.exe
- d60000.svchost.exe
- d60000.svchost.exe.tag
- report.json

Suspected...

Detected!

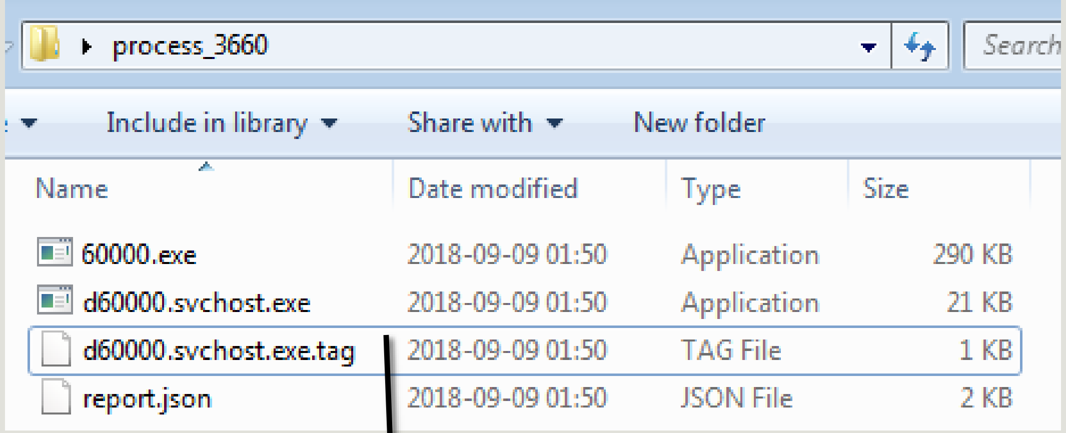
What does PE-sieve detect?

- Inline hooks
- Packed and self-modifying PE files
- Replaced processes: i.e. Process Hollowing, Process Doppelgänging
- Manually loaded PE-files (Reflective DLL Injection and others)
- Shellcodes

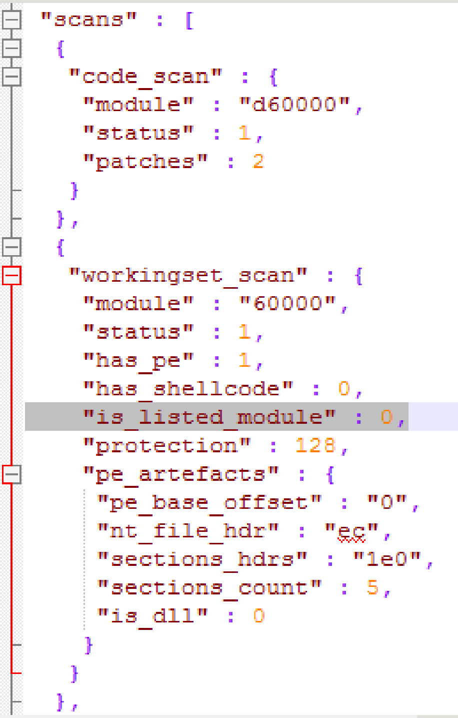
What PE-sieve is NOT?

- Not an automated anti-malware scanner
  - It **collects raw material** and some indicators
  - but does **not do automated classification**
  - it is conceptually similar to GMER
- Not a tool for analyzing memory dumps and process post-mortem analysis (try Volatility+plugins instead)

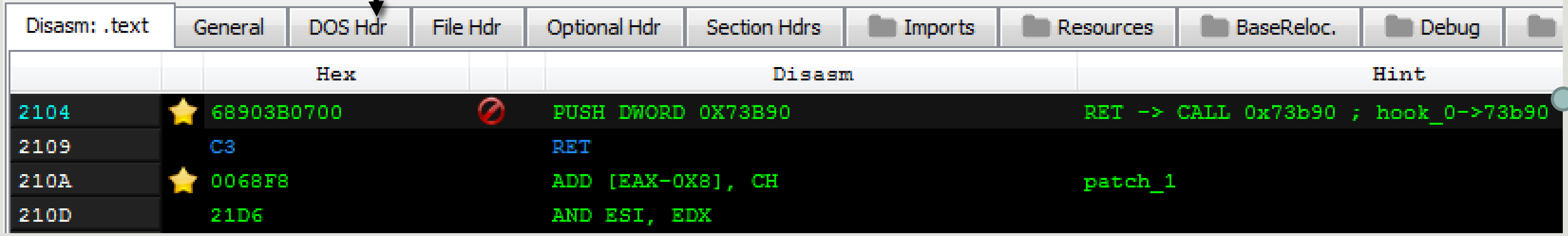
Dumping modified and implanted modules



Name	Date modified	Type	Size
60000.exe	2018-09-09 01:50	Application	290 KB
d60000.svchost.exe	2018-09-09 01:50	Application	21 KB
d60000.svchost.exe.tag	2018-09-09 01:50	TAG File	1 KB
report.json	2018-09-09 01:50	JSON File	2 KB



```
"scans" : [
 {
  "code_scan" : {
   "module" : "d60000",
   "status" : 1,
   "patches" : 2
  }
 },
 {
  "workingset_scan" : {
   "module" : "60000",
   "status" : 1,
   "has_pe" : 1,
   "has_shellcode" : 0,
   "is_listed_module" : 0,
   "protection" : 128,
   "pe_artefacts" : {
    "pe_base_offset" : "0",
    "nt_file_hdr" : "ec",
    "sections_hdrs" : "1e0",
    "sections_count" : 5,
    "is_dll" : 0
   }
  }
 },
```

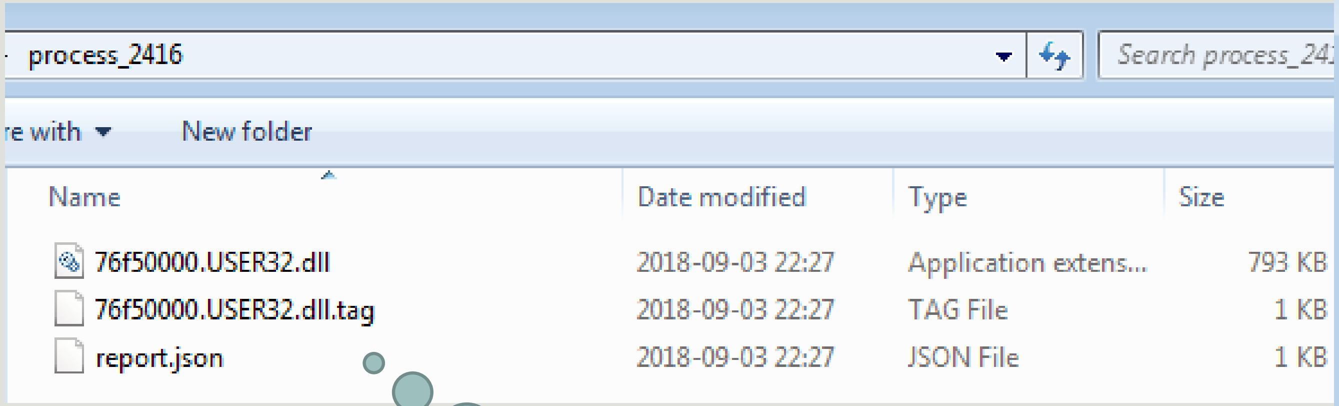


```
        Hex                 Disasm                     Hint
2104    68903B0700          PUSH DWORD 0X73B90         RET -> CALL 0x73b90 ; hook_0->73b90
2109    C3                  RET
210A    0068F8              ADD [EAX-0X8], CH          patch_1
210D    21D6                AND ESI, EDX
```

Entry Point of svchost is patched to redirect to the implant

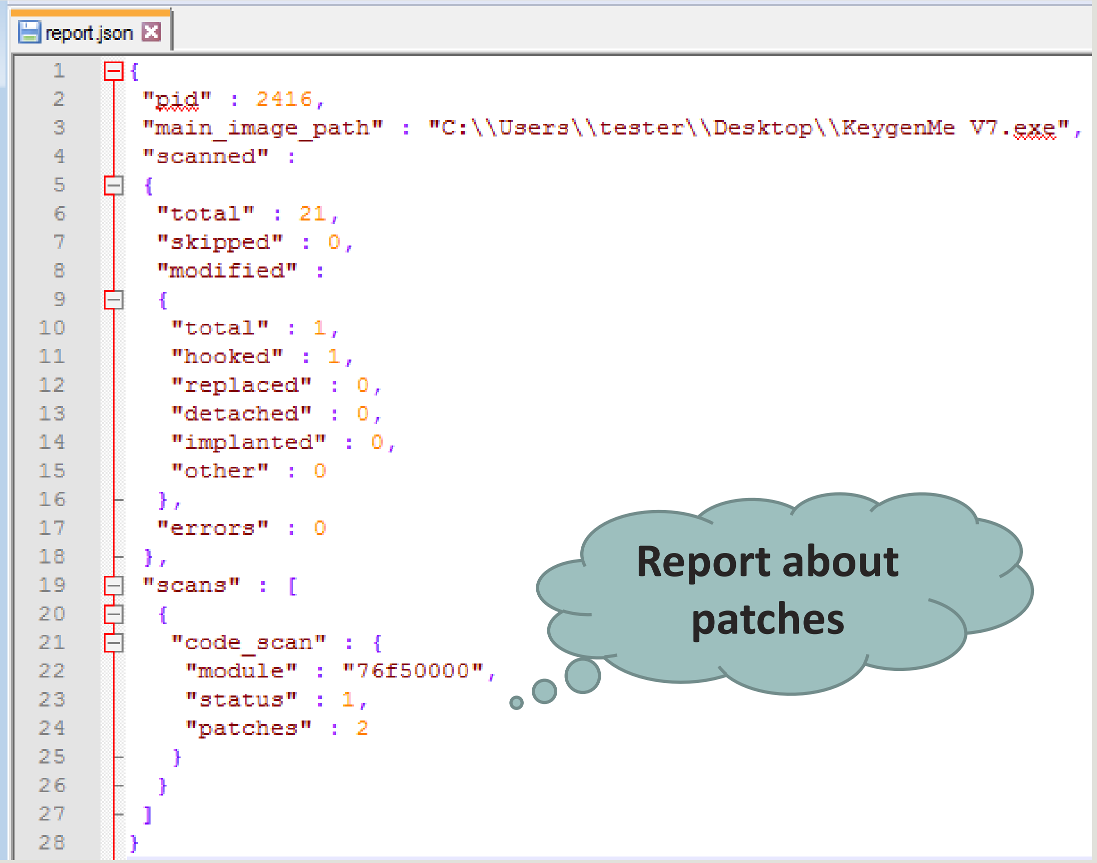
Inline hooking detection

Test case: a crackme with inline hooks



Name	Date modified	Type	Size
76f50000.USER32.dll	2018-09-03 22:27	Application extens...	793 KB
76f50000.USER32.dll.tag	2018-09-03 22:27	TAG File	1 KB
report.json	2018-09-03 22:27	JSON File	1 KB

