Live Memory Attacks and Forensics

2020-02-15

UlfFrisk

MemProcFS: Fast easy-to-use Memory Analysis

PCILeech: Direct Memory Access (DMA) Attack Software

Agenda

LIVE MEMORY DMA ATTACKS with **PCILeech**

PWN Linux and Windows with DMA code injection

MOUNT live file systems and spawn **SHELLS**

LIVE MEMORY FORENSICS with **MemProcFS**

What is The Memory Process File System?

In-Depth: Capabilities Design, API and Plugins

Demos - **Live Demos**!

whoami: Ulf Frisk

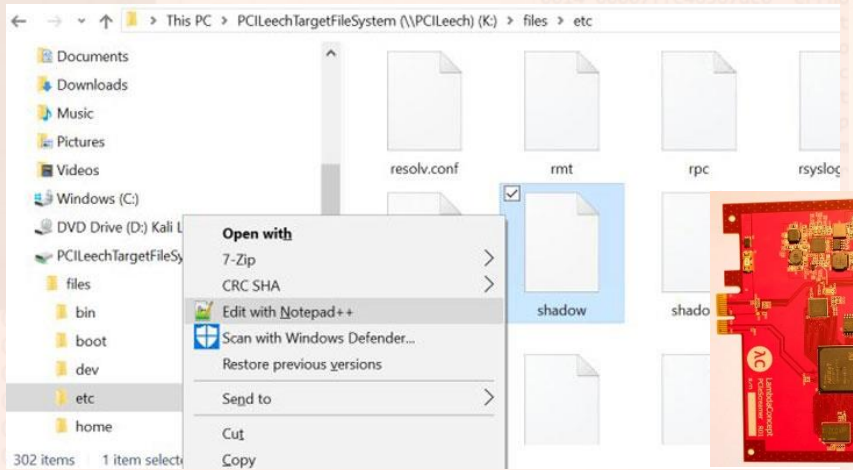
Pentester by day @ Polisen internal IT – Stockholm, Sweden

