Vortessence

Automating Memory Forensics

Area 41 Zurich

Endre Bangerter
Ramona Cioccarelli
Beni Urech

A project of the Security Engineering Lab
http://sel.bfh.ch
Bern University of Applied Sciences

Why Vortessence?

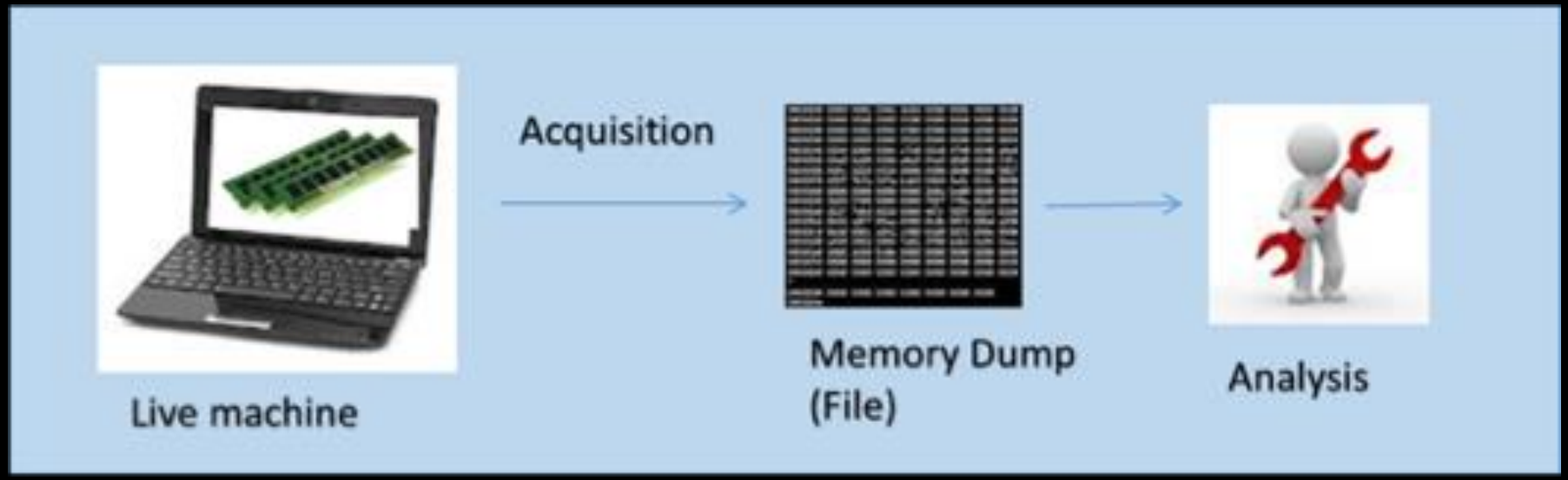
- **Memory forensics are key technique to detect malware / attack tools**

- Detection by finding anomalies
  - Need to memorize normal state of clean system (hard / impossible to memorize)
  - Looking for anomalies can be **rather mechanical and boring task**

- Vortessence **partially automates Volatility based forensics**
  - Better detection
  - Quicker
  - Will be open sourced later in 2014

- Memory forensics 101 - What's the problem?
- Finding plain system anomalies
- Improving the Volatility malware tools
- The role of whitelisting
- The future: conclusions and outlook

Memory forensic 101



Hunt for system anomalies, e.g., using Volatility

Processes - Example of an anomaly

```
$ vol.py -f image05.bin --profile=WinXPSP3x86 pstree
Volatility Foundation Volatility Framework 2.3.1
Name                                                  Pid   PPid   Thds   Hnds Time
-------------------------------------------------- ------ ------ ------ ------ ----
 0x825c8830:System                                      4      0     56    194 1970-01-01 00:00:00 UTC+0000
. 0x82172210:smss.exe                                 384      4      3     19 2013-12-06 09:48:00 UTC+0000
.. 0x82170878:csrss.exe                               608    384     13    377 2013-12-06 09:48:00 UTC+0000
.. 0x8221f7e8:winlogon.exe                            636    384     19    520 2013-12-06 09:48:02 UTC+0000
... 0x82162950:services.exe                           680    636     15    269 2013-12-06 09:48:02 UTC+0000
.... 0x824a9c08:vmacthlp.exe                          904    680      1     25 2013-12-06 09:48:02 UTC+0000

[snip]

.... 0x82499348:svchost.exe                          3148    680     44    840 2013-12-20 15:40:49 UTC+0000
.... 0x82043310:msiexec.exe                          1360    680      5    111 2013-12-20 15:41:04 UTC+0000
.... 0x8216f198:svchost.exe                           980    680     11    268 2013-12-06 09:48:02 UTC+0000
.... 0x82452248:svchost.exe                          1124    680      0 ------ 2013-12-06 09:48:02 UTC+0000
..... 0x82085020:wuauclt.exe                         1068   1124      8    258 2013-12-20 15:31:34 UTC+0000
..... 0x82113da0:wuauclt.exe                         1368   1124      4    123 2013-12-06 09:49:31 UTC+0000
... 0x82220da0:lsass.exe                              692    636     23    353 2013-12-06 09:48:02 UTC+0000
 0x821183d0:explorer.exe                             1780   1760     17    609 2013-12-06 09:48:07 UTC+0000
. 0x822352c8:vmtoolsd.exe                            1880   1780      3    245 2013-12-06 09:48:08 UTC+0000
. 0x82386438:ctfmon.exe                              1892   1780      1     78 2013-12-06 09:48:08 UTC+0000
. 0x84f0da88:explore.exe                             2400   1780      9    115 2013-12-20 15:40:44 UTC+0000
. 0x81f0da88:iexplore.exe                            2280   1780     13    387 2013-12-20 15:33:44 UTC+0000
.. 0x81eccb88:iexplore.exe                           2376   2280     23    648 2013-12-20 15:33:47 UTC+0000
```