The hooked/patched module is automatically dumped

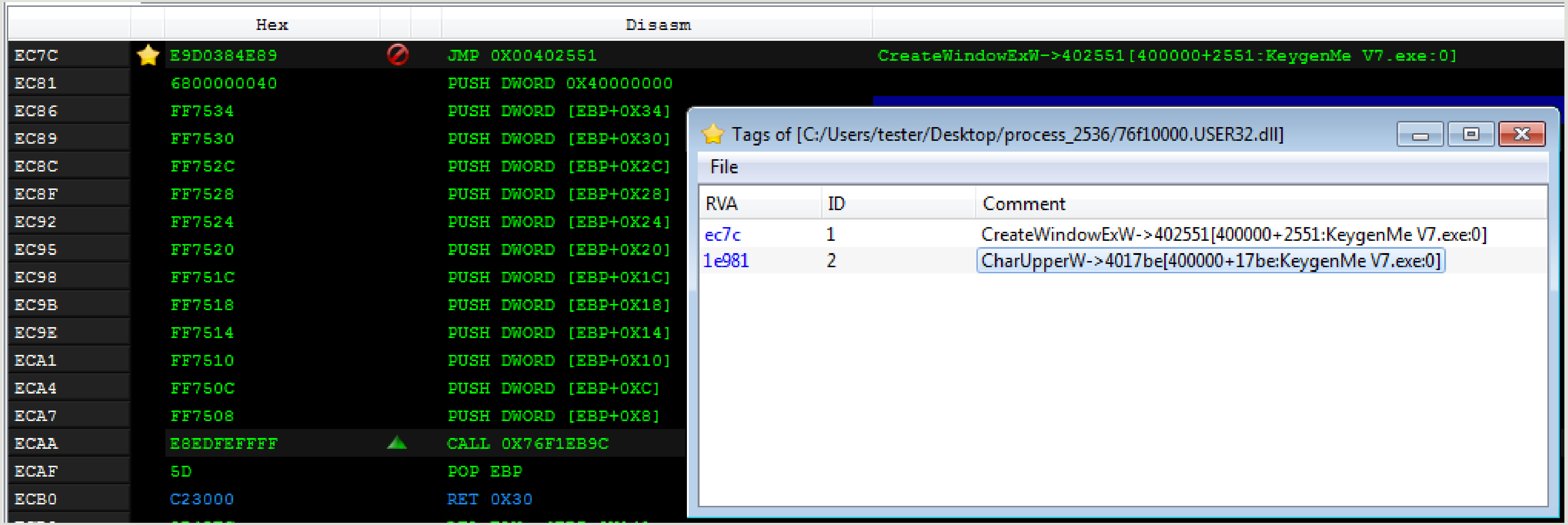


```
{
 "pid" : 2416,
 "main_image_path" : "C:\\Users\\tester\\Desktop\\KeygenMe V7.exe",
 "scanned" :
 {
  "total" : 21,
  "skipped" : 0,
  "modified" :
  {
   "total" : 1,
   "hooked" : 1,
   "replaced" : 0,
   "detached" : 0,
   "implanted" : 0,
   "other" : 0
  },
  "errors" : 0
 },
 "scans" : [
  {
   "code_scan" : {
    "module" : "76f50000",
    "status" : 1,
    "patches" : 2
   }
  }
 ]
}
```

Report about patches

Inline hooking detection

- The TAG file, along with the dumped module, can be loaded to PE-bear or IDA and further analyzed



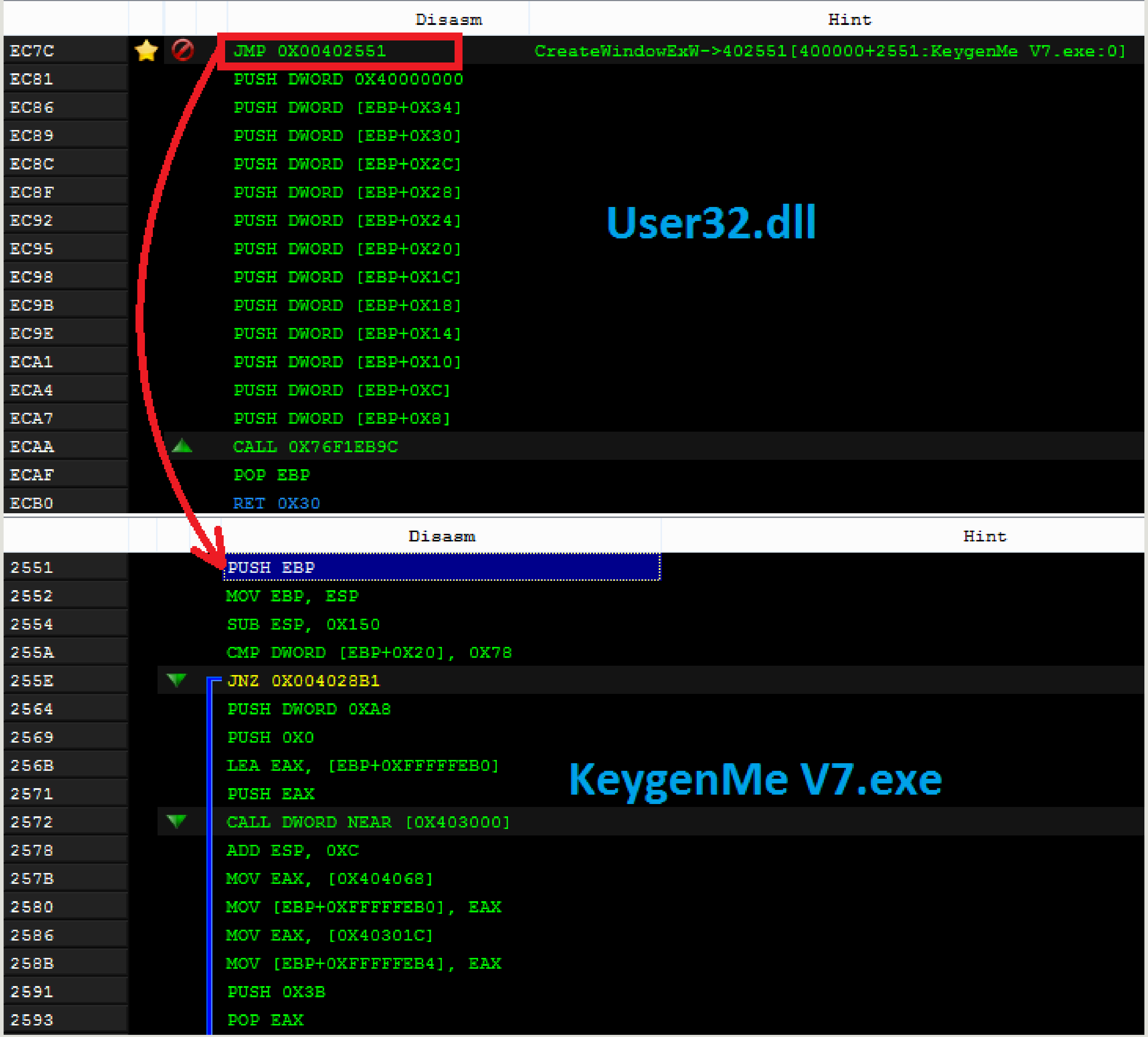
```
                Hex                     Disasm
EC7C    E9D0384E89      JMP 0X00402551              CreateWindowExW->402551[400000+2551:KeygenMe V7.exe:0]
EC81    6800000040      PUSH DWORD 0X40000000
EC86    FF7534          PUSH DWORD [EBP+0X34]
EC89    FF7530          PUSH DWORD [EBP+0X30]
EC8C    FF752C          PUSH DWORD [EBP+0X2C]
EC8F    FF7528          PUSH DWORD [EBP+0X28]
EC92    FF7524          PUSH DWORD [EBP+0X24]
EC95    FF7520          PUSH DWORD [EBP+0X20]
EC98    FF751C          PUSH DWORD [EBP+0X1C]
EC9B    FF7518          PUSH DWORD [EBP+0X18]
EC9E    FF7514          PUSH DWORD [EBP+0X14]
ECA1    FF7510          PUSH DWORD [EBP+0X10]
ECA4    FF750C          PUSH DWORD [EBP+0XC]
ECA7    FF7508          PUSH DWORD [EBP+0X8]
ECAA    E8EDFEFFFF      CALL 0X76F1EB9C
ECAF    5D              POP EBP
ECB0    C23000          RET 0X30
```

Tags of [C:/Users/tester/Desktop/process_2536/76f10000.USER32.dll]

RVA	ID	Comment
ec7c	1	CreateWindowExW->402551[400000+2551:KeygenMe V7.exe:0]
1e981	2	CharUpperW->4017be[400000+17be:KeygenMe V7.exe:0]

Inline hooking detection

Generated tags allow viewing the patches in their original context, and analyzing with typical tools



User32.dll

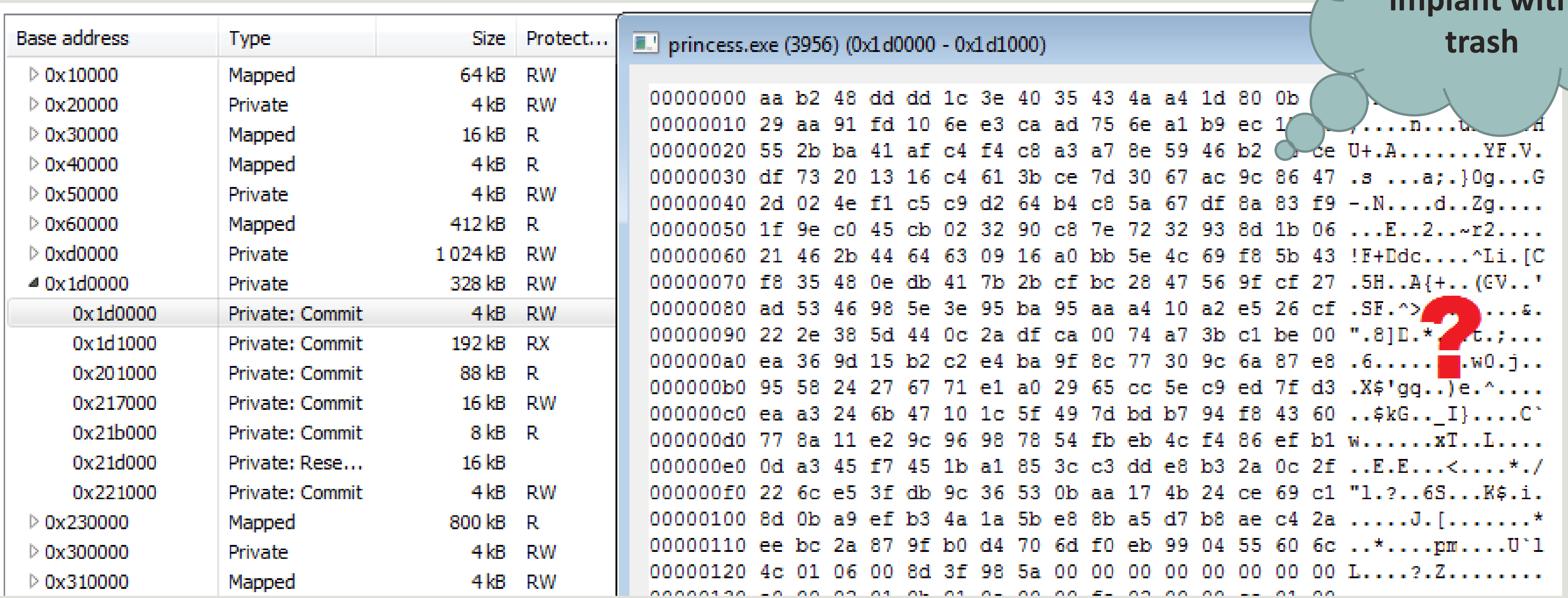
```
          Disasm                              Hint
EC7C      JMP 0X00402551                      CreateWindowExW->402551[400000+2551:KeygenMe V7.exe:0]
EC81      PUSH DWORD 0X40000000
EC86      PUSH DWORD [EBP+0X34]
EC89      PUSH DWORD [EBP+0X30]
EC8C      PUSH DWORD [EBP+0X2C]
EC8F      PUSH DWORD [EBP+0X28]
EC92      PUSH DWORD [EBP+0X24]
EC95      PUSH DWORD [EBP+0X20]
EC98      PUSH DWORD [EBP+0X1C]
EC9B      PUSH DWORD [EBP+0X18]
EC9E      PUSH DWORD [EBP+0X14]
ECA1      PUSH DWORD [EBP+0X10]
ECA4      PUSH DWORD [EBP+0XC]
ECA7      PUSH DWORD [EBP+0X8]
ECAA      CALL 0X76F1EB9C
ECAF      POP EBP
ECB0      RET 0X30
```

KeygenMe V7.exe

```
          Disasm                              Hint
2551      PUSH EBP
2552      MOV EBP, ESP
2554      SUB ESP, 0X150
255A      CMP DWORD [EBP+0X20], 0X78
255E      JNZ 0X004028B1
2564      PUSH DWORD 0XA8
2569      PUSH 0X0
256B      LEA EAX, [EBP+0XFFFFFEB0]
2571      PUSH EAX
2572      CALL DWORD NEAR [0X403000]
2578      ADD ESP, 0XC
257B      MOV EAX, [0X404068]
2580      MOV [EBP+0XFFFFFEB0], EAX
2586      MOV EAX, [0X40301C]
258B      MOV [EBP+0XFFFFFEB4], EAX
2591      PUSH 0X3B
2593      POP EAX
```