Security Researcher by night

Author of the PCILeech Direct Memory Acccess Attack Toolkit

Presented at different cons including DEF CON, BlueHat and the CCC

100% Open Source



PCILeech and MemProcFS

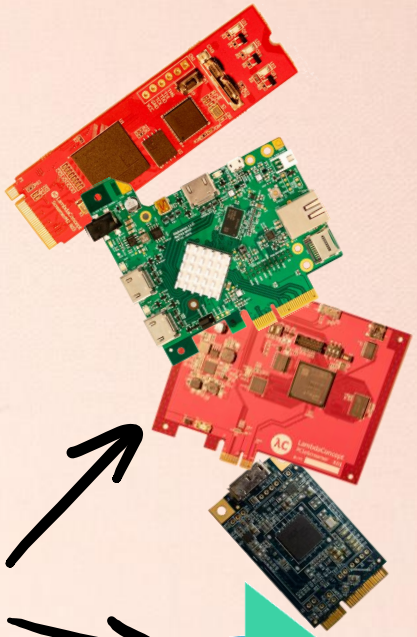
PCILeech

Direct Memory Access
Attack toolkit

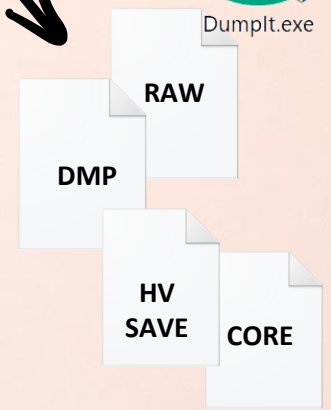
MemProcFS

High Speed Memory
Analysis and Forensics

LeechCore Library



Dumplt.exe



PCILeech

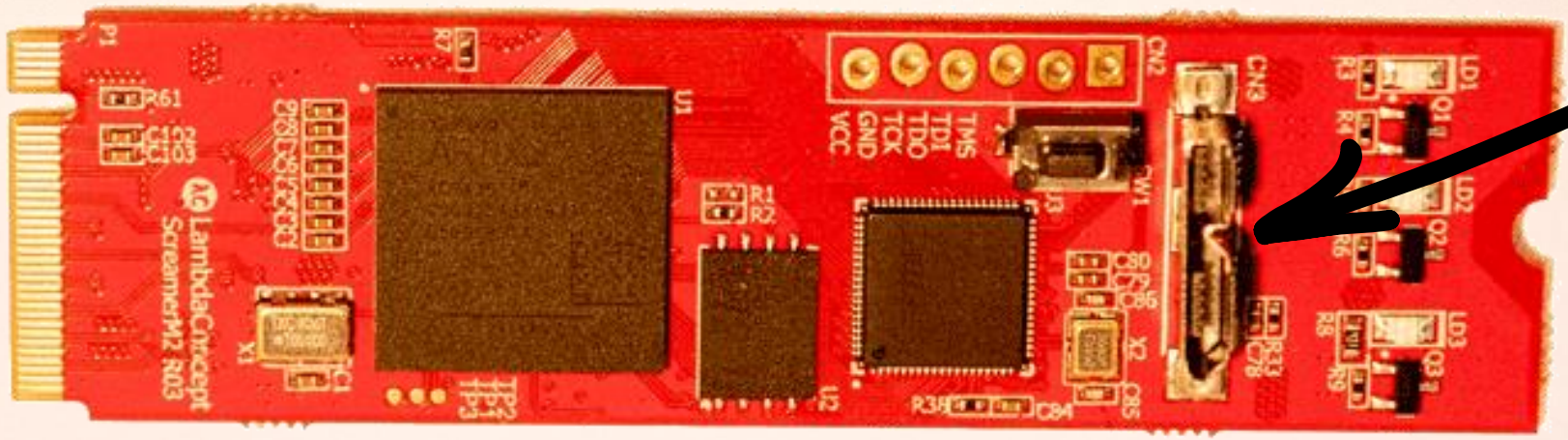
DEMO: LINUX

github.com/ufrisk/PCILeech

The Screamer M.2

Send and Receive PCIe Transaction Layer Packets (TLPs)
over USB3 onto PCIe

USB3



PCIe

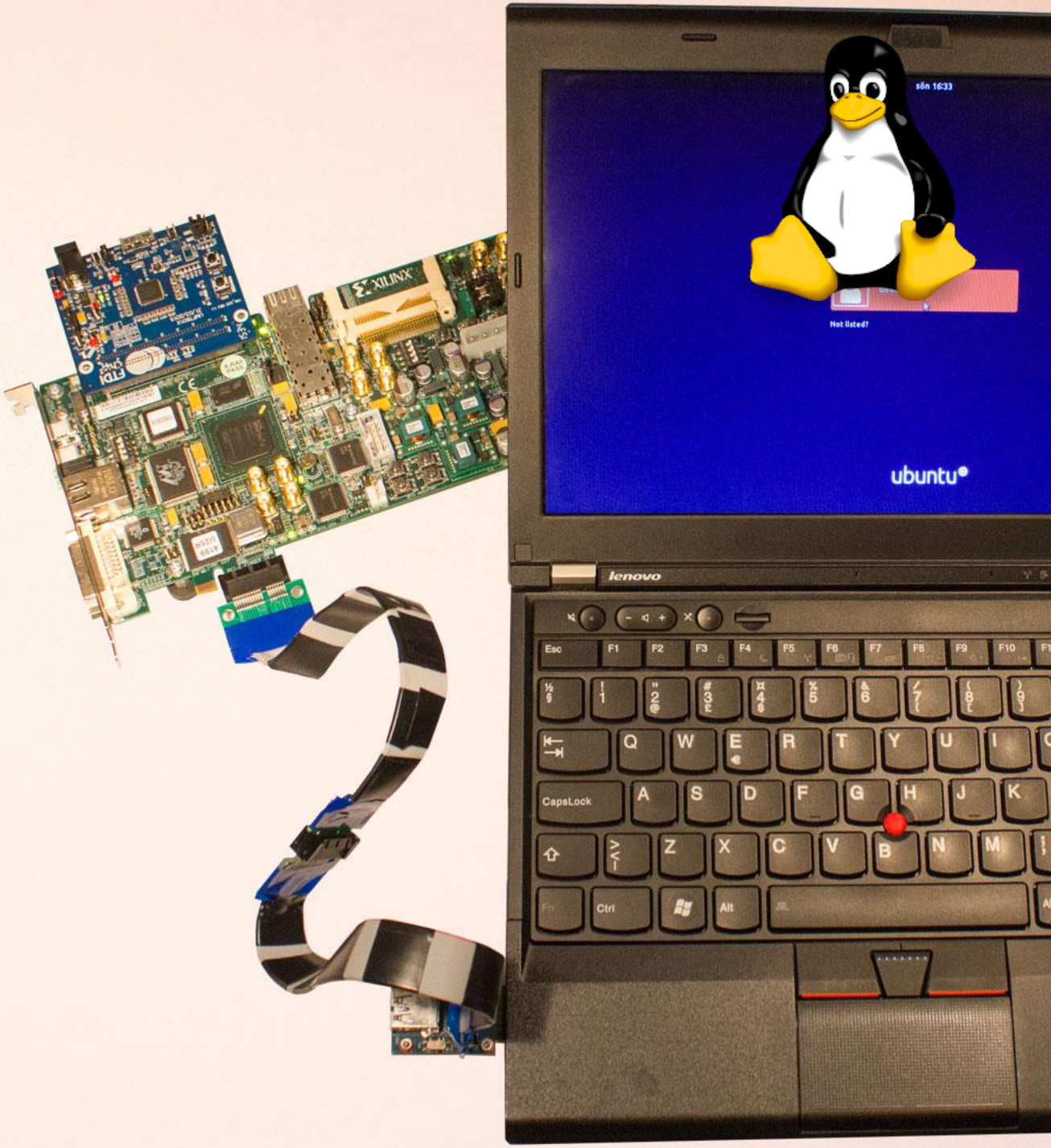
~150MB/s **€265**

Sold by 3rd party vendor. PCIeScreamer hardware is not directly related to PCILeech/MemProcFS projects.

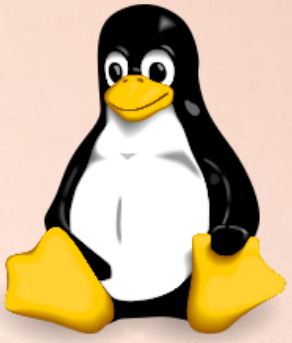
Linux DEMO

KERNEL IMPLANT
MOUNT file system
UNLOCK

Target System: Kali 2019.4



Linux Security

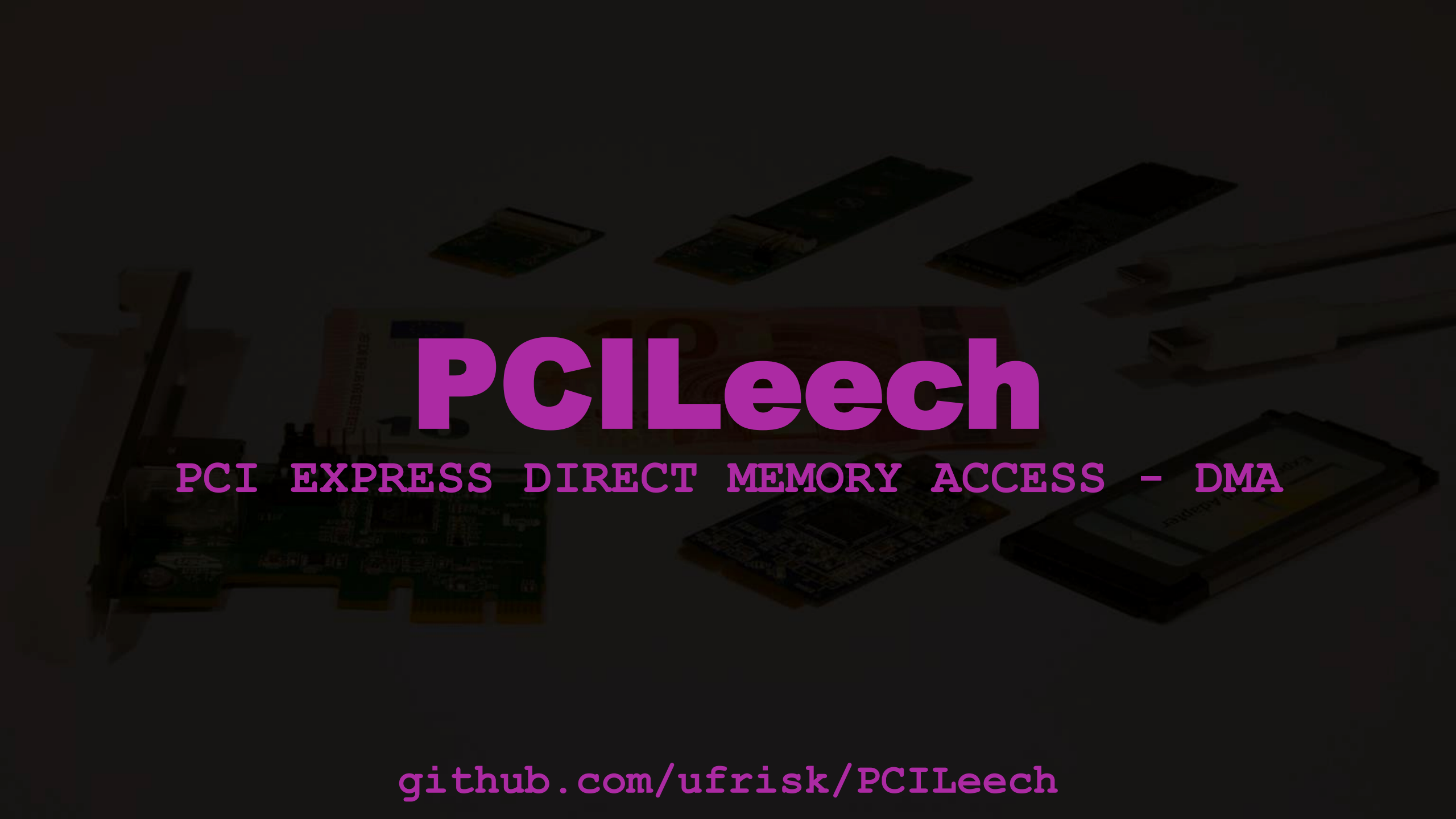


Hardware without DMA ports

BIOS password, DMA port lockdown, Pre-boot authentication

IOMMU / VT-d (virtualization-based security)