Processes - Example of an anomaly


```
$ vol.py -f stuxnet.vmem --profile=WinXPSP3x86 pstree
Name                                                  Pid   PPid   Thds   Hnds Time
-------------------------------------------------- ------ ------ ------ ------ ----
0x823c8830:System                                       4      0     59    403 1970-01-01 00:00:00 UTC+0000
. 0x820df020:smss.exe                                 376      4      3     19 2010-10-29 17:08:53 UTC+0000
[SNIP]
.... 0x81f14938:ipconfig.exe                          304    968      0 ------ 2011-06-03 04:31:35 UTC+0000
.... 0x822843e8:svchost.exe                          1032    668     61   1169 2010-10-29 17:08:55 UTC+0000
..... 0x822b9a10:wuauclt.exe                          976   1032      3    133 2010-10-29 17:12:03 UTC+0000
..... 0x820ecc10:wscntfy.exe                         2040   1032      1     28 2010-10-29 17:11:49 UTC+0000
.... 0x81e61da0:svchost.exe                           940    668     13    312 2010-10-29 17:08:55 UTC+0000
.... 0x81db8da0:svchost.exe                           856    668     17    193 2010-10-29 17:08:55 UTC+0000
..... 0x81fa5390:wmiprvse.exe                        1872    856      5    134 2011-06-03 04:25:58 UTC+0000
.... 0x821a0568:VMUpgradeHelper                      1816    668      3     96 2010-10-29 17:09:08 UTC+0000
.... 0x81fee8b0:spoolsv.exe                          1412    668     10    118 2010-10-29 17:08:56 UTC+0000
.... 0x81ff7020:svchost.exe                          1200    668     14    197 2010-10-29 17:08:55 UTC+0000
.... 0x81c47c00:lsass.exe                            1928    668      4     65 2011-06-03 04:26:55 UTC+0000
.... 0x81e18b28:svchost.exe                          1080    668      5     80 2010-10-29 17:08:55 UTC+0000
.... 0x8205ada0:alg.exe                               188    668      6    107 2010-10-29 17:09:09 UTC+0000
.... 0x823315d8:vmacthlp.exe                          844    668      1     25 2010-10-29 17:08:55 UTC+0000
.... 0x81e0eda0:jqs.exe                              1580    668      5    148 2010-10-29 17:09:05 UTC+0000
.... 0x81c498c8:lsass.exe                             868    668      2     23 2011-06-03 04:26:55 UTC+0000
.... 0x82279998:imapi.exe                             756    668      4    116 2010-10-29 17:11:54 UTC+0000
... 0x81e70020:lsass.exe                              680    624     19    342 2010-10-29 17:08:54 UTC+0000
```

Processes - Example of an anomaly

```
$ vol.py -f image02.vmss --profile=Win7SP1x86  dlllist -p 544
services.exe pid:    544
Command line : C:\Windows\system32\services.exe

Base             Size  LoadCount Path
---------- ---------- ---------- ----
0x00210000    0x41000     0xffff C:\Windows\system32\services.exe
0x77080000   0x13c000     0xffff C:\Windows\SYSTEM32\ntdll.dll
0x76690000    0xd4000     0xffff C:\Windows\system32\kernel32.dll
0x753b0000    0x4b000     0xffff C:\Windows\system32\KERNELBASE.dll
0x765e0000    0xac000     0xffff C:\Windows\system32\msvcrt.dll
0x76d50000    0xa2000     0xffff C:\Windows\system32\RPCRT4.dll
0x76c40000    0x1f000        0x2 C:\Windows\system32\IMM32.DLL
0x75080000    0x29000        0x1 C:\Windows\system32\WINSTA.dll
[SNIP]
0x76d10000    0x35000        0x8 C:\Windows\system32\WS2_32.dll
0x76c60000     0x6000        0x8 C:\Windows\system32\NSI.dll
0x74af0000    0x3c000        0x4 C:\Windows\system32\mswsock.dll
0x74640000     0x5000        0x1 C:\Windows\System32\wshtcpip.dll
0x74ae0000     0x6000        0x1 C:\Windows\System32\wship6.dll
0x72bd0000     0x3000        0x1 C:\Windows\system32\lz32.DLL
0x74b30000    0x16000        0x1 C:\Windows\system32\CRYPTSP.dll
0x748d0000    0x3b000        0x1 C:\Windows\system32\rsaenh.dll
```

Zeroaccess

More tricky details…

Process priorities

…

Files

Unknown drivers

Number of DLLs per process

…

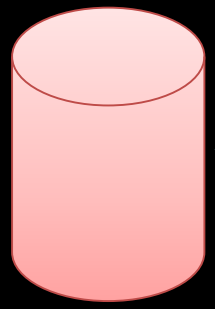
Network ports

Registry

- Memory forensics 101 - What's the problem?
- Finding plain system anomalies
- Improving the Volatility malware tools
- The role of whitelisting
- The future: conclusions and outlook