Detecting partially erased headers

Princess Locker overwrites headers of the implant with trash

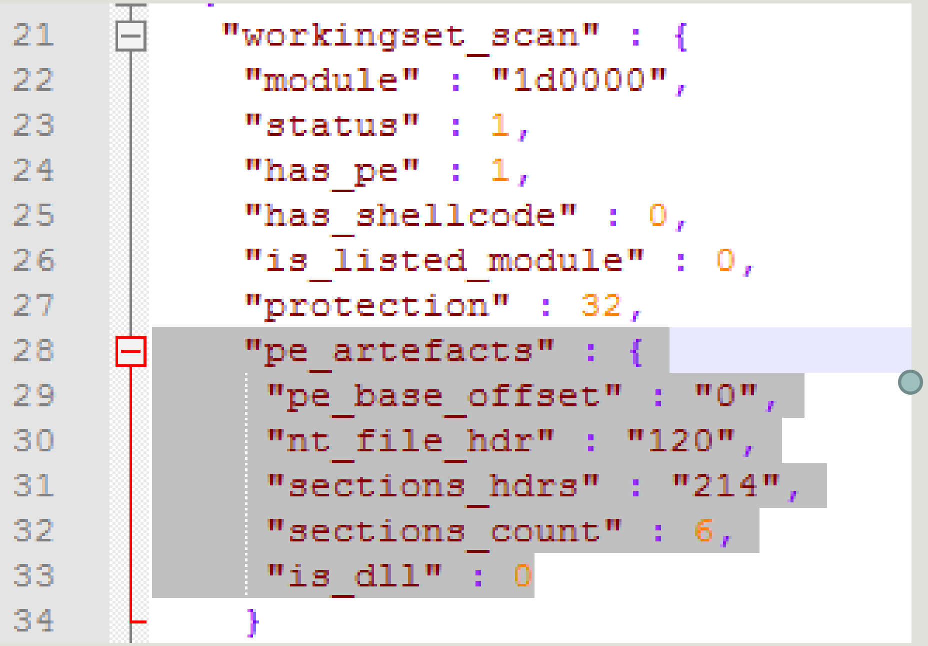


Base address	Type	Size	Protect...
▷ 0x10000	Mapped	64 kB	RW
▷ 0x20000	Private	4 kB	RW
▷ 0x30000	Mapped	16 kB	R
▷ 0x40000	Mapped	4 kB	R
▷ 0x50000	Private	4 kB	RW
▷ 0x60000	Mapped	412 kB	R
▷ 0xd0000	Private	1 024 kB	RW
◢ 0x1d0000	Private	328 kB	RW
0x1d0000	Private: Commit	4 kB	RW
0x1d1000	Private: Commit	192 kB	RX
0x201000	Private: Commit	88 kB	R
0x217000	Private: Commit	16 kB	RW
0x21b000	Private: Commit	8 kB	R
0x21d000	Private: Rese...	16 kB	
0x221000	Private: Commit	4 kB	RW
▷ 0x230000	Mapped	800 kB	R
▷ 0x300000	Private	4 kB	RW
▷ 0x310000	Mapped	4 kB	RW

princess.exe (3956) (0x1d0000 - 0x1d1000)

```
00000000 aa b2 48 dd dd 1c 3e 40 35 43 4a a4 1d 80 0b
00000010 29 aa 91 fd 10 6e e3 ca ad 75 6e a1 b9 ec 1       ;....n...u      H
00000020 55 2b ba 41 af c4 f4 c8 a3 a7 8e 59 46 b2     ce U+.A.......YF.V.
00000030 df 73 20 13 16 c4 61 3b ce 7d 30 67 ac 9c 86 47 .s ...a;.}0g...G
00000040 2d 02 4e f1 c5 c9 d2 64 b4 c8 5a 67 df 8a 83 f9 -.N....d..Zg....
00000050 1f 9e c0 45 cb 02 32 90 c8 7e 72 32 93 8d 1b 06 ...E..2..~r2....
00000060 21 46 2b 44 64 63 09 16 a0 bb 5e 4c 69 f8 5b 43 !F+Ddc....^Li.[C
00000070 f8 35 48 0e db 41 7b 2b cf bc 28 47 56 9f cf 27 .5H..A{+..(GV..'
00000080 ad 53 46 98 5e 3e 95 ba 95 aa a4 10 a2 e5 26 cf .SF.^>     ...&.
00000090 22 2e 38 5d 44 0c 2a df ca 00 74 a7 3b c1 be 00 ".8]D.*   t.;...
000000a0 ea 36 9d 15 b2 c2 e4 ba 9f 8c 77 30 9c 6a 87 e8 .6.....  .w0.j..
000000b0 95 58 24 27 67 71 e1 a0 29 65 cc 5e c9 ed 7f d3 .X$'gq..)e.^....
000000c0 ea a3 24 6b 47 10 1c 5f 49 7d bd b7 94 f8 43 60 ..$kG.._I}....C`
000000d0 77 8a 11 e2 9c 96 98 78 54 fb eb 4c f4 86 ef b1 w......xT..L....
000000e0 0d a3 45 f7 45 1b a1 85 3c c3 dd e8 b3 2a 0c 2f ..E.E...<....*./
000000f0 22 6c e5 3f db 9c 36 53 0b aa 17 4b 24 ce 69 c1 "l.?..6S...K$.i.
00000100 8d 0b a9 ef b3 4a 1a 5b e8 8b a5 d7 b8 ae c4 2a .....J.[.......*
00000110 ee bc 2a 87 9f b0 d4 70 6d f0 eb 99 04 55 60 6c ..*....pm....U`l
00000120 4c 01 06 00 8d 3f 98 5a 00 00 00 00 00 00 00 00 L....?.Z........
```

Detecting partially erased headers

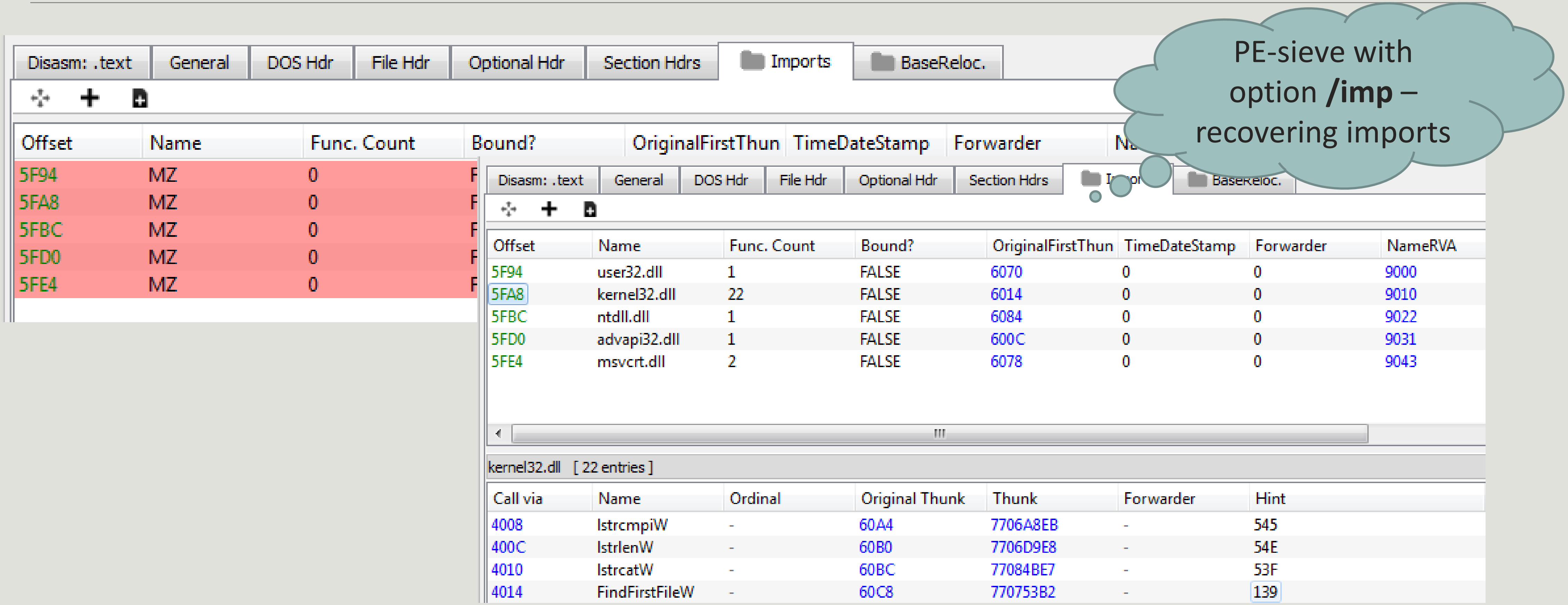


```
  "workingset_scan" : {
   "module" : "1d0000",
   "status" : 1,
   "has_pe" : 1,
   "has_shellcode" : 0,
   "is_listed_module" : 0,
   "protection" : 32,
   "pe_artefacts" : {
    "pe_base_offset" : "0",
    "nt_file_hdr" : "120",
    "sections_hdrs" : "214",
    "sections_count" : 6,
    "is_dll" : 0
   }
```

PE-sieve is still able to detect the remainings of the header and reconstruct the full PE

https://www.youtube.com/watch?v=dFJcGYUFB0s

Reconstructing erased imports



Offset	Name	Func. Count	Bound?	OriginalFirstThun	TimeDateStamp	Forwarder	Na
5F94	MZ	0	F				
5FA8	MZ	0	F				
5FBC	MZ	0	F				
5FD0	MZ	0	F				
5FE4	MZ	0	F				

Offset	Name	Func. Count	Bound?	OriginalFirstThun	TimeDateStamp	Forwarder	NameRVA
5F94	user32.dll	1	FALSE	6070	0	0	9000
5FA8	kernel32.dll	22	FALSE	6014	0	0	9010
5FBC	ntdll.dll	1	FALSE	6084	0	0	9022
5FD0	advapi32.dll	1	FALSE	600C	0	0	9031
5FE4	msvcrt.dll	2	FALSE	6078	0	0	9043

kernel32.dll [22 entries]

Call via	Name	Ordinal	Original Thunk	Thunk	Forwarder	Hint
4008	lstrcmpiW	-	60A4	7706A8EB	-	545
400C	lstrlenW	-	60B0	7706D9E8	-	54E
4010	lstrcatW	-	60BC	77084BE7	-	53F
4014	FindFirstFileW	-	60C8	770753B2	-	139

PE-sieve with option **/imp** – recovering imports

https://www.youtube.com/watch?v=YJjm5yT1rdM

Use-Cases

- Unpacking malware (selected sample), examining a single process: PE-sieve
- Scanning a full system to detect hidden implants: HollowsHunter
- Unpacking a big set of samples: MalUnpack (https://youtu.be/hoyHz9qSCY8)

Demo #1

PE-sieve vs Process Doppelgänging

https://youtu.be/4Brqslk3ni4

Demo #2

PE-sieve vs Finfisher variant

https://youtu.be/cQ-51Wn_Kco

Various approaches of finding code implants

SIMILARITIES AND DIFFERENCES WITH OTHER TOOLS

Code implants

- Malicious and non-malicious purposes:
  - Micro-patching applications without recompiling code
  - Packed executables
  - Self-modifying code
  - Hooking: userland rootkits, data interception, sandboxes

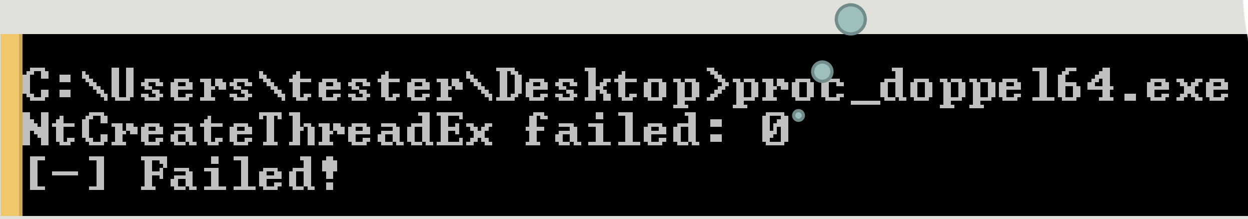
Infecting a running process

- Malware impersonates processes to run under their cover
- Examples of the techniques:
  - Process Hollowing (RunPE)
  - Manual PE loading (various variants, including Reflective DLL injection)
  - Process Doppelgänging
  - Combinations of multiple techniques (i.e. Transacted Hollowing)

Approach #1: monitoring and blocking API calls

- Many AV products monitor called APIs to prevent installing malicious implants

Blocked by AV



```
C:\Users\tester\Desktop>proc_doppel64.exe
NtCreateThreadEx failed: 0
[-] Failed!
```

Approach #1: monitoring and blocking API calls

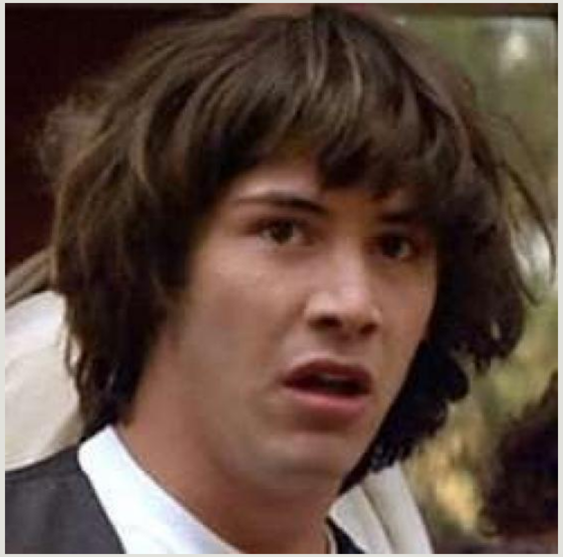
- Malware authors/offensive researchers try to evade it by finding uncommon APIs that can be used to make injection.