 **== secure by default!**



PCILeech

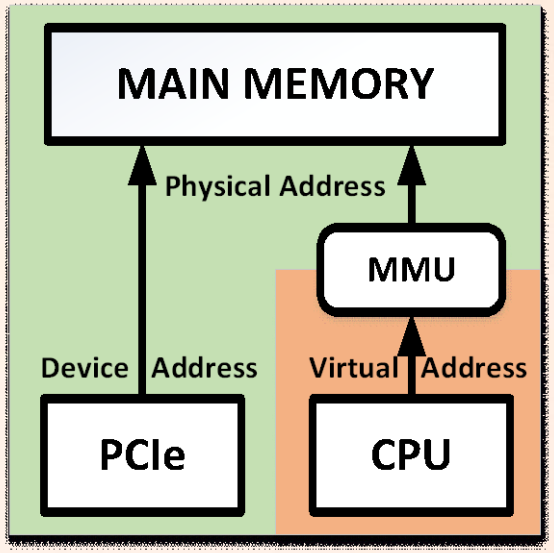
PCI EXPRESS DIRECT MEMORY ACCESS - DMA

github.com/ufrisk/PCILeech

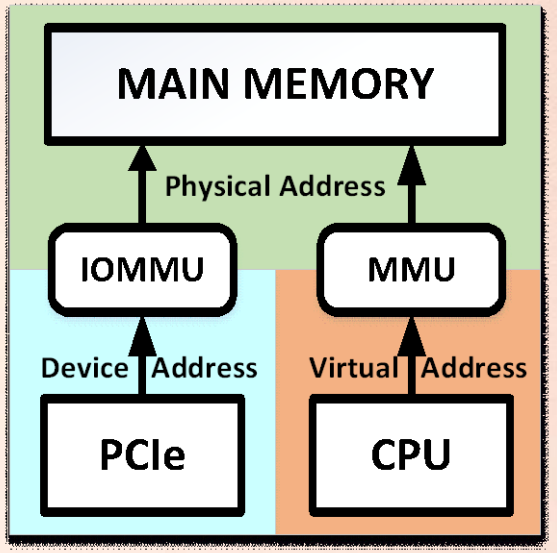
DMA – Direct Memory Access

PCIe devices accesses memory with DMA w/ physical (device) addresses

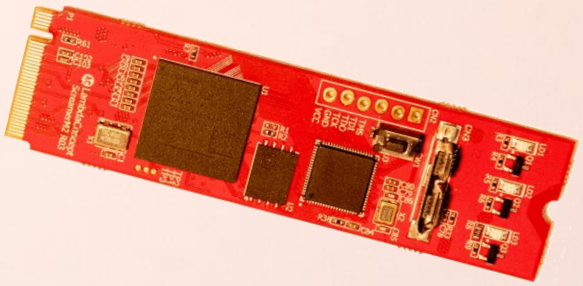
PCIe devices can access memory directly if the IOMMU is not used



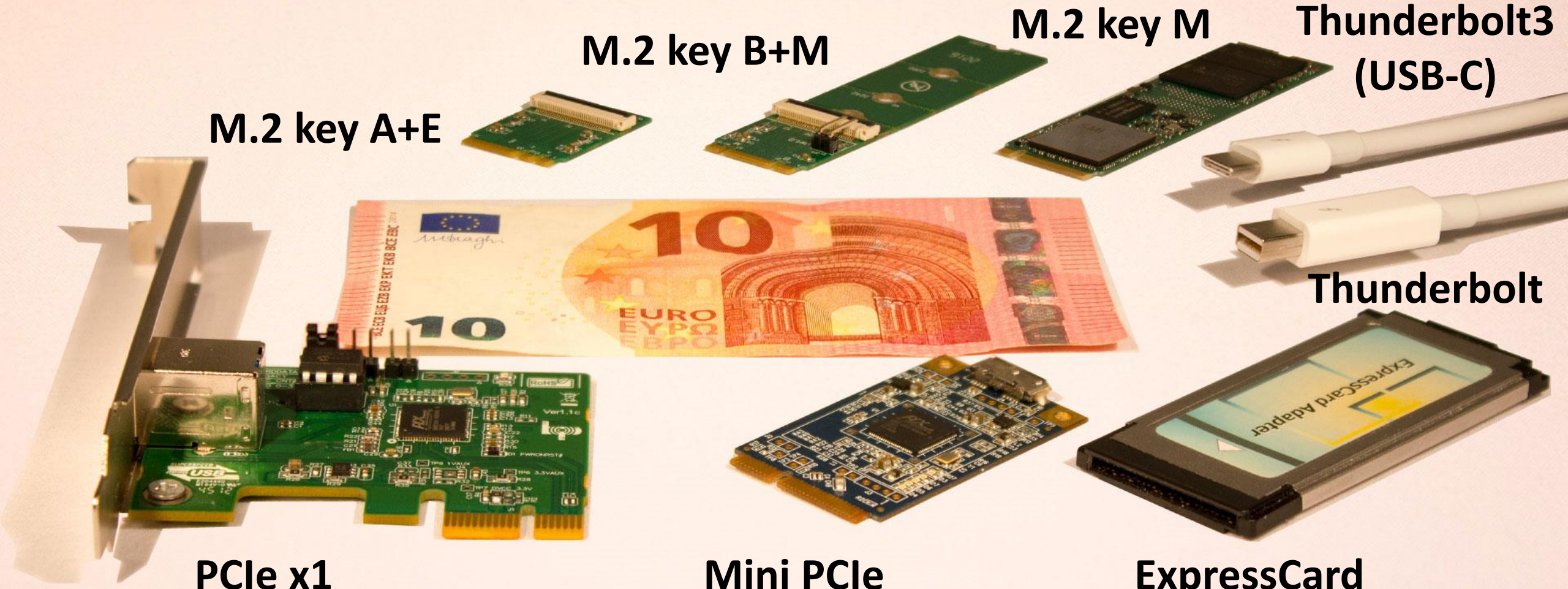
No VT-d (“normal”)