Vortessence – Basic idea

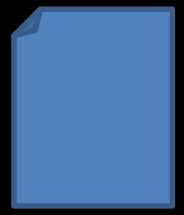
Clean systems



White list DB

Tools / scripts
using Volatility

Potentially
infected system



Report containing
real anomalies

Anomaly Detection Algorithm

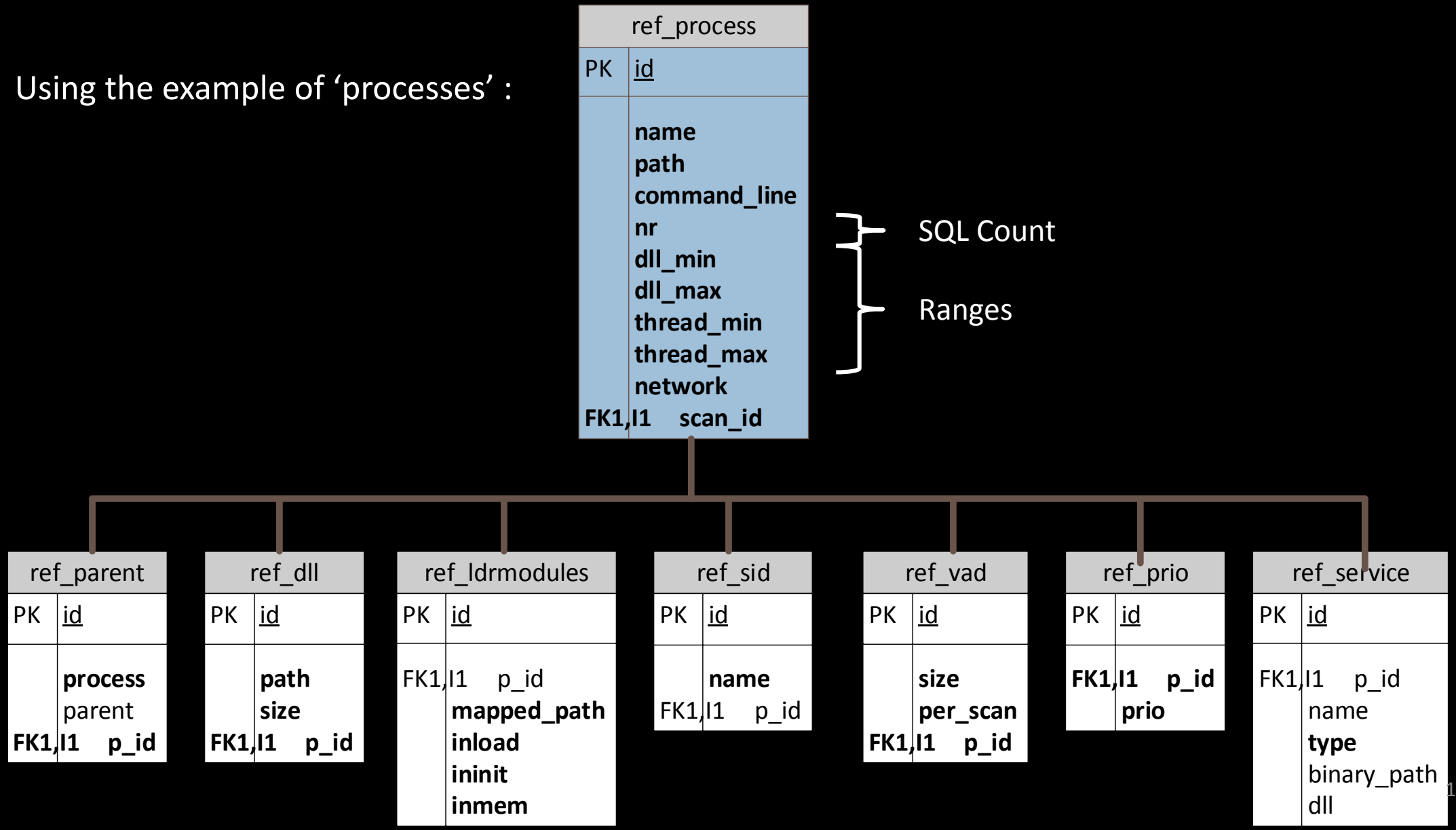
- For instance,
-

  - To many instances of this process: **3** | **1**
  - Unknown number of threads: **3** | **7 - 14**
  - Unknown number of DLLs: **13** | **52 - 62**
  - Unknown command line: "C:\Windows\\system32\\lsass.exe"
  - Unknown parent process: services.exe

  - Unknown command line values (reference!= snapshot)
  - Unknown number of DLLs (snapshot number < reference range > snapshot number)
  - .. and many more ..

Grouping of related Volatility outputs

- Using the example of ‘processes’ :



- Memory forensics 101 - What's the problem?
- Finding plain system anomalies
- Improving the Volatility malware tools
- The role of whitelisting
- The future: conclusions and outlook

Intro

- Some of the Volatility malware tools use heuristics that result in false positives
  - `malfind`
  - `apihooks`
  - `ldrmodules`

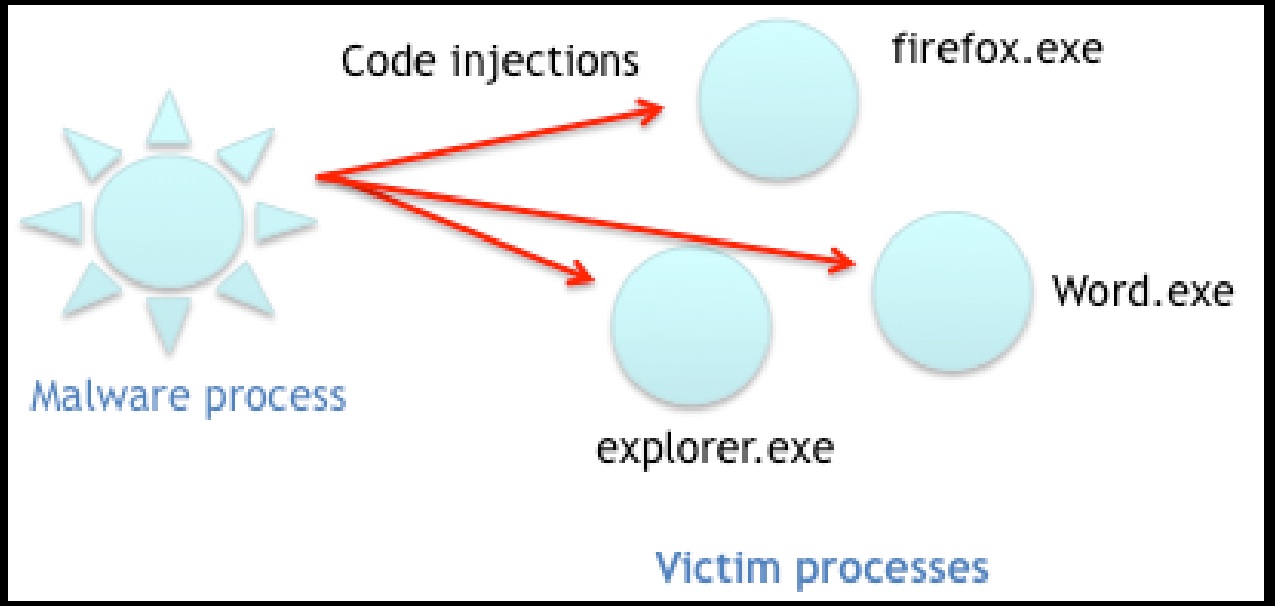
- Not criticizing Volatility, their heuristics are good and “false positives are normal”

- Two questions pop up here:
  - Can we use whitelisting to reduce false positive rate of Volatility malware tools?
  - Are there other techniques than the Volatility heuristics?

Malfind

Malware uses code injections

- Hide
- Gain privileges
- Corrupt process for information theft, modification etc.