Some newer examples:

- AtomBombing technique
- Process Doppelgänging

Approach #1: monitoring and blocking API calls

- What if some unknown API was used for injection?
- What if we want to scan a system post-factum?
- How to detect and implant without knowing how it was injected?



Approach #2: search implants post-infection

- Some applications use another approach:
  - search implants in the memory post-infection
- Examples:
  - MalFind (a Volatility plugin)
  - RunPE detector
  - **PE-sieve**

PE-sieve – implementation details

OVERVIEW OF THE THE CODE & USED APPROACHES

Just follow the artefacts…

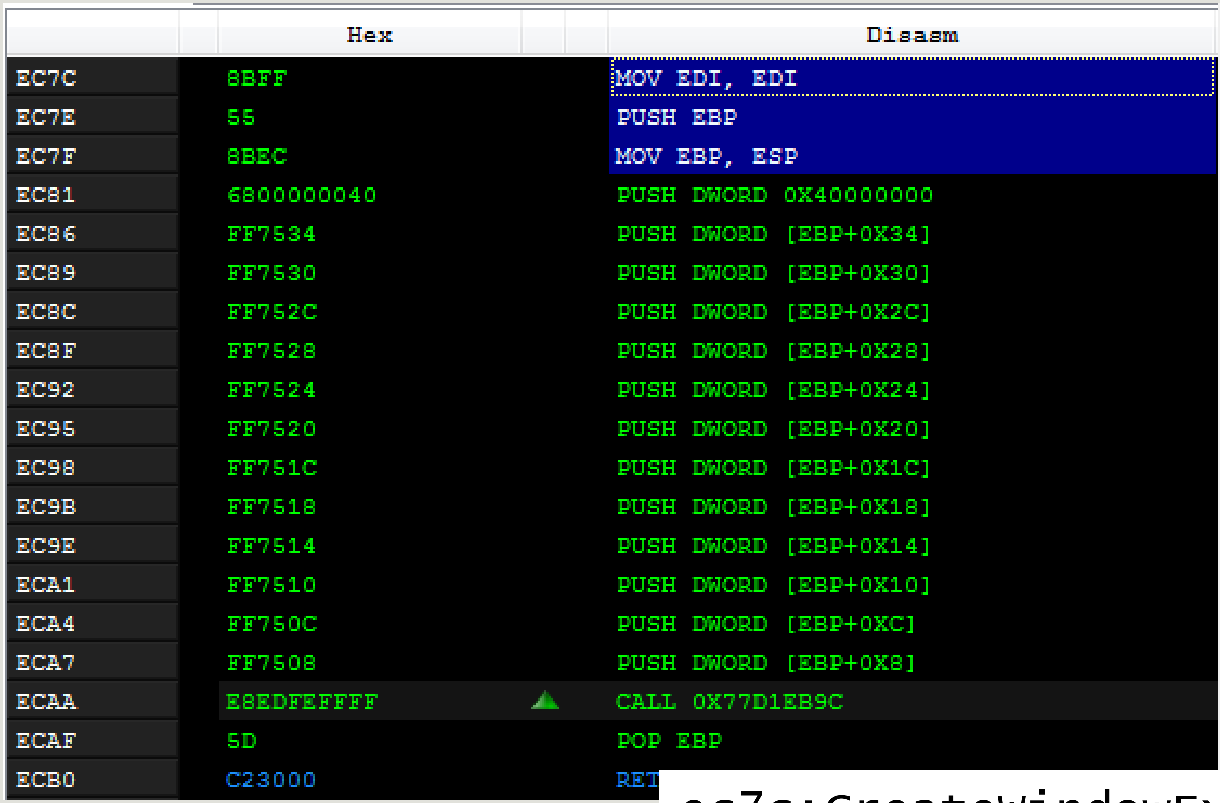
- No impersonation technique is perfect: they all leave some suspicious artefacts
- See what was modified, see how the code area was mapped...

Detection: inline hooking, self-modifying code

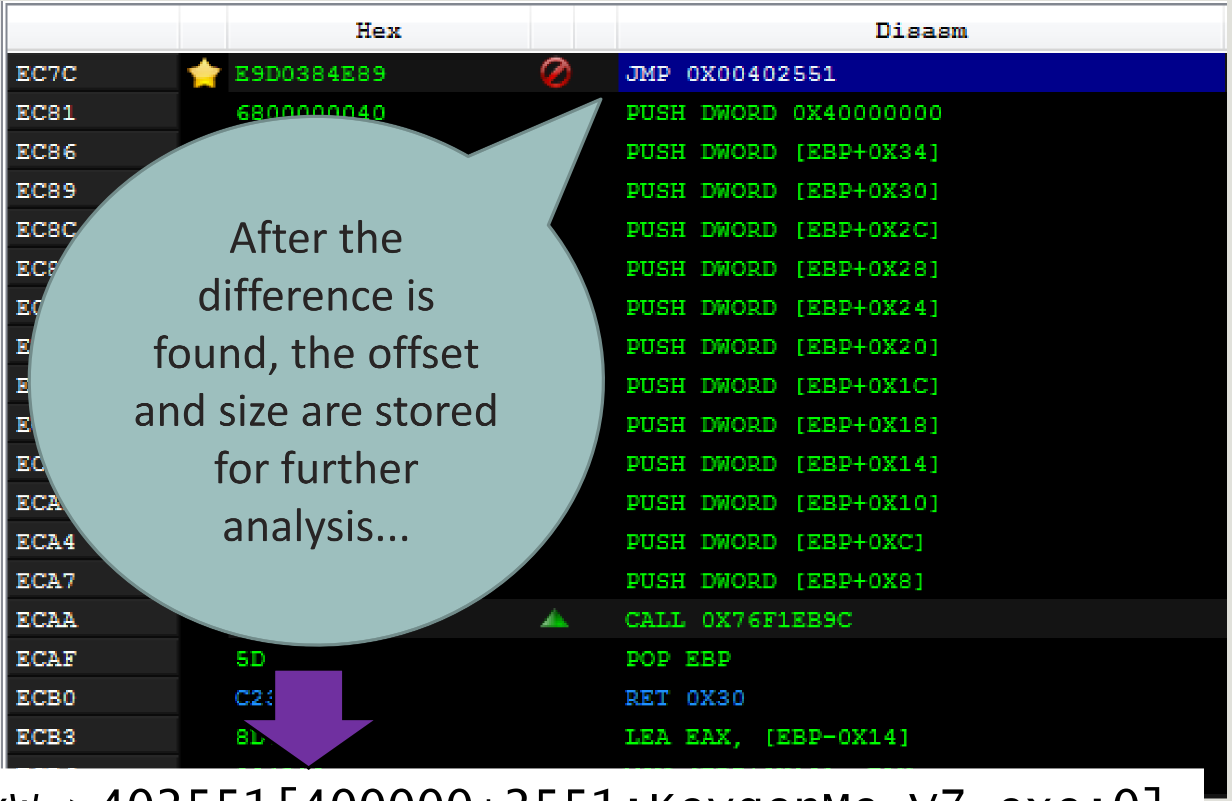
- **Code scan**
  - Load the PE from the disk that corresponds to the module within the process
  - Detect all the sections containing code
  - Transform both sections into the same format (relocate to the same base, remove IAT, etc.)
  - Compare

Code scan

- **Normalize and compare…**



```
        Hex                 Disasm
EC7C    8BFF                MOV EDI, EDI
EC7E    55                  PUSH EBP
EC7F    8BEC                MOV EBP, ESP
EC81    6800000040          PUSH DWORD 0X40000000
EC86    FF7534              PUSH DWORD [EBP+0X34]
EC89    FF7530              PUSH DWORD [EBP+0X30]
EC8C    FF752C              PUSH DWORD [EBP+0X2C]
EC8F    FF7528              PUSH DWORD [EBP+0X28]
EC92    FF7524              PUSH DWORD [EBP+0X24]
EC95    FF7520              PUSH DWORD [EBP+0X20]
EC98    FF751C              PUSH DWORD [EBP+0X1C]
EC9B    FF7518              PUSH DWORD [EBP+0X18]
EC9E    FF7514              PUSH DWORD [EBP+0X14]
ECA1    FF7510              PUSH DWORD [EBP+0X10]
ECA4    FF750C              PUSH DWORD [EBP+0XC]
ECA7    FF7508              PUSH DWORD [EBP+0X8]
ECAA    E8EDFEFFFF          CALL 0X77D1EB9C
ECAF    5D                  POP EBP
ECB0    C23000              RET
```



```
        Hex                 Disasm
EC7C    E9D0384E89          JMP 0X00402551
EC81    6800000040          PUSH DWORD 0X40000000
EC86                        PUSH DWORD [EBP+0X34]
EC89                        PUSH DWORD [EBP+0X30]
EC8C                        PUSH DWORD [EBP+0X2C]
EC8                         PUSH DWORD [EBP+0X28]
EC                          PUSH DWORD [EBP+0X24]
E                           PUSH DWORD [EBP+0X20]
E                           PUSH DWORD [EBP+0X1C]
E                           PUSH DWORD [EBP+0X18]
EC                          PUSH DWORD [EBP+0X14]
ECA                         PUSH DWORD [EBP+0X10]
ECA4                        PUSH DWORD [EBP+0XC]
ECA7                        PUSH DWORD [EBP+0X8]
ECAA                        CALL 0X76F1EB9C
ECAF    5D                  POP EBP
ECB0    C2                  RET 0X30
ECB3    8D                  LEA EAX, [EBP-0X14]
```