VT-d enabled

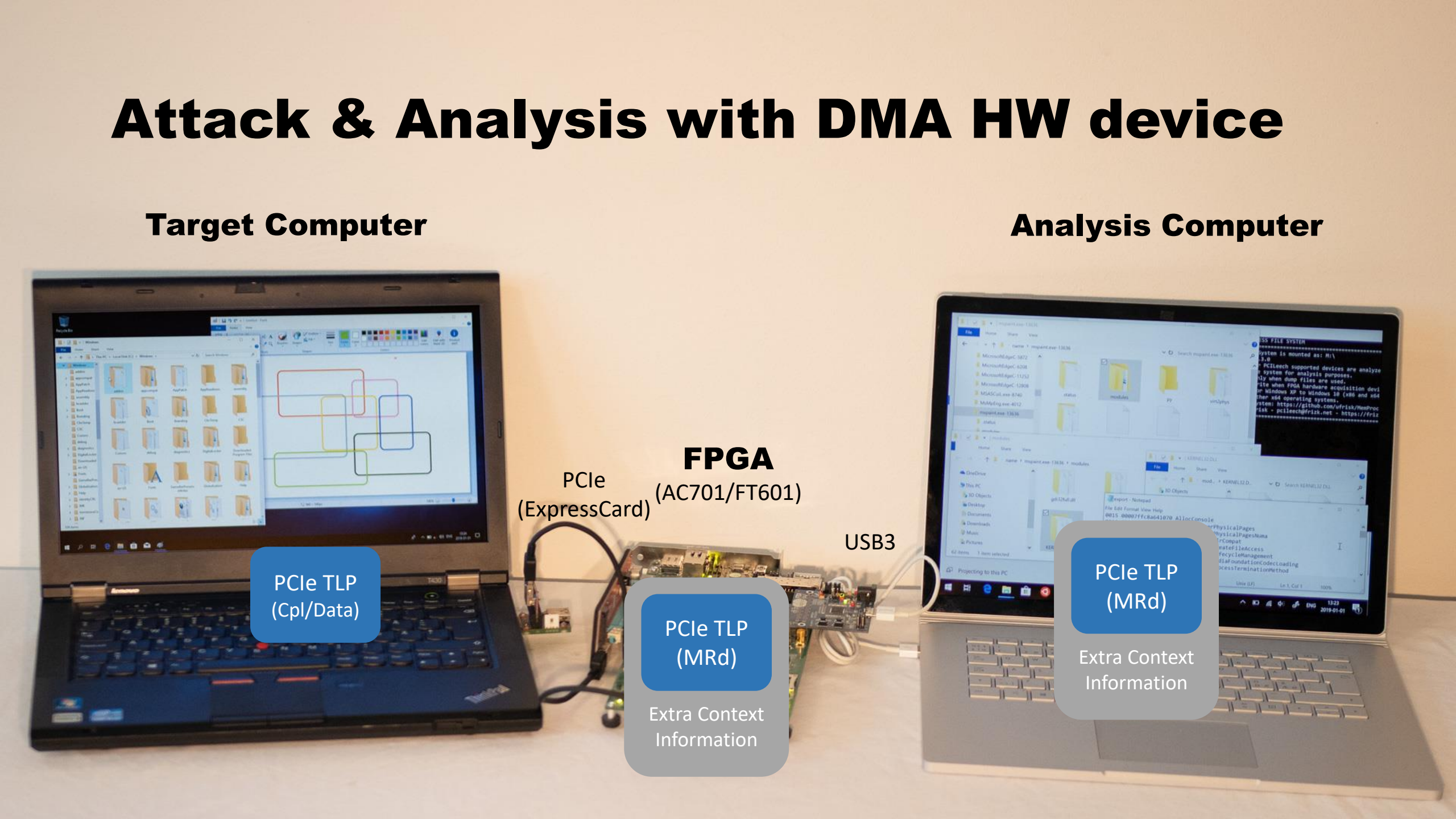


PCI Express Form Factors



Everything here is PCI Express in different form factors and variations.

Attack & Analysis with DMA HW device



PCILeech

DEMO: WINDOWS

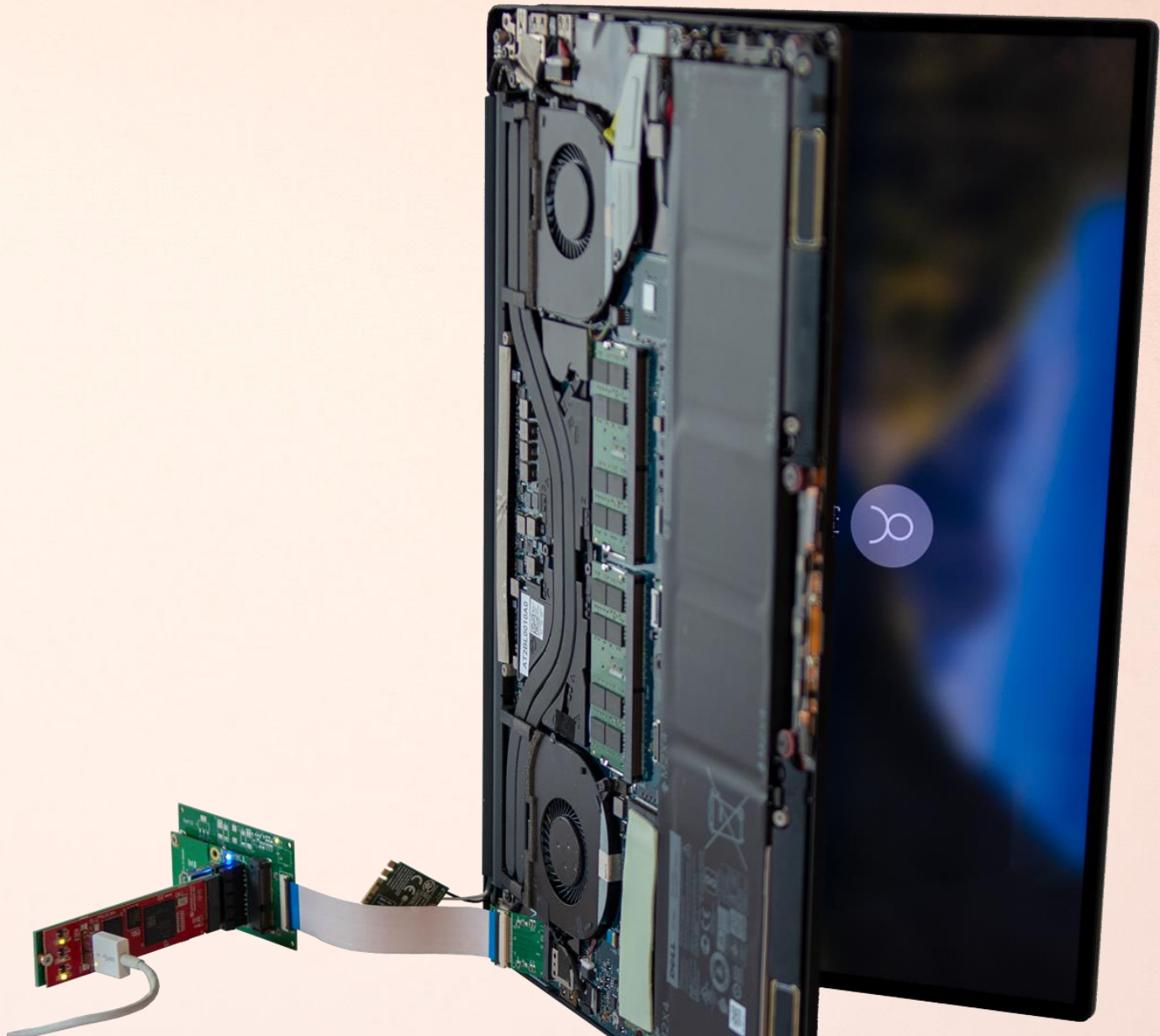
github.com/ufrisk/PCILeech

Windows DEMO:

Execute Code and
Spawn Shell

MOUNT:
target file system

Target System: Windows 10 1909



Windows Security:

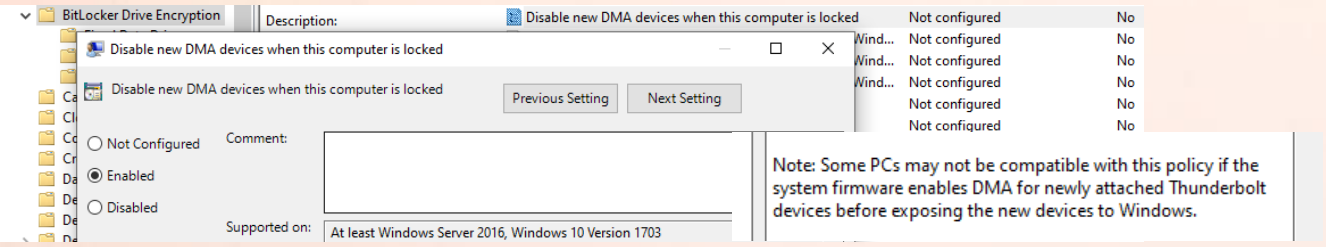
Win10 = (in)secure depending on hardware and config