- `malfind` is a volatility tool to detect code injections – based on some heuristics

malfind has false positives

```
Process: iexplore.exe Pid: 2376 Address: 0x1b10000
Vad Tag: VadS Protection: PAGE_EXECUTE_READWRITE
Flags: CommitCharge: 2, MemCommit: 1, PrivateMemory: 1, Protection: 6

0x01b10000  b0 00 eb 70 b0 01 eb 6c b0 02 eb 68 b0 03 eb 64   ...p...l...h...d
0x01b10010  b0 04 eb 60 b0 05 eb 5c b0 06 eb 58 b0 07 eb 54   ...`...\...X...T
0x01b10020  b0 08 eb 50 b0 09 eb 4c b0 0a eb 48 b0 0b eb 44   ...P...L...H...D
0x01b10030  b0 0c eb 40 b0 0d eb 3c b0 0e eb 38 b0 0f eb 34   ...@...<...8...4

0x1b10000 b000              MOV AL, 0x0
0x1b10002 eb70              JMP 0x1b10074
0x1b10004 b001              MOV AL, 0x1
0x1b10006 eb6c              JMP 0x1b10074
0x1b10008 b002              MOV AL, 0x2
0x1b1000a eb68              JMP 0x1b10074
0x1b1000c b003              MOV AL, 0x3
0x1b1000e eb64              JMP 0x1b10074
0x1b10010 b004              MOV AL, 0x4
0x1b10012 eb60              JMP 0x1b10074
```

False positive

Sort them out manually….

True positive

```
Process: chrome.exe Pid: 4052 Address: 0x130000
Vad Tag: VadS Protection: PAGE_EXECUTE_READWRITE
Flags: CommitCharge: 39, MemCommit: 1, PrivateMemory: 1, Protection: 6

0x00130000  4d 5a 00 00 00 00 00 00 00 00 00 00 00 00 00 00   MZ..............
0x00130010  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
0x00130020  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
0x00130030  00 00 00 00 00 00 00 00 00 00 00 00 d0 00 00 00   ................

0x130000 4d                 DEC EBP
0x130001 5a                 POP EDX
0x130002 0000               ADD [EAX], AL
0x130004 0000               ADD [EAX], AL
0x130006 0000               ADD [EAX], AL
0x130008 0000               ADD [EAX], AL
0x13000a 0000               ADD [EAX], AL
```

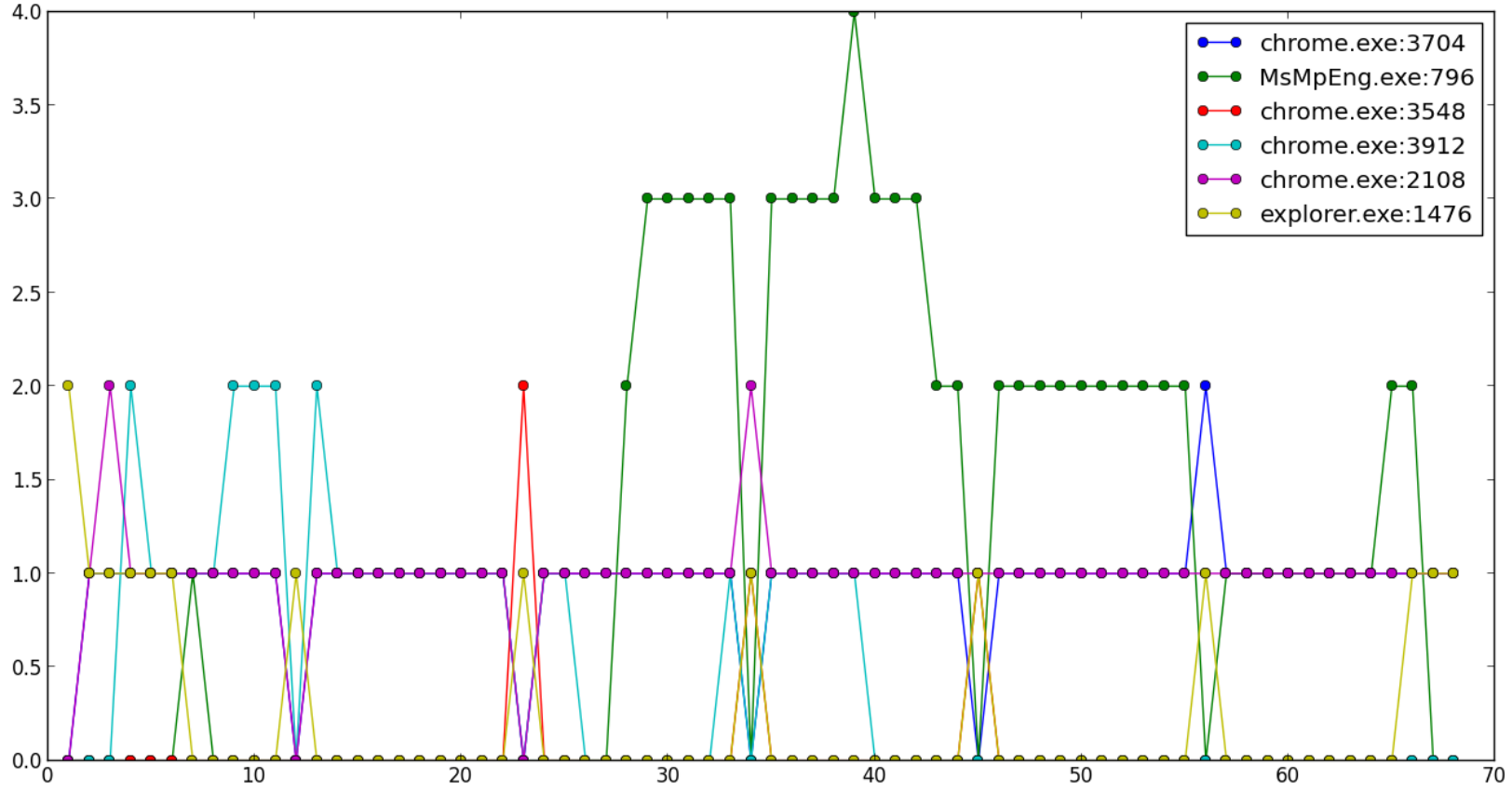
Understanding `malfind` false positives

- Need to better understand malware false positives
  - Does every process have false positives?
  - Are false positives constant over time?
  - ….