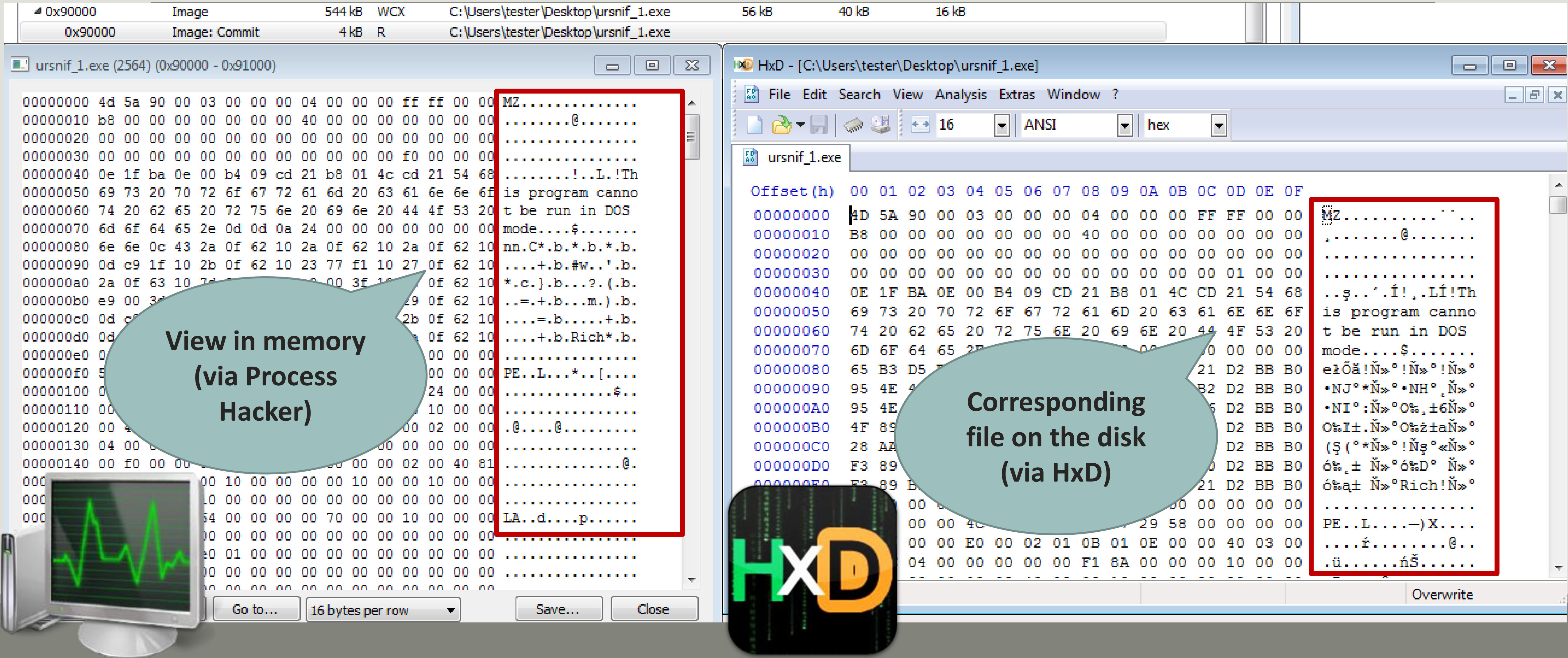
After the difference is found, the offset and size are stored for further analysis…

```
ec7c;CreateWindowExW->402551[400000+2551:KeygenMe V7.exe:0]
```

Detection: impersonated process

- **Headers scan**
  - Load the PE from the disk that corresponds to the module within the process
  - Are their headers matching?
- When it works?
  - For all the techniques that rely on connecting the implanted PE to the PEB
  - Covers Process Hollowing, Process Doppelgänging...

Headers scan



```
0x90000     Image           544 kB  WCX   C:\Users\tester\Desktop\ursnif_1.exe   56 kB   40 kB   16 kB
  0x90000   Image: Commit     4 kB  R     C:\Users\tester\Desktop\ursnif_1.exe
```

16 bytes per row

```
00000000 4d 5a 90 00 03 00 00 00 04 00 00 00 ff ff 00 00 MZ..............
00000010 b8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00 ........@.......
00000020 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00000030 00 00 00 00 00 00 00 00 00 00 00 00 f0 00 00 00 ................
00000040 0e 1f ba 0e 00 b4 09 cd 21 b8 01 4c cd 21 54 68 ........!..L.!Th
00000050 69 73 20 70 72 6f 67 72 61 6d 20 63 61 6e 6e 6f is program canno
00000060 74 20 62 65 20 72 75 6e 20 69 6e 20 44 4f 53 20 t be run in DOS
00000070 6d 6f 64 65 2e 0d 0d 0a 24 00 00 00 00 00 00 00 mode....$.......
00000080 6e 6e 0c 43 2a 0f 62 10 2a 0f 62 10 2a 0f 62 10 nn.C*.b.*.b.*.b.
00000090 0d c9 1f 10 2b 0f 62 10 23 77 f1 10 27 0f 62 10 ....+.b.#w..'.b.
000000a0 2a 0f 63 10 7d          00 3f       0f 62 10 *.c.}.b...?.(.b.
000000b0 e9 00 3d                         29 0f 62 10 ..=.+.b...m.).b.
000000c0 0d c                             2b 0f 62 10 ....=.b.....+.b.
000000d0 0d                                  0f 62 10 ....+.b.Rich*.b.
000000e0 0                                   00 00 00 ................
000000f0 5                                   00 00 00 PE..L...*..[....
00000100 0                                   24 00 00 ..............$..
00000110 00                                  10 00 00 ................
00000120 00 4                             00 02 00 00 .@....@.........
00000130 04 00                         00 00 00 00 00 ................
00000140 00 f0 00 00               00 00 02 00 40 81 ..............@.
000            00 10 00 00 00 00 10 00 00 10 00 00 ................
000            10 00 00 00 00 00 00 00 00 00 00 00 ................
000            54 00 00 00 00 70 00 00 10 00 00 00 LA..d....p......
               00 00 00 00 00 00 00 00 00 00 00 00 ................
               e0 01 00 00 00 00 00 00 00 00 00 00 ................
               00 00 00 00 00 00 00 00 00 00 00 00 ................
```

View in memory (via Process Hacker)

16 ANSI hex

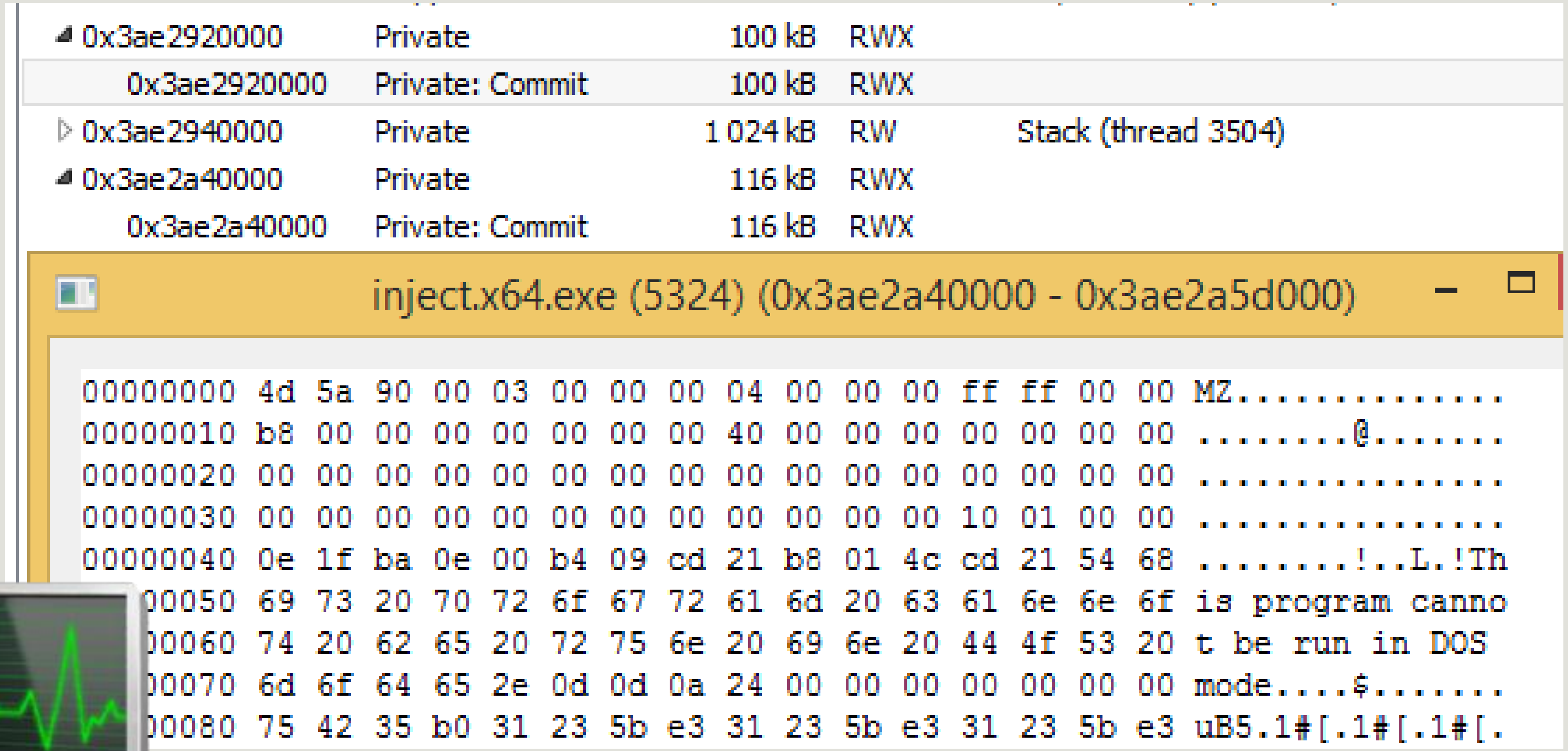
```
Offset(h) 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F

00000000  4D 5A 90 00 03 00 00 00 04 00 00 00 FF FF 00 00  MZ..........˙˙..
00000010  B8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00  ¸.......@.......
00000020  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000030  00 00 00 00 00 00 00 00 00 00 00 00 00 01 00 00  ................
00000040  0E 1F BA 0E 00 B4 09 CD 21 B8 01 4C CD 21 54 68  ..ş..´.Í!¸.LÍ!Th
00000050  69 73 20 70 72 6F 67 72 61 6D 20 63 61 6E 6E 6F  is program canno
00000060  74 20 62 65 20 72 75 6E 20 69 6E 20 44 4F 53 20  t be run in DOS
00000070  6D 6F 64 65                      0 00 00 00 00  mode....$.......
00000080  65 B3 D5                           21 D2 BB B0  ełŐă!Ň»°!Ň»°!Ň»°
00000090  95 4E                              B2 D2 BB B0  •NJ°*Ň»°•NH°¸Ň»°
000000A0  95 4E                               6 D2 BB B0  •NI°:Ň»°O‰¸±6Ň»°
000000B0  4F 89                                 D2 BB B0  O‰I±.Ň»°O‰ż±aŇ»°
000000C0  28 AA                                 D2 BB B0  (Ş(°*Ň»°!Ňş°«Ň»°
000000D0  F3 89                               0 D2 BB B0  ó‰¸± Ň»°ó‰D° Ň»°
          F3 89 B                            21 D2 BB B0  ó‰ą± Ň»°Rich!Ň»°
             00                            00 00 00 00 00  ................
             00 00 4C                      29 58 00 00 00 00  PE..L....—)X....
             00 00 E0 00 02 01 0B 01 0E 00 00 40 03 00  ....ŕ.........@..
             04 00 00 00 00 00 F1 8A 00 00 00 10 00 00  .ü......ńŠ......
```

Corresponding file on the disk (via HxD)

Detection: manually mapped PE

- **Working Set scan**
  - Search executable memory pages that are not a part of any module
  - Suspicious mapping type? Other indicators?
  - Are they part of a PE file? Detection of PE headers /artefacts

#1: Find the odd thing...