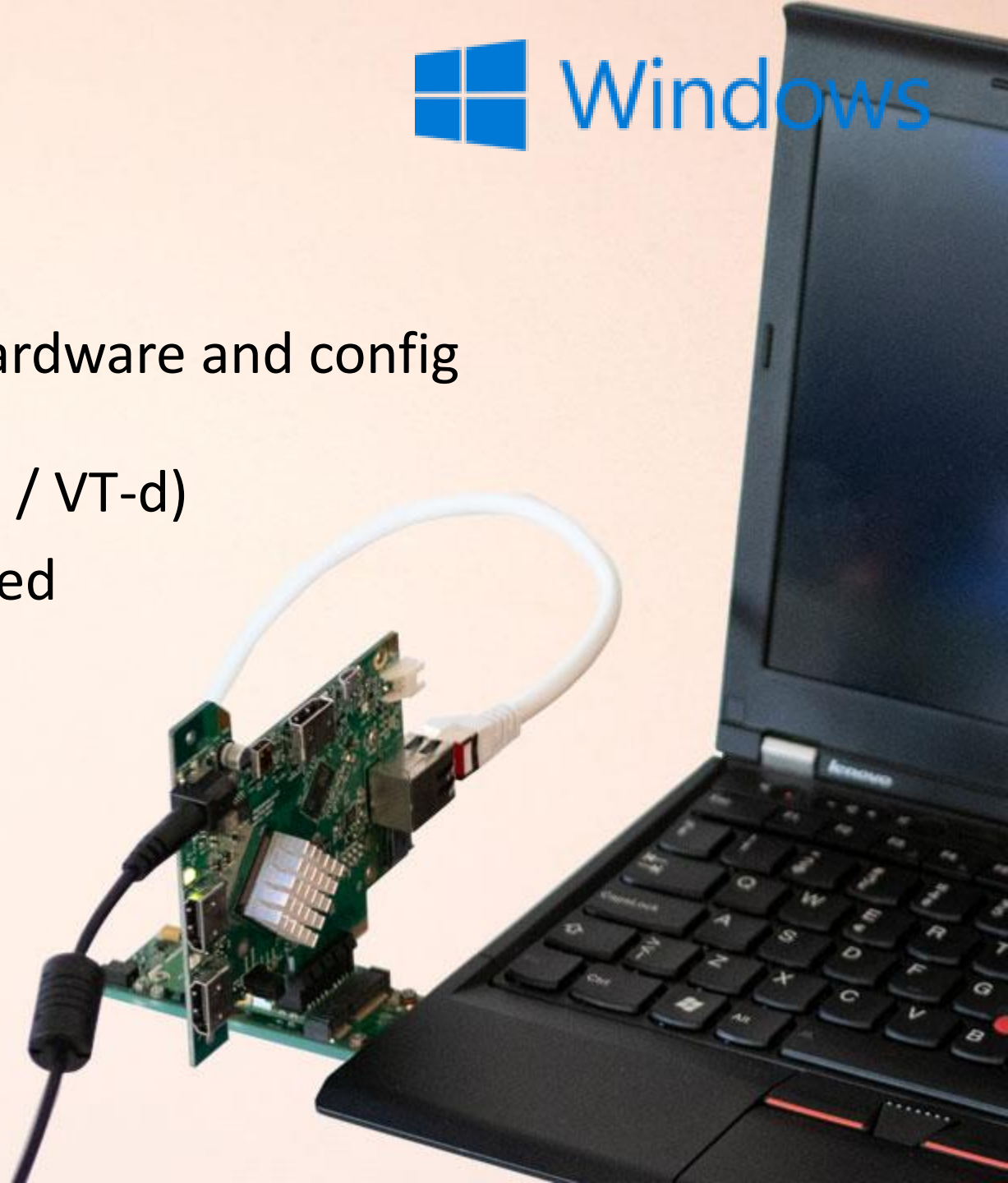
Virtualization based security (IOMMU / VT-d)

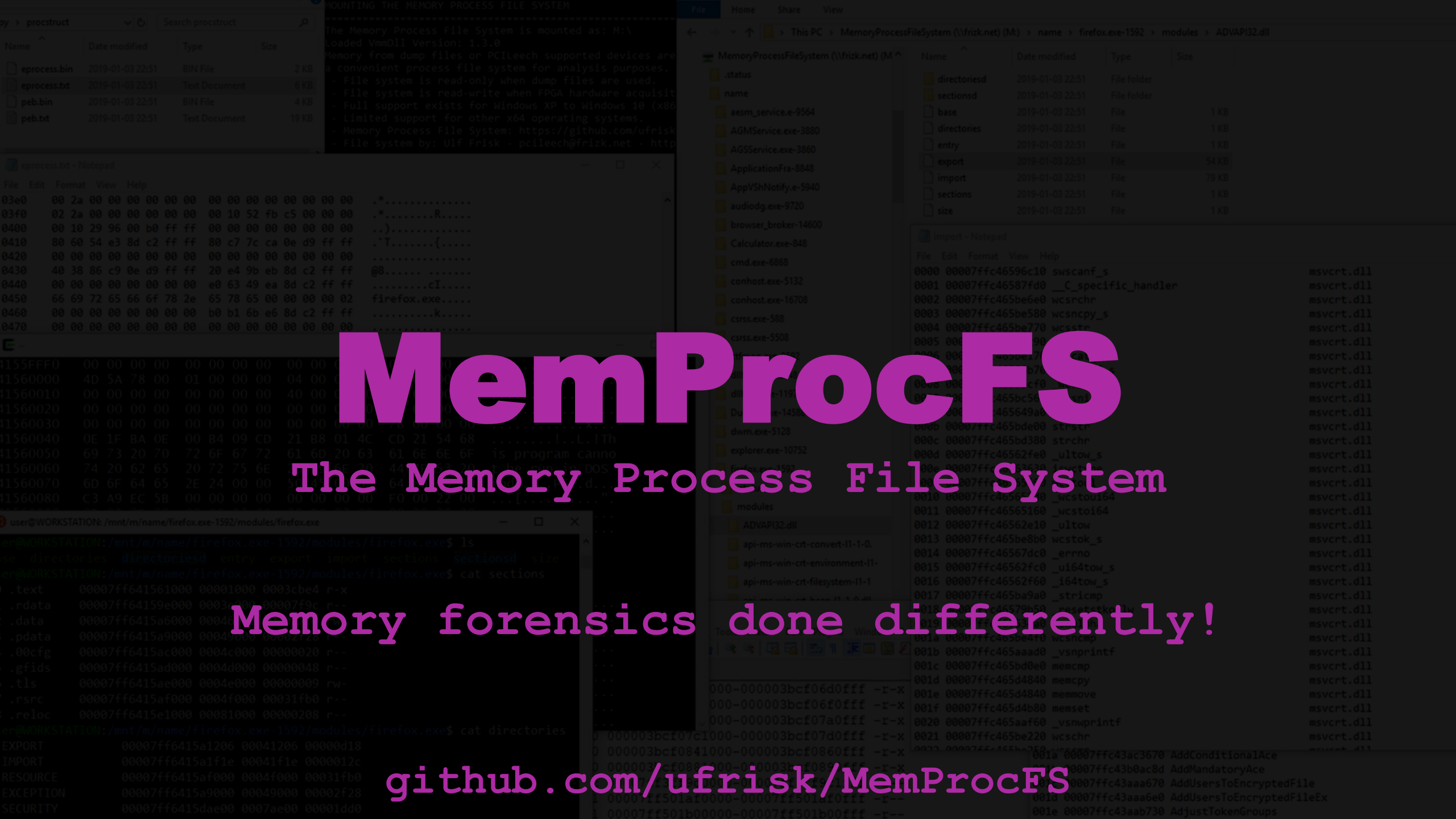
Disables new DMA devices when locked

HVCI, Secure Core PC

Bitlocker PIN






MemProcFS
The Memory Process File System
Memory forensics done differently!
github.com/ufrisk/MemProcFS

MemProcFS - The Memory Process File System

Windows Memory Analysis

In-Memory objects as **Files** and **Folders**

Multi-threading + C + intelligent parsing → **Super Fast!**

Hardware and **Software** memory acquisition

Why Memory Forensics?

Capture **current running state** of a system.