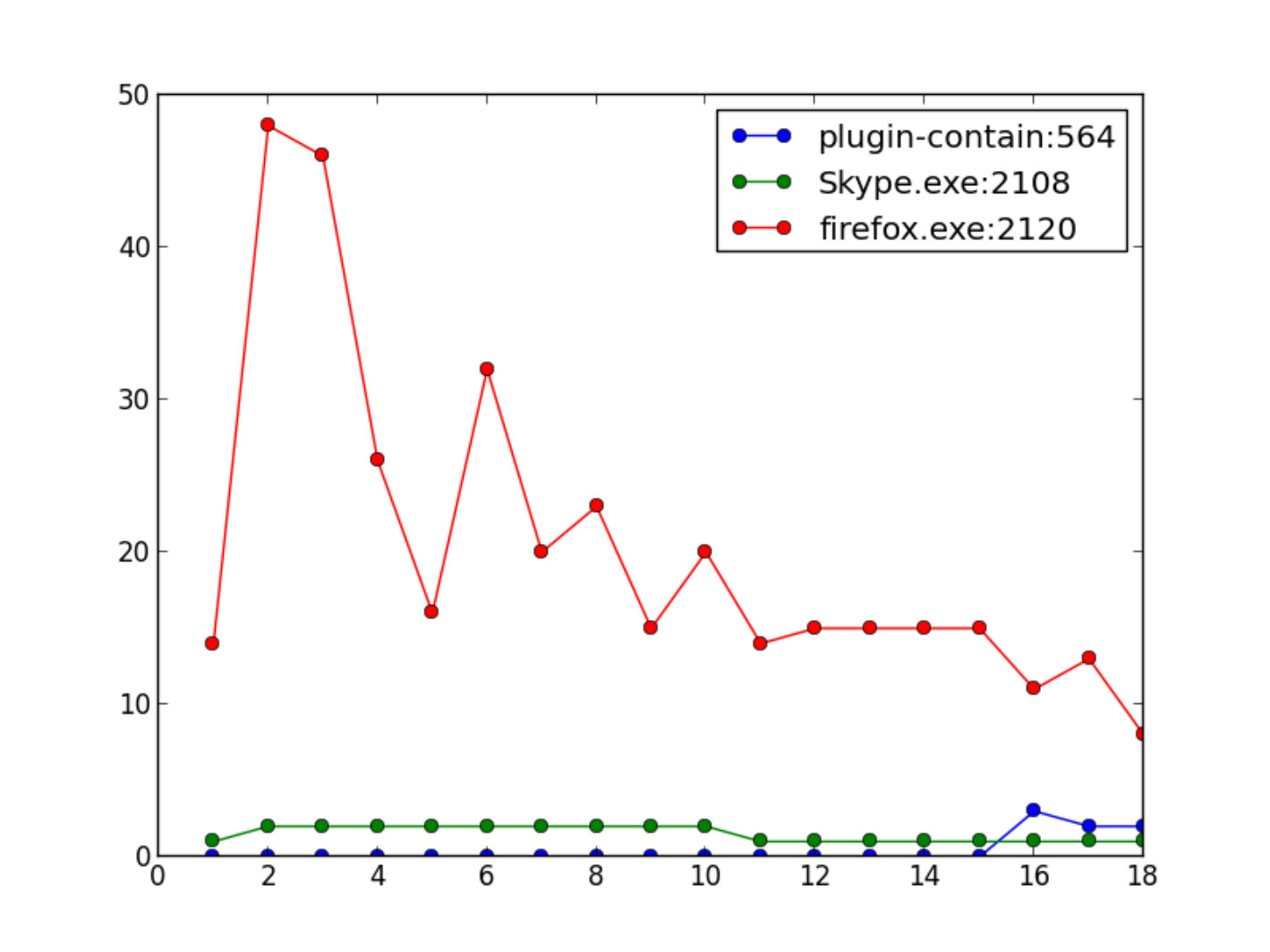
- Record memory snapshots on running system every 5 seconds

```
#!/usr/bin/env python
import subprocess
import time

num_snaps = 100
delta = 5

for i in range(0,num_snaps):
    subprocess.call("DumpIt.exe", shell=True)
    time.sleep(5)
```

Understanding `malfind` false positives



Understanding `malfind` false positives



Can `malfind` FPs be whitelisted?

- Done some experiments to answer question: Over many memory snapshots, checked if there are recurrent (identical) malfind false positives

- YES, some `malfind` false positives are recurrent across different OS installs and executions
  - "static content"

- But not all of them
  - Some malfind false positives seem to be unique
  - Probably dynamic content – don't know what exactly

- In summary, good news, malfind false positives are "whitelistable"

Zeroaccess example

```
[$ python mf_detector.py -t ../memory_traces/infected/zero_access/malfind-infected.txt –detect

services.exe
WL- process=   services.exe pid= 504 addr=  310000 vad_tag= Vad vad_protection=PAGE_EXECUTE_READWRITE commit_charge=None mem_commit=None priv_mem=None protection= 6

explorer.exe
WL+ process=   explorer.exe pid=1376 addr= 33f0000 vad_tag= VadS vad_protection=PAGE_EXECUTE_READWRITE commit_charge= 2 mem_commit=   1 priv_mem= 1 protection= 6
WL+ process=   explorer.exe pid=1376 addr= 2e10000 vad_tag= VadS vad_protection=PAGE_EXECUTE_READWRITE commit_charge= 1 mem_commit=   1 priv_mem= 1 protection= 6
WL- process=   explorer.exe pid=1376 addr= 2c60000 vad_tag= Vad vad_protection=PAGE_EXECUTE_READWRITE commit_charge=None mem_commit=None priv_mem=None protection= 6

SearchFilterHo
Process known to have 1-2 false positives
WL- process= SearchFilterHo pid=2352 addr=  ad0000 vad_tag= VadS vad_protection=PAGE_EXECUTE_READWRITE commit_charge= 1 mem_commit=None priv_mem= 1 protection=  6
WL- process= SearchFilterHo pid=2352 addr=  8f0000 vad_tag= VadS vad_protection=PAGE_EXECUTE_READWRITE commit_charge= 1 mem_commit=None priv_mem= 1 protection=  6
```