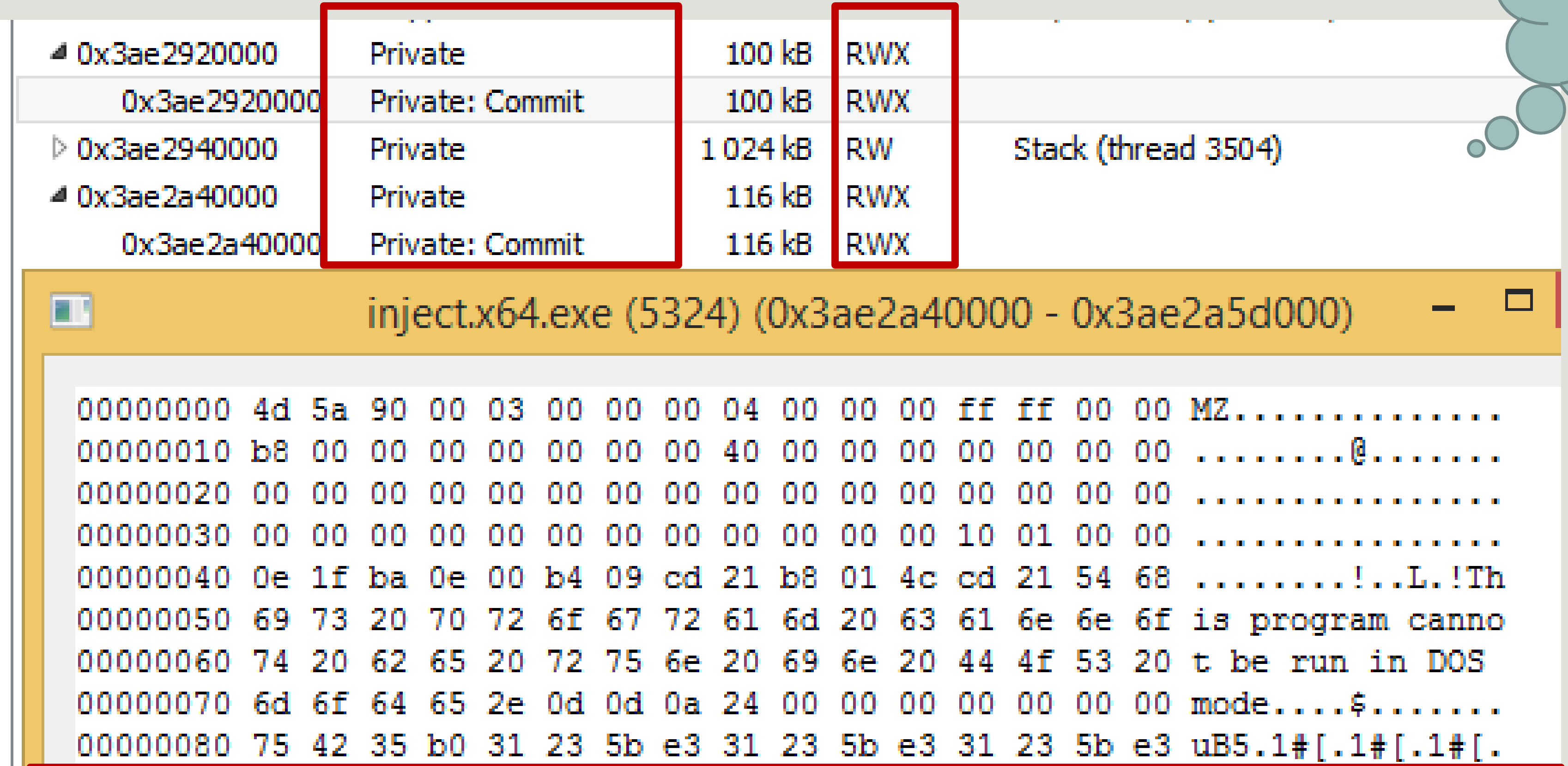
```
◢ 0x3ae2920000        Private              100 kB   RWX
    0x3ae2920000      Private: Commit      100 kB   RWX
▷ 0x3ae2940000        Private            1 024 kB   RW        Stack (thread 3504)
◢ 0x3ae2a40000        Private              116 kB   RWX
    0x3ae2a40000      Private: Commit      116 kB   RWX
```

inject.x64.exe (5324) (0x3ae2a40000 - 0x3ae2a5d000)

```
00000000 4d 5a 90 00 03 00 00 00 04 00 00 00 ff ff 00 00 MZ..............
00000010 b8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00 ........@.......
00000020 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00000030 00 00 00 00 00 00 00 00 00 00 00 00 10 01 00 00 ................
00000040 0e 1f ba 0e 00 b4 09 cd 21 b8 01 4c cd 21 54 68 ........!..L.!Th
   00050 69 73 20 70 72 6f 67 72 61 6d 20 63 61 6e 6e 6f is program canno
   00060 74 20 62 65 20 72 75 6e 20 69 6e 20 44 4f 53 20 t be run in DOS 
   00070 6d 6f 64 65 2e 0d 0d 0a 24 00 00 00 00 00 00 00 mode....$.......
   00080 75 42 35 b0 31 23 5b e3 31 23 5b e3 31 23 5b e3 uB5.1#[.1#[.1#[.
```

#1: Find the odd thing...

Reflective DLL injection



```
◢ 0x3ae2920000     Private            100 kB  RWX
    0x3ae2920000   Private: Commit    100 kB  RWX
▷ 0x3ae2940000     Private          1 024 kB  RW    Stack (thread 3504)
◢ 0x3ae2a40000     Private            116 kB  RWX
    0x3ae2a40000   Private: Commit    116 kB  RWX
```

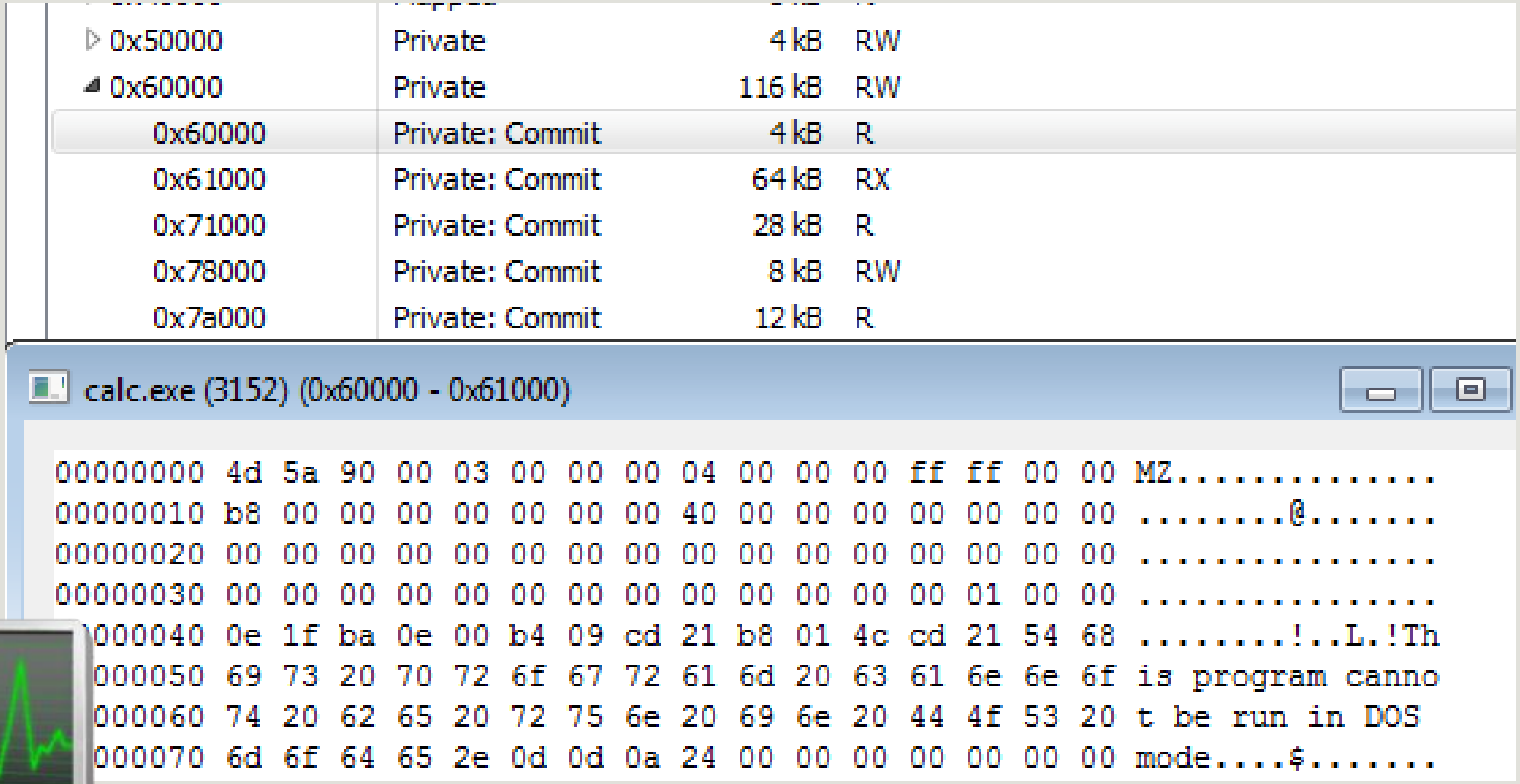
inject.x64.exe (5324) (0x3ae2a40000 - 0x3ae2a5d000)

```
00000000 4d 5a 90 00 03 00 00 00 04 00 00 00 ff ff 00 00 MZ..............
00000010 b8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00 ........@.......
00000020 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00000030 00 00 00 00 00 00 00 00 00 00 00 00 10 01 00 00 ................
00000040 0e 1f ba 0e 00 b4 09 cd 21 b8 01 4c cd 21 54 68 ........!..L.!Th
00000050 69 73 20 70 72 6f 67 72 61 6d 20 63 61 6e 6e 6f is program canno
00000060 74 20 62 65 20 72 75 6e 20 69 6e 20 44 4f 53 20 t be run in DOS 
00000070 6d 6f 64 65 2e 0d 0d 0a 24 00 00 00 00 00 00 00 mode....$.......
00000080 75 42 35 b0 31 23 5b e3 31 23 5b e3 31 23 5b e3 uB5.1#[.1#[.1#[.
```

[-] PE in MEM_PRIVATE (vs typical: MEM_IMAGE)

[-] RWX – very unusual protection

#2: Find the odd thing...



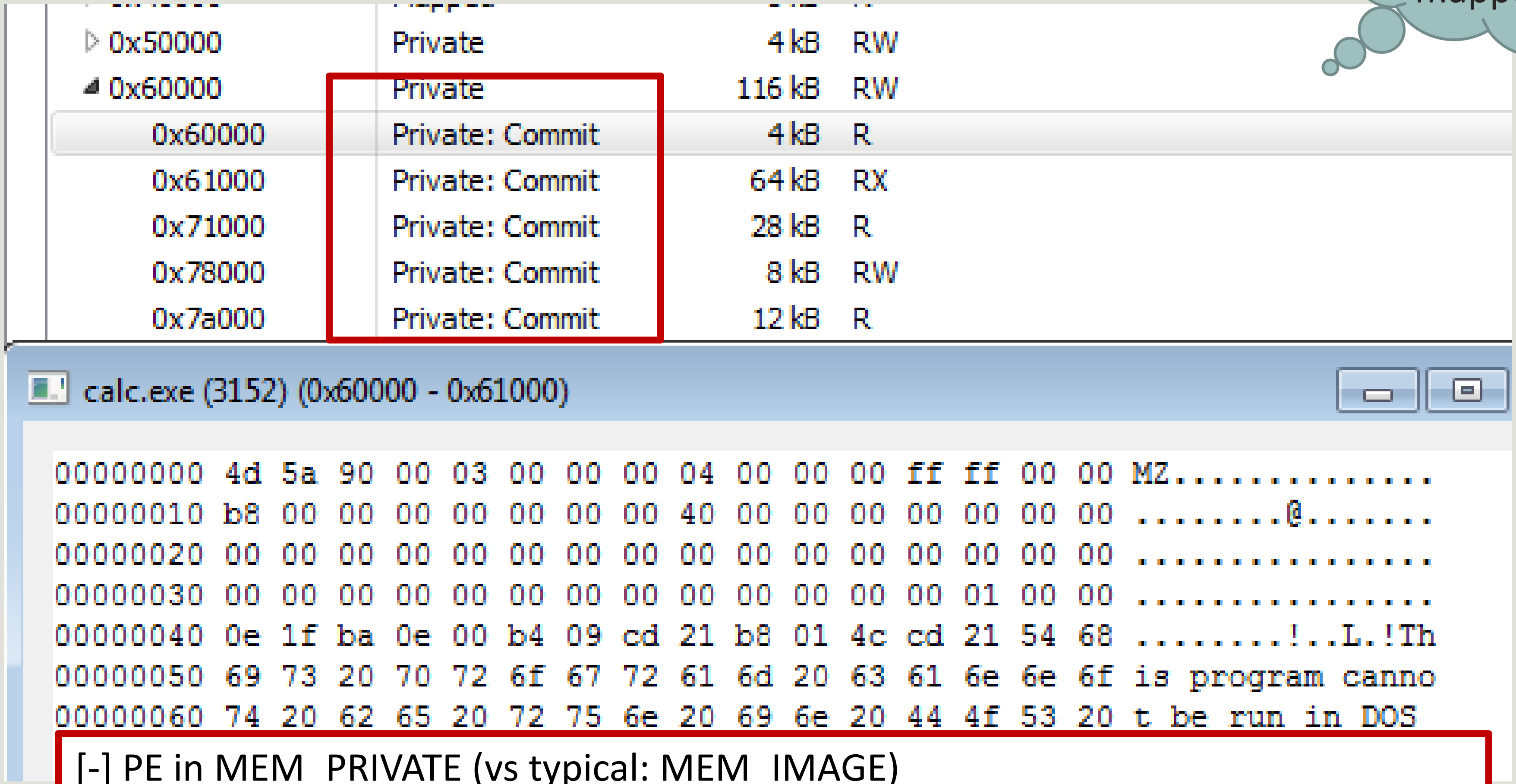
▷ 0x50000	Private	4 kB	RW
◢ 0x60000	Private	116 kB	RW
0x60000	Private: Commit	4 kB	R
0x61000	Private: Commit	64 kB	RX
0x71000	Private: Commit	28 kB	R
0x78000	Private: Commit	8 kB	RW
0x7a000	Private: Commit	12 kB	R

calc.exe (3152) (0x60000 - 0x61000)

```
00000000 4d 5a 90 00 03 00 00 00 04 00 00 00 ff ff 00 00 MZ..............
00000010 b8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00 ........@.......
00000020 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00000030 00 00 00 00 00 00 00 00 00 00 00 00 01 00 00 00 ................
 000040 0e 1f ba 0e 00 b4 09 cd 21 b8 01 4c cd 21 54 68 ........!..L.!Th
 000050 69 73 20 70 72 6f 67 72 61 6d 20 63 61 6e 6e 6f is program canno
 000060 74 20 62 65 20 72 75 6e 20 69 6e 20 44 4f 53 20 t be run in DOS 
 000070 6d 6f 64 65 2e 0d 0d 0a 24 00 00 00 00 00 00 00 mode....$.......
```