Processes, Network connections, …

Not everything exists on disk

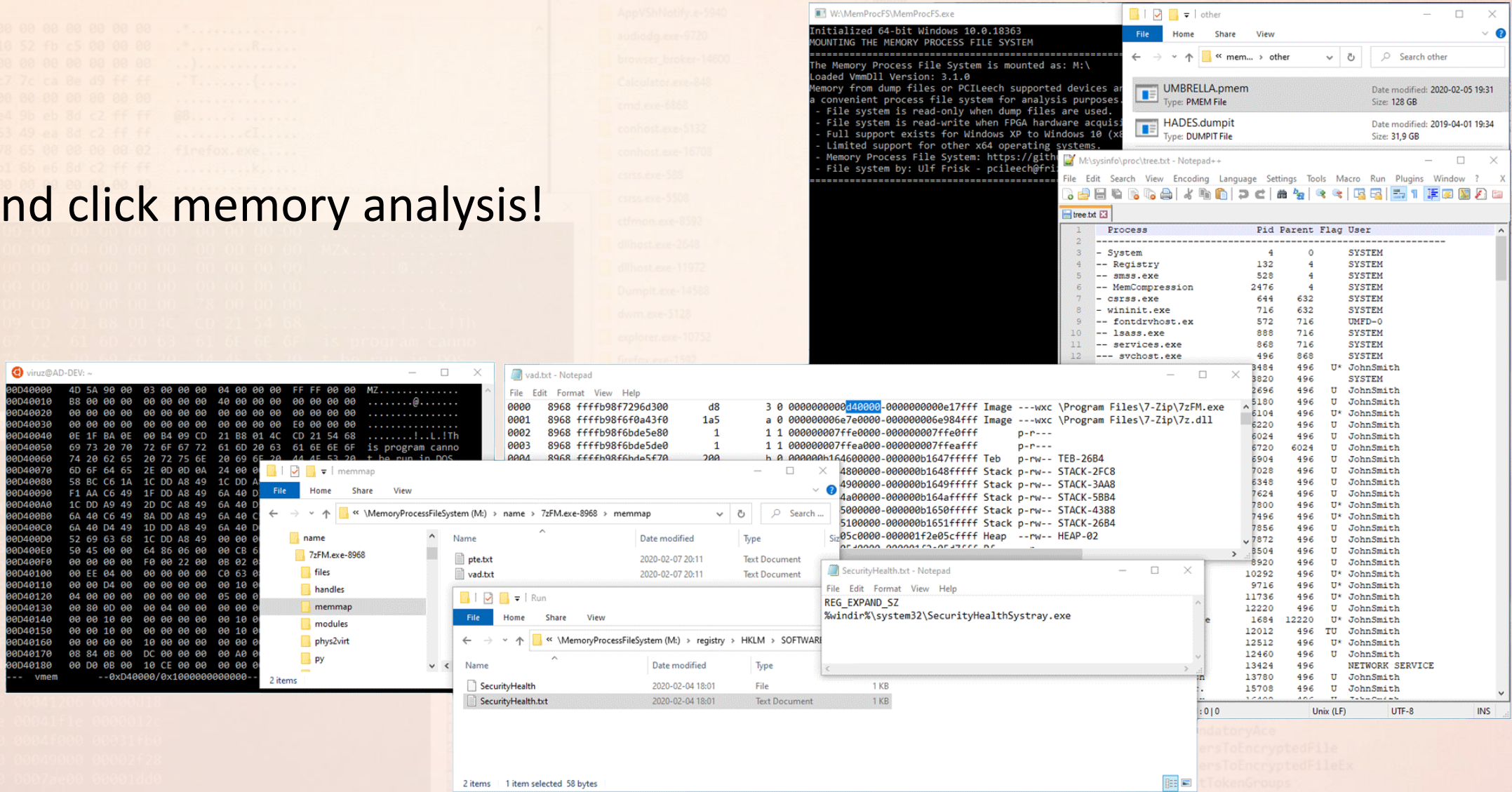
In-memory malware,

Crypto-keys (Bitlocker, …)

…

Demo: Mount 128 GB in seconds!

Point and click memory analysis!




MemProcFS
Design: How it's done!
github.com/ufrisk/MemProcFS

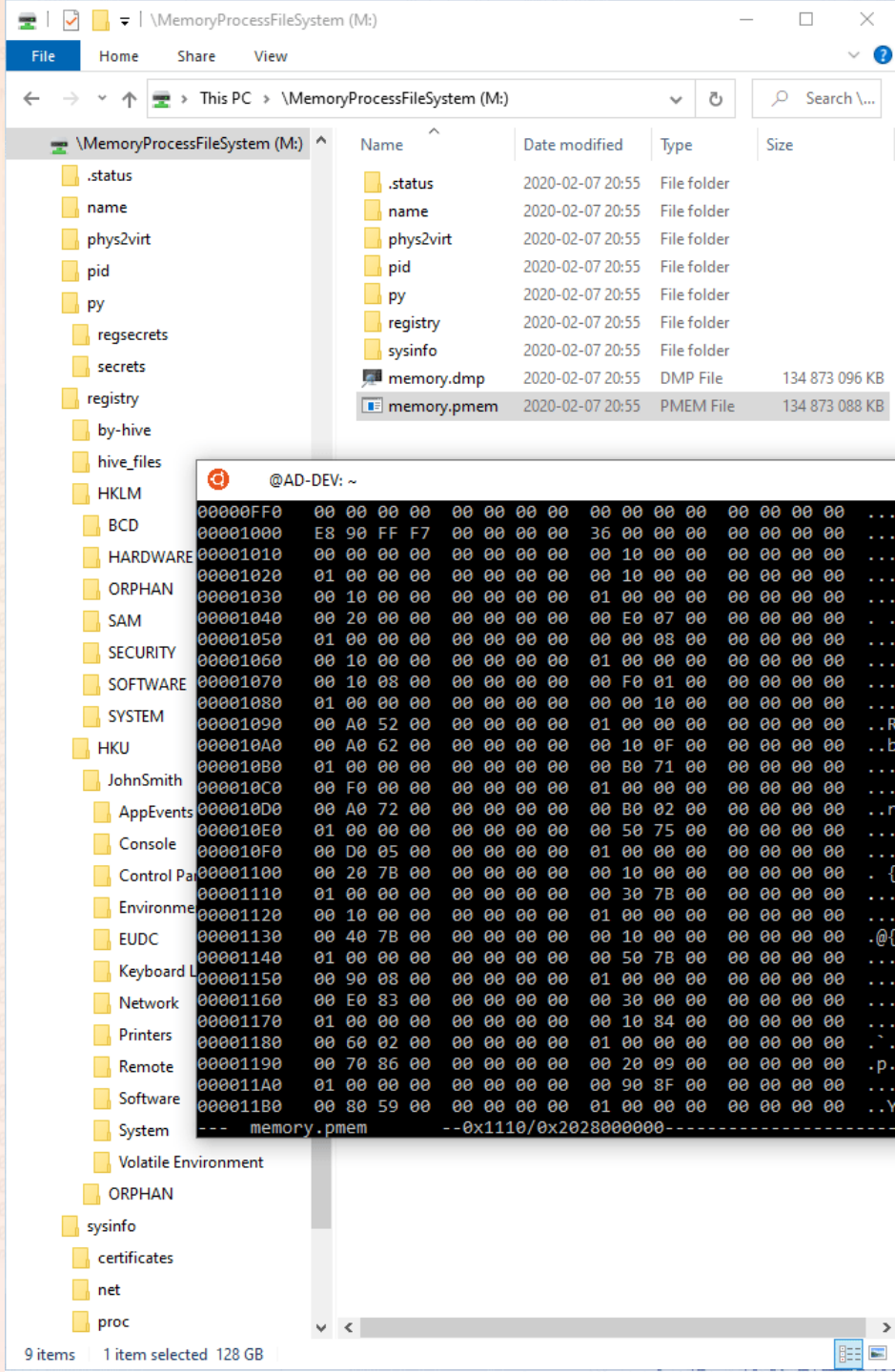
Design Goals

Ease of use – but yet powerful

Modular design and **Plugin** functionality

APIs – C and Python

Performance



Design Overview

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

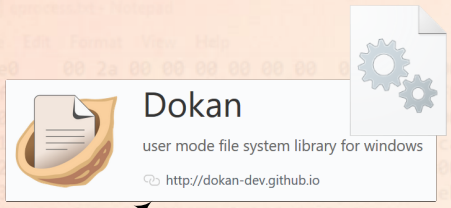
Remote

vmmpyc.pyd

Python API
(wrapper around C API)