Not in white list

White list hit

Process is known to have 1-2 false positive

Not in white list

API hooks

apihooks

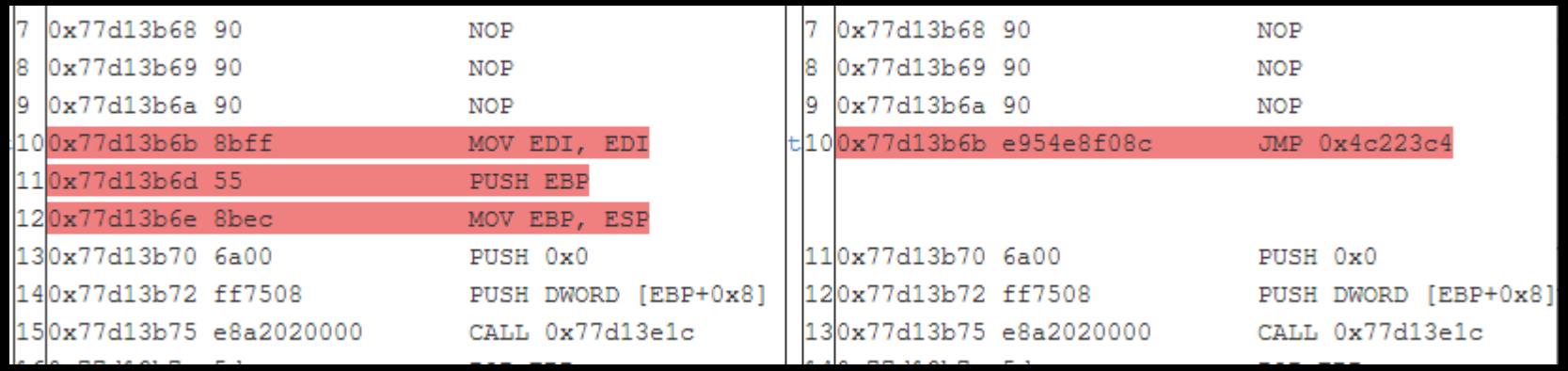
To find API hooks in user mode or kernel mode, use the apihooks plugin. This finds IAT, EAT, Inline style hooks, and several special types of hooks. For Inline hooks, it detects CALLs and JMPs to direct and indirect locations, and it detects PUSH/RET instruction sequences. It also detects CALLs or JMPs to registers after an immediate value (address) is moved into the register. The special types of hooks that it detects include syscall hooking in ntdll.dll and calls to unknown code pages in kernel memory.

- Uses simple but good heuristics
- Very slow
- Massive amount of FPs: between some hundreds to thousands per snapshot
- 75% FPs filterable with whitelist
- Filtered results contain few FPs

PE file diffing

Idea:

- Code whitelisting: store known good code in whitelist
- Find API hooks and injections by comparing code sections of PE files byte-by-byte



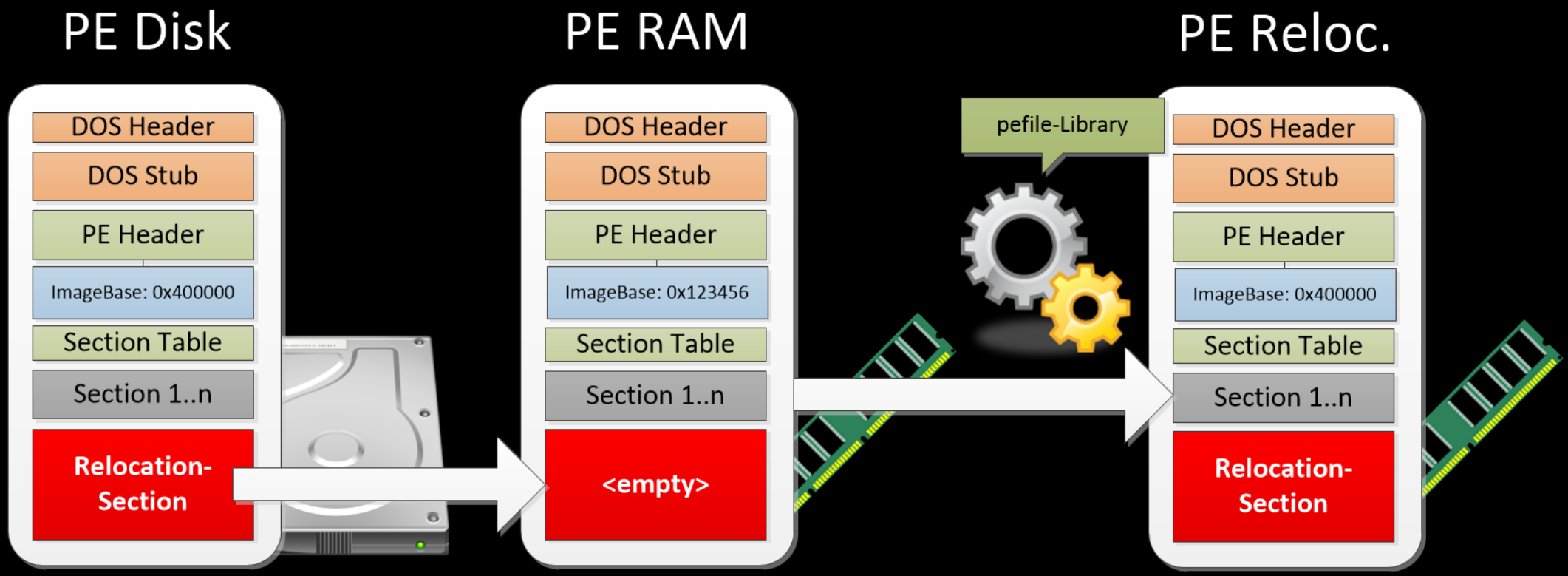
7	0x77d13b68	90	NOP	7	0x77d13b68	90	NOP
8	0x77d13b69	90	NOP	8	0x77d13b69	90	NOP
9	0x77d13b6a	90	NOP	9	0x77d13b6a	90	NOP
10	0x77d13b6b	8bff	MOV EDI, EDI	10	0x77d13b6b	e954e8f08c	JMP 0x4c223c4
11	0x77d13b6d	55	PUSH EBP				
12	0x77d13b6e	8bec	MOV EBP, ESP				
13	0x77d13b70	6a00	PUSH 0x0	11	0x77d13b70	6a00	PUSH 0x0
14	0x77d13b72	ff7508	PUSH DWORD [EBP+0x8]	12	0x77d13b72	ff7508	PUSH DWORD [EBP+0x8]
15	0x77d13b75	e8a2020000	CALL 0x77d13e1c	13	0x77d13b75	e8a2020000	CALL 0x77d13e1c