#2: Find the odd thing…

Process Hollowing or manually mapped PE



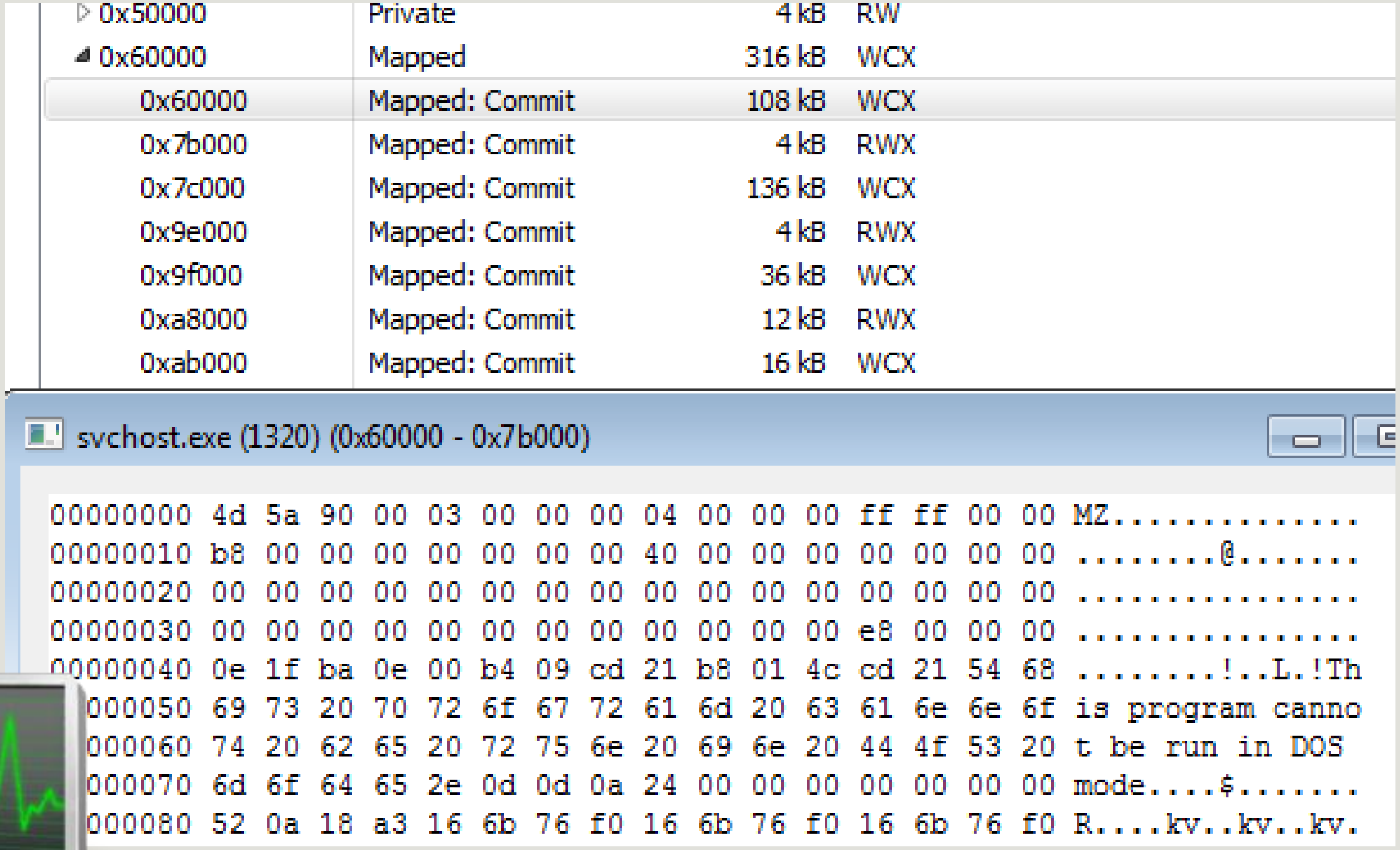
▷ 0x50000	Private	4 kB	RW
◢ 0x60000	Private	116 kB	RW
0x60000	Private: Commit	4 kB	R
0x61000	Private: Commit	64 kB	RX
0x71000	Private: Commit	28 kB	R
0x78000	Private: Commit	8 kB	RW
0x7a000	Private: Commit	12 kB	R

calc.exe (3152) (0x60000 - 0x61000)

```
00000000 4d 5a 90 00 03 00 00 00 04 00 00 00 ff ff 00 00 MZ..............
00000010 b8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00 ........@.......
00000020 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00000030 00 00 00 00 00 00 00 00 00 00 00 00 00 01 00 00 ................
00000040 0e 1f ba 0e 00 b4 09 cd 21 b8 01 4c cd 21 54 68 ........!..L.!Th
00000050 69 73 20 70 72 6f 67 72 61 6d 20 63 61 6e 6e 6f is program canno
00000060 74 20 62 65 20 72 75 6e 20 69 6e 20 44 4f 53 20 t be run in DOS
```

[-] PE in MEM_PRIVATE (vs typical: MEM_IMAGE)

#3: Find the odd thing...



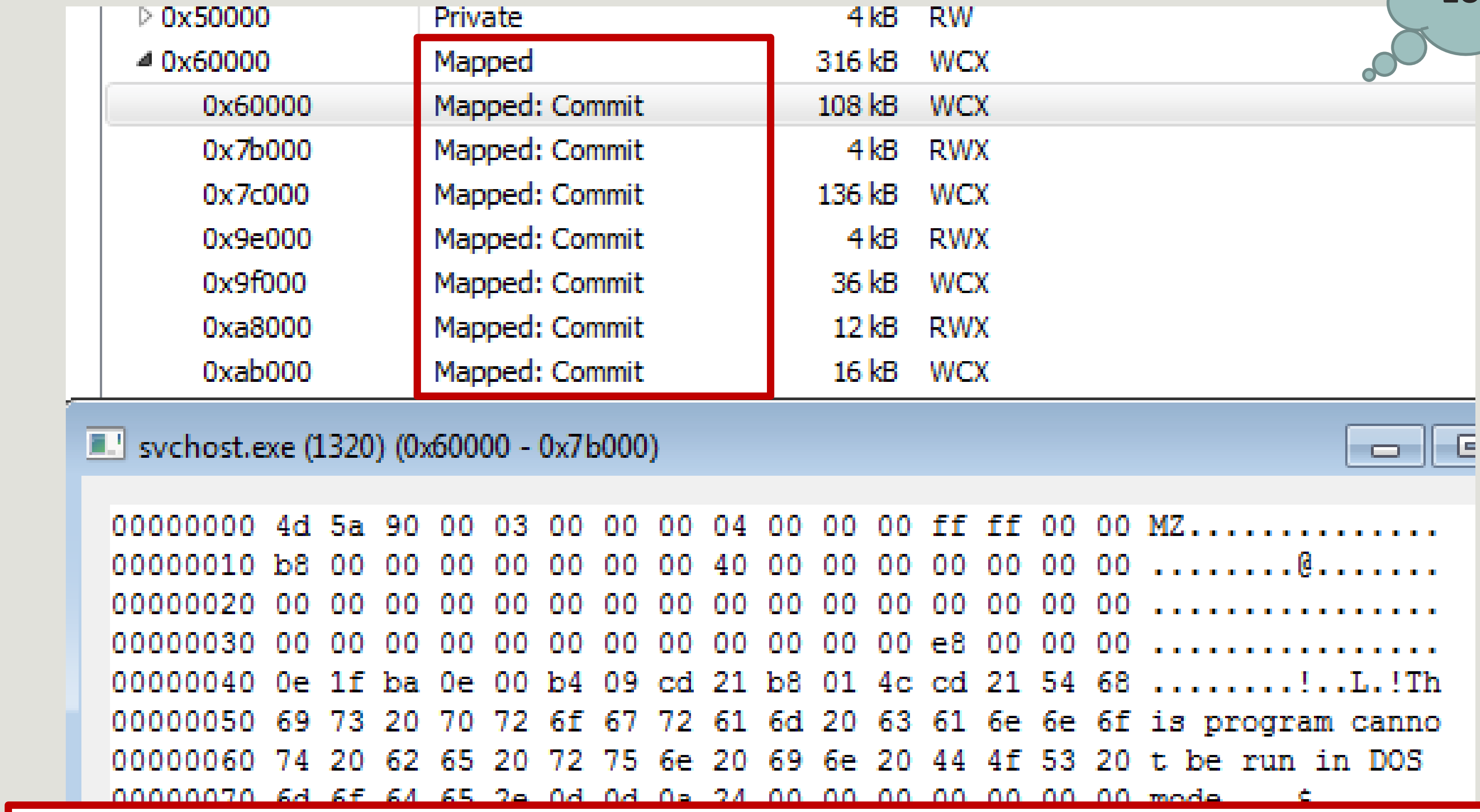
```
▷ 0x50000      Private              4 kB  RW
◢ 0x60000      Mapped             316 kB  WCX
    0x60000    Mapped: Commit     108 kB  WCX
    0x7b000    Mapped: Commit       4 kB  RWX
    0x7c000    Mapped: Commit     136 kB  WCX
    0x9e000    Mapped: Commit       4 kB  RWX
    0x9f000    Mapped: Commit      36 kB  WCX
    0xa8000    Mapped: Commit      12 kB  RWX
    0xab000    Mapped: Commit      16 kB  WCX
```

svchost.exe (1320) (0x60000 - 0x7b000)

```
00000000 4d 5a 90 00 03 00 00 00 04 00 00 00 ff ff 00 00 MZ..............
00000010 b8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00 ........@.......
00000020 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00000030 00 00 00 00 00 00 00 00 00 00 00 00 e8 00 00 00 ................
00000040 0e 1f ba 0e 00 b4 09 cd 21 b8 01 4c cd 21 54 68 ........!..L.!Th
 000050 69 73 20 70 72 6f 67 72 61 6d 20 63 61 6e 6e 6f is program canno
 000060 74 20 62 65 20 72 75 6e 20 69 6e 20 44 4f 53 20 t be run in DOS 
 000070 6d 6f 64 65 2e 0d 0d 0a 24 00 00 00 00 00 00 00 mode....$.......
 000080 52 0a 18 a3 16 6b 76 f0 16 6b 76 f0 16 6b 76 f0 R....kv..kv..kv.
```