Plugins
C and Python

The Vmm Library



File System callbacks
(very little code)

API

Analysis

Plugin Manager

Built-In Plugins

Processes

"Maps": Mem, Vads, Dlls, …

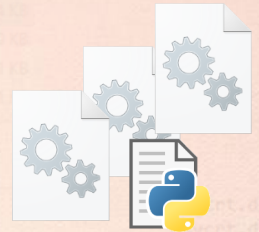
"Housekeeper" Thread

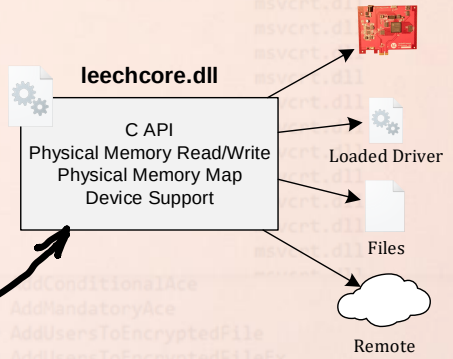
Virtual2Physical:
Memory Models – x64, PAE, x86

Cache:
Physical Memory

Cache:
Page Tables

object manager - refcount



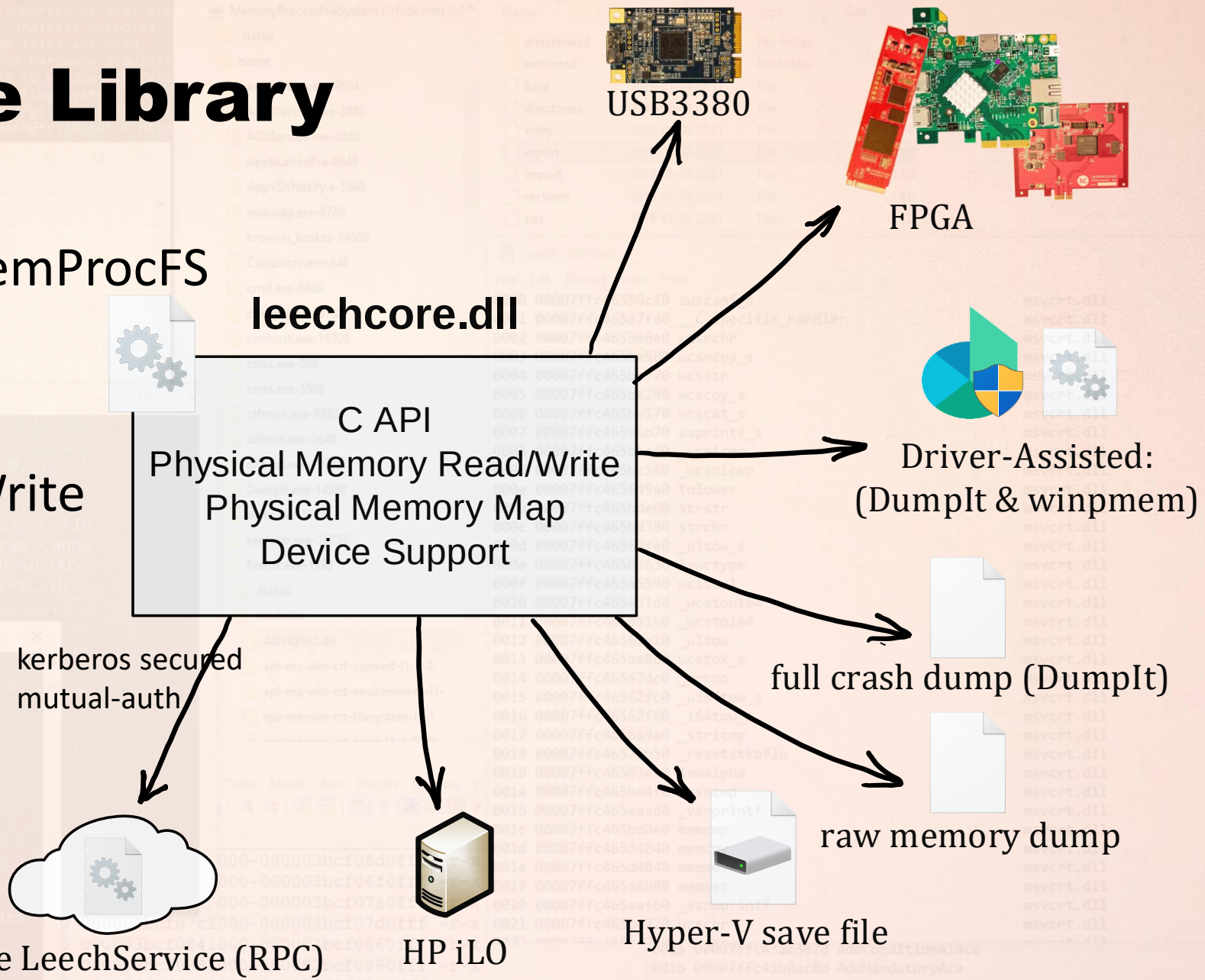


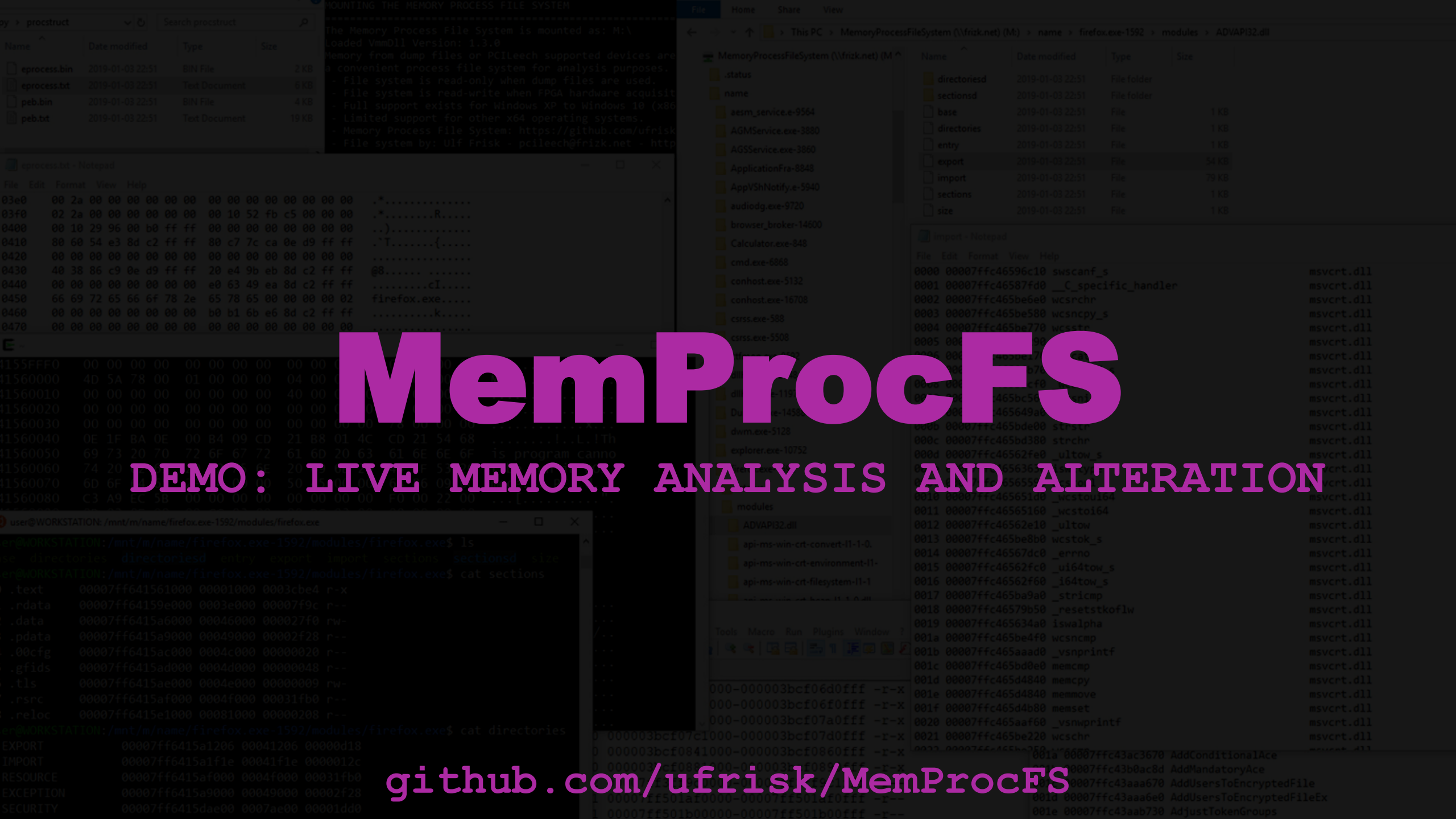
The LeechCore Library

Used by PCILeech and MemProcFS

Focus:
Physical Memory Read/Write

Separates memory acquisition from analysis

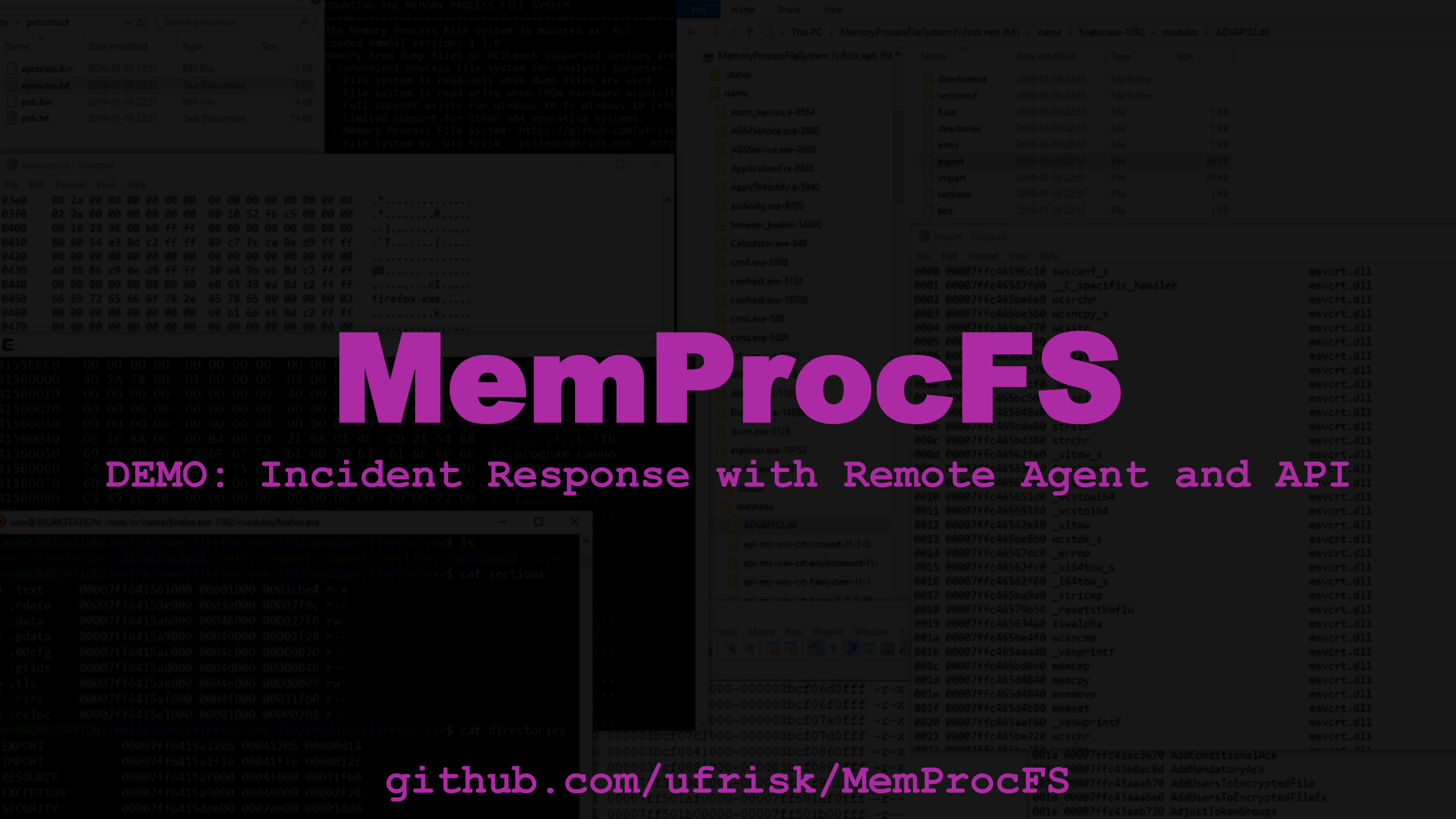



MemProcFS
DEMO: LIVE MEMORY ANALYSIS AND ALTERATION
github.com/ufrisk/MemProcFS

Demo: Analysis and write to Live Memory

Hunt for data in Process Heap

Overwrite data in memory


MemProcFS
DEMO: Incident Response with Remote Agent and API
github.com/ufrisk/MemProcFS

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 MemProcFS

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

Python API

Read/Write Physical/Virtual Memory

Process+Modules information

Registry+Debugging

List / Read / Write MemProcFS “files”