Advantages:

- No heuristic ➔ less FPs, can’t be circumvented easily
- Performance

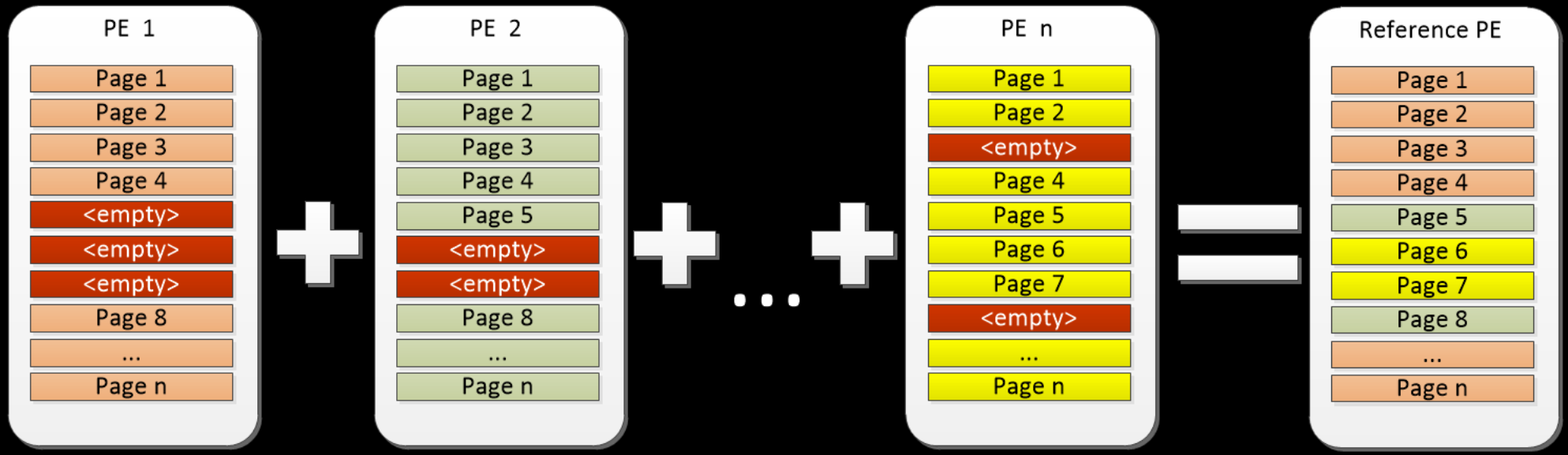
Dealing with ASLR

- Problem: direct comparison of dumped PE files not possible
- Solution: relocation section from disk



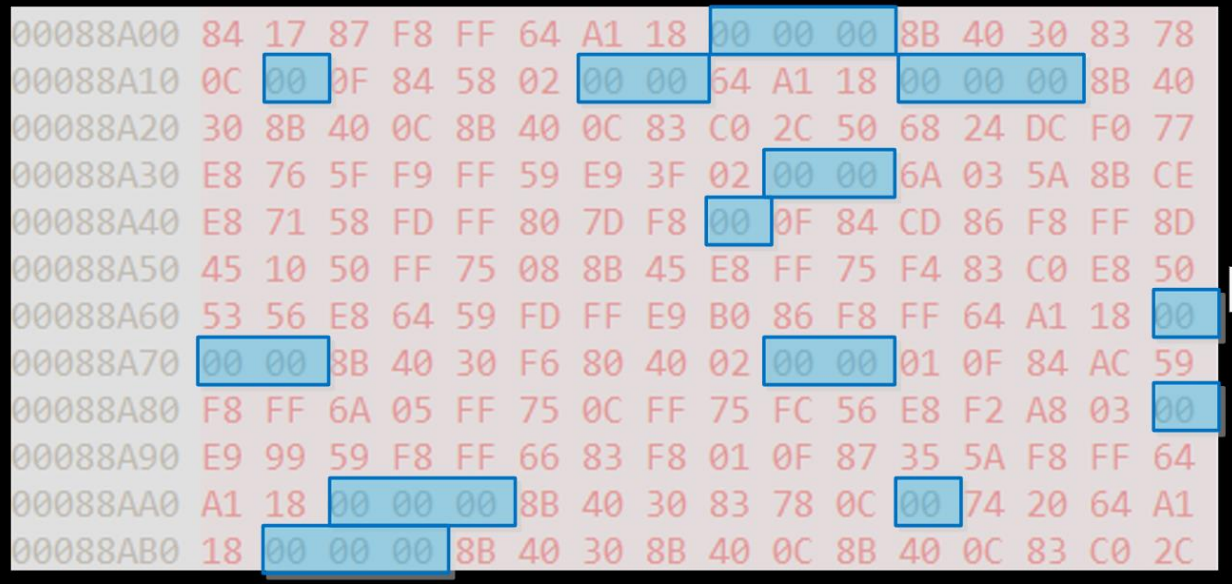
Filling empty pages

- Problem: empty pages in memory (not loaded, swapped)
- Solution: assemble reference PE file with multiple dumps



Ignoring variable code parts

- Problem: variable code parts (not analyzed further)
- Solution: Exclude them from the comparison