#3: Find the odd thing…

Kronos Loader



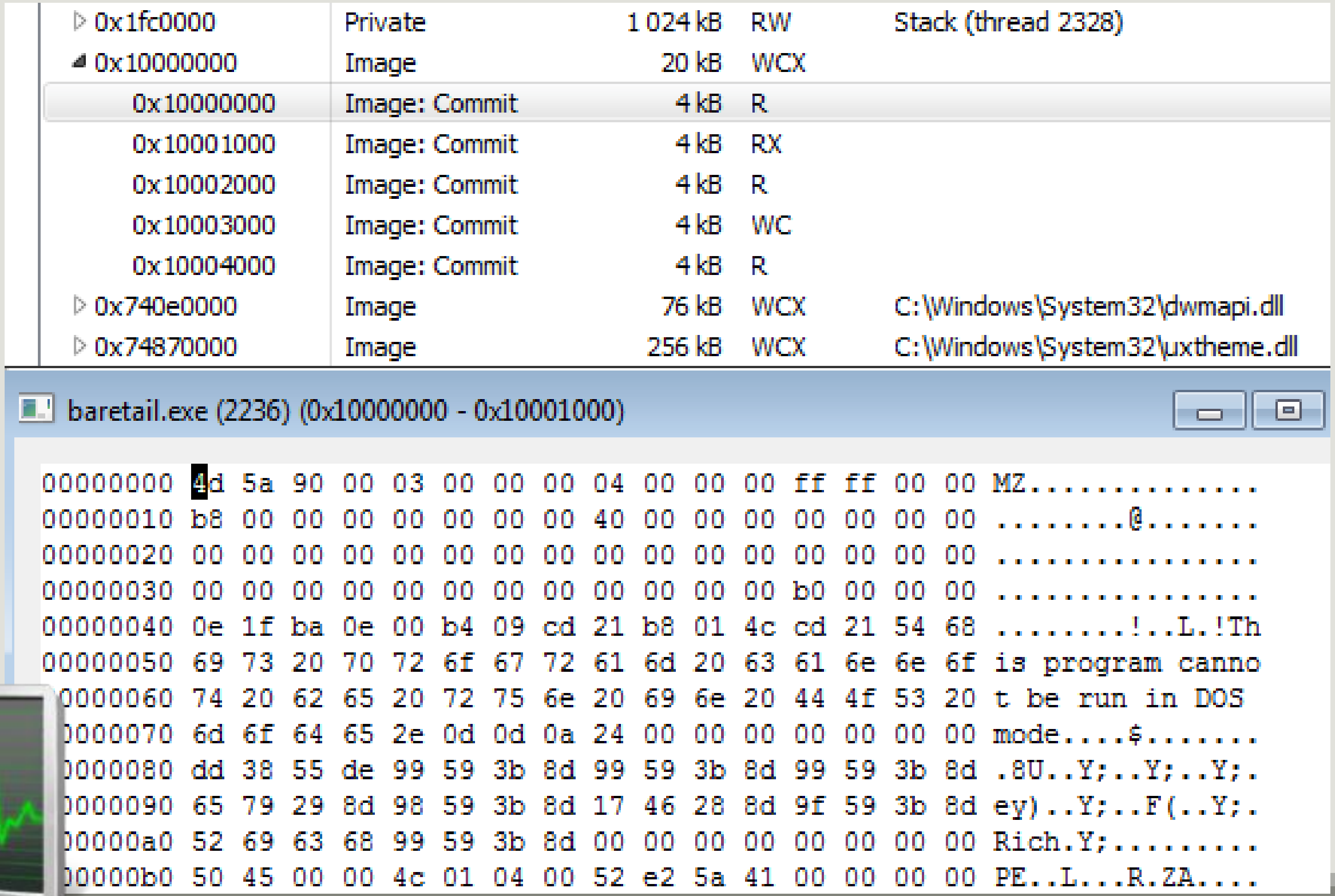
```
▷ 0x50000      Private            4 kB   RW
◢ 0x60000      Mapped           316 kB   WCX
    0x60000    Mapped: Commit   108 kB   WCX
    0x7b000    Mapped: Commit     4 kB   RWX
    0x7c000    Mapped: Commit   136 kB   WCX
    0x9e000    Mapped: Commit     4 kB   RWX
    0x9f000    Mapped: Commit    36 kB   WCX
    0xa8000    Mapped: Commit    12 kB   RWX
    0xab000    Mapped: Commit    16 kB   WCX
```

svchost.exe (1320) (0x60000 - 0x7b000)

```
00000000 4d 5a 90 00 03 00 00 00 04 00 00 00 ff ff 00 00 MZ..............
00000010 b8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00 ........@.......
00000020 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00000030 00 00 00 00 00 00 00 00 00 00 00 00 e8 00 00 00 ................
00000040 0e 1f ba 0e 00 b4 09 cd 21 b8 01 4c cd 21 54 68 ........!..L.!Th
00000050 69 73 20 70 72 6f 67 72 61 6d 20 63 61 6e 6e 6f is program canno
00000060 74 20 62 65 20 72 75 6e 20 69 6e 20 44 4f 53 20 t be run in DOS 
00000070 6d 6f 64 65 2e 0d 0d 0a 24 00 00 00 00 00 00 00 mode....$.......
```

[-] PE in MEM_MAPPED (vs typical: MEM_IMAGE)

#4: Find the odd thing…

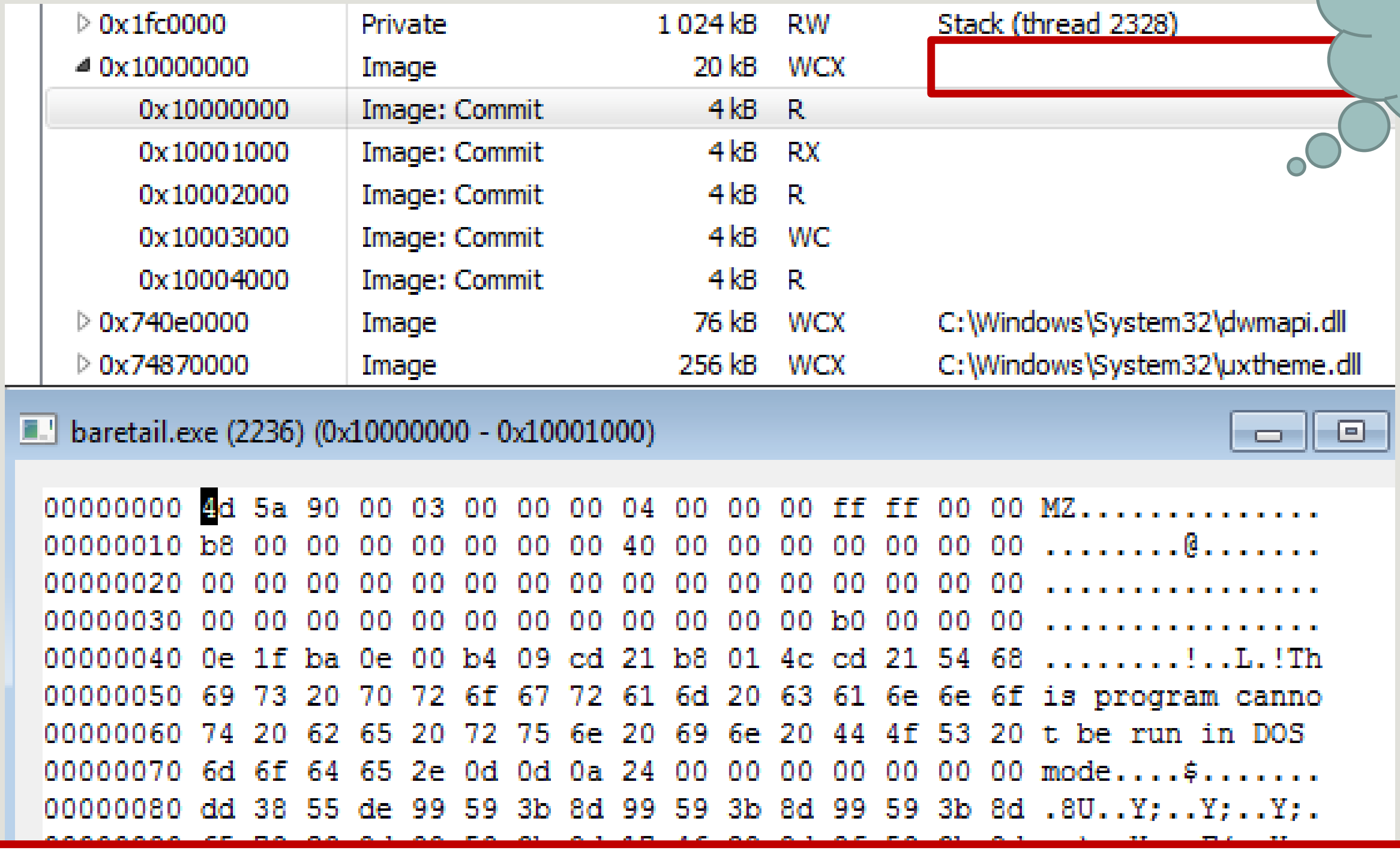


▷ 0x1fc0000	Private	1 024 kB	RW	Stack (thread 2328)
◢ 0x10000000	Image	20 kB	WCX	
0x10000000	Image: Commit	4 kB	R	
0x10001000	Image: Commit	4 kB	RX	
0x10002000	Image: Commit	4 kB	R	
0x10003000	Image: Commit	4 kB	WC	
0x10004000	Image: Commit	4 kB	R	
▷ 0x740e0000	Image	76 kB	WCX	C:\Windows\System32\dwmapi.dll
▷ 0x74870000	Image	256 kB	WCX	C:\Windows\System32\uxtheme.dll

baretail.exe (2236) (0x10000000 - 0x10001000)

```
00000000 4d 5a 90 00 03 00 00 00 04 00 00 00 ff ff 00 00 MZ..............
00000010 b8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00 ........@.......
00000020 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00000030 00 00 00 00 00 00 00 00 00 00 00 00 b0 00 00 00 ................
00000040 0e 1f ba 0e 00 b4 09 cd 21 b8 01 4c cd 21 54 68 ........!..L.!Th
00000050 69 73 20 70 72 6f 67 72 61 6d 20 63 61 6e 6e 6f is program canno
00000060 74 20 62 65 20 72 75 6e 20 69 6e 20 44 4f 53 20 t be run in DOS 
00000070 6d 6f 64 65 2e 0d 0d 0a 24 00 00 00 00 00 00 00 mode....$.......
00000080 dd 38 55 de 99 59 3b 8d 99 59 3b 8d 99 59 3b 8d .8U..Y;..Y;..Y;.
00000090 65 79 29 8d 98 59 3b 8d 17 46 28 8d 9f 59 3b 8d ey)..Y;..F(..Y;.
000000a0 52 69 63 68 99 59 3b 8d 00 00 00 00 00 00 00 00 Rich.Y;.........
000000b0 50 45 00 00 4c 01 04 00 52 e2 5a 41 00 00 00 00 PE..L...R.ZA....
```

#4: Find the odd thing...



```
▷ 0x1fc0000      Private          1 024 kB   RW     Stack (thread 2328)
◢ 0x10000000     Image               20 kB   WCX
    0x10000000   Image: Commit        4 kB   R
    0x10001000   Image: Commit        4 kB   RX
    0x10002000   Image: Commit        4 kB   R
    0x10003000   Image: Commit        4 kB   WC
    0x10004000   Image: Commit        4 kB   R
▷ 0x740e0000     Image               76 kB   WCX    C:\Windows\System32\dwmapi.dll
▷ 0x74870000     Image              256 kB   WCX    C:\Windows\System32\uxtheme.dll
```

baretail.exe (2236) (0x10000000 - 0x10001000)

```
00000000 4d 5a 90 00 03 00 00 00 04 00 00 00 ff ff 00 00 MZ..............
00000010 b8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00 ........@.......
00000020 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00000030 00 00 00 00 00 00 00 00 00 00 00 00 b0 00 00 00 ................
00000040 0e 1f ba 0e 00 b4 09 cd 21 b8 01 4c cd 21 54 68 ........!..L.!Th
00000050 69 73 20 70 72 6f 67 72 61 6d 20 63 61 6e 6e 6f is program canno
00000060 74 20 62 65 20 72 75 6e 20 69 6e 20 44 4f 53 20 t be run in DOS 
00000070 6d 6f 64 65 2e 0d 0d 0a 24 00 00 00 00 00 00 00 mode....$.......
00000080 dd 38 55 de 99 59 3b 8d 99 59 3b 8d 99 59 3b 8d .8U..Y;..Y;..Y;.
```

Process Doppelganging

[+] MEM_IMAGE -> OK
[-] PE Image has no path!

Summary

PE-sieve: current status

- Detecting anomalies
- Dumping payloads from memory
- Reconstructing corrupt payloads
- Read more:
  - https://github.com/hasherezade/pe-sieve/wiki

PE-sieve - TODO

- IAT/EAT hooking detection
- Classic DLL injection detection
- Whitelisting known hooks
- Bugs? Ideas?
  - https://github.com/hasherezade/pe-sieve/issues

Thank you!

Rate the talk: https://goo.gl/xHa2U1