Whitelist in database

Offset	*Length*
88A08	3
88A11	1
88A16	2
88A1B	3
88A39	2
…	…
88AB1	3

Parts containing variable code or data

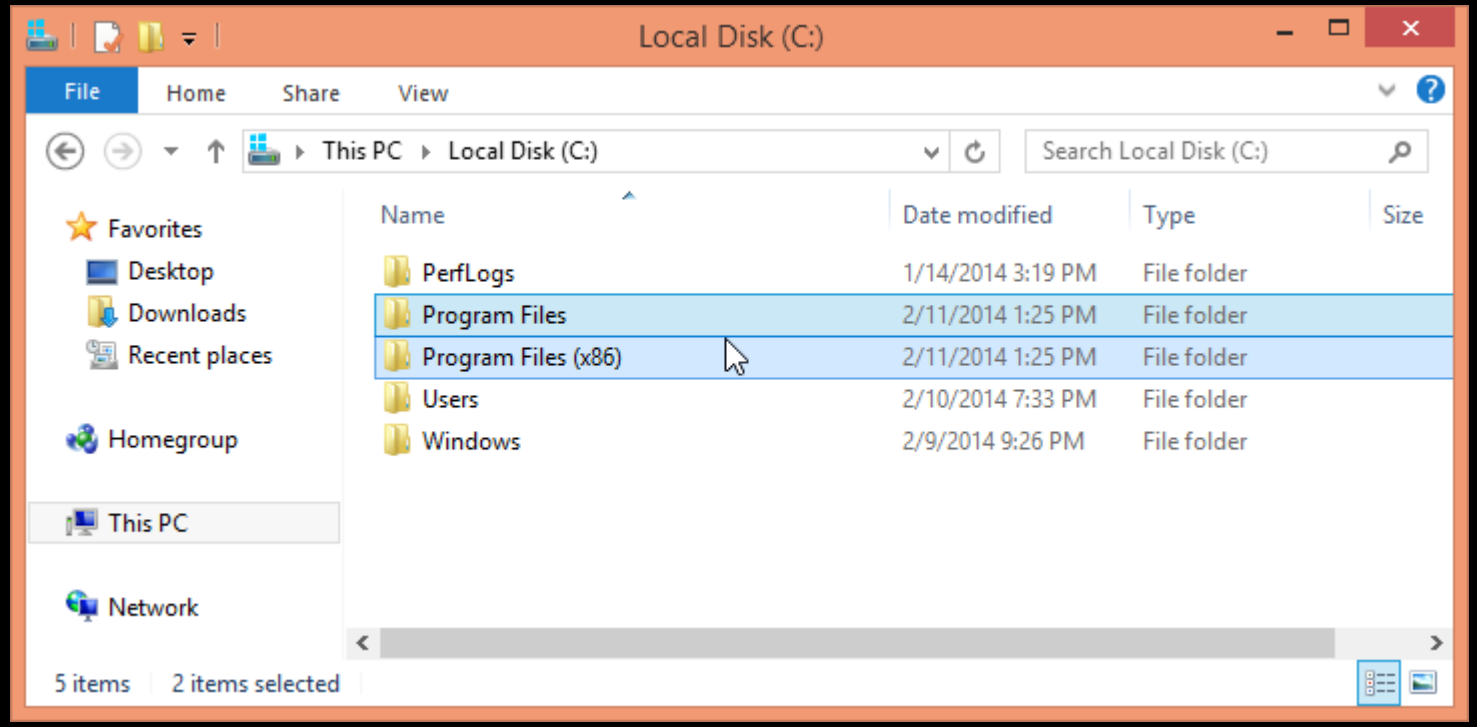
Static code or data

Conclusion

- API hook whitelisting and PE file diffing deliver good results
  - PE file diffing results better than unfiltered apihooks
- Matching the result helps identifying true positives
  - In combination, FPs unlikely
- With further improvement the PE file diffing method promises to give better results than whitelisted apihooks
  - No heuristics
- Target: Replace apihooks in Vortessence by PE file diffing
  - Todo: deal with ASLR without access to relocation section, fix algorithms to reduce FPs

- Memory forensics 101 - What's the problem?
- Finding plain system anomalies
- Improving the Volatility malware tools
- The role of whitelisting
- The future: conclusions and outlook

Whitelisting problems



- Differences in OS versions
  - 32 vs. 64 bit architecture
  - WinXP, Vista, Win7, Win8

- Software Updates
  - Every software version has to be mapped to the database

Whitelisting problems

- Whitelist with little or missing information results in false positives
- Can we assemble “the complete whitelist”?

Whitelist & Versions

- Not only Software / PE Versions

- Whitelist versions
  - System
  - Purpose

- Memory forensics 101 - What's the problem?
- Finding plain system anomalies
- Improving the Volatility malware tools
- The role of whitelisting
- The future: conclusions and outlook

Conclusions & outlook

- Vortessence, using straightforward whitelisting approach can automate substantial parts of memory analysis
- Vortessence is complementary to Volatility, not competition
- Will be released in fall 2014
  - Have “fully” working system now
  - Need to refactor, need to review rules, etc. before release
  - Check http://sel.bfh.ch or other channels
- Future work
  - Better understanding of whitelists, how good can they get
  - More fancy detection techniques than the simple rules used currently
  - Platform support: Linux, OS X,…

Zeroaccess example

```
[$ python mf_detector.py -t ../memory_traces/infected/zero_access/malfind-infected.txt –detect


services.exe
WL- process=   services.exe pid= 504 addr=  310000 vad_tag= Vad vad_protection=PAGE_EXECUTE_READWRITE commit_charge=None mem_commit=None priv_mem=None protection= 6

explorer.exe
WL+ process=   explorer.exe pid=1376 addr= 33f0000 vad_tag= VadS vad_protection=PAGE_EXECUTE_READWRITE commit_charge= 2 mem_commit=   1 priv_mem= 1 protection= 6
WL+ process=   explorer.exe pid=1376 addr= 2e10000 vad_tag= VadS vad_protection=PAGE_EXECUTE_READWRITE commit_charge= 1 mem_commit=   1 priv_mem= 1 protection= 6
WL- process=   explorer.exe pid=1376 addr= 2c60000 vad_tag= Vad vad_protection=PAGE_EXECUTE_READWRITE commit_charge=None mem_commit=None priv_mem=None protection= 6

SearchFilterHo
Process known to have 1-2 false positives
WL- process= SearchFilterHo pid=2352 addr=  ad0000 vad_tag= VadS vad_protection=PAGE_EXECUTE_READWRITE commit_charge= 1 mem_commit=None priv_mem= 1 protection=  6
WL- process= SearchFilterHo pid=2352 addr=  8f0000 vad_tag= VadS vad_protection=PAGE_EXECUTE_READWRITE commit_charge= 1 mem_commit=None priv_mem= 1 protection=  6
```