```
>>> VmmPy_
VmmPy_Close(                        VmmPy_ProcessListInforma
VmmPy_Refresh(                      VmmPy_ProcessGetEAT(
VmmPy_Initialize(                   VmmPy_ProcessGetIAT(
VmmPy_ConfigGet(                    VmmPy_ProcessGetDirector
VmmPy_ConfigSet(                    VmmPy_ProcessGetSections
VmmPy_GetVersion(                   VmmPy_WinReg_HiveList(
VmmPy_MemRead(                      VmmPy_WinReg_HiveRead(
VmmPy_MemReadScatter(               VmmPy_WinReg_HiveWrite(
VmmPy_MemWrite(                     VmmPy_WinReg_KeyList(
VmmPy_MemVirt2Phys(                 VmmPy_WinReg_ValueRead(
VmmPy_PidList(                      VmmPy_WinNet_Get(
VmmPy_PidGetFromName(               VmmPy_VfsList(
VmmPy_ProcessGetPteMap(             VmmPy_VfsRead(
VmmPy_ProcessGetMemoryMap(          VmmPy_VfsWrite(
VmmPy_ProcessGetVadMap(             VmmPy_PdbSymbolAddress(
VmmPy_ProcessGetHeapMap(            VmmPy_PdbTypeSize(
VmmPy_ProcessGetThreadMap(          VmmPy_PdbTypeChildOffset
VmmPy_ProcessGetHandleMap(          VmmPy_GetUsers(
VmmPy_ProcessGetModuleMap(          VmmPy_WinGetThunkInfoEAT
VmmPy_ProcessGetModuleFromName(     VmmPy_WinGetThunkInfoIAT
VmmPy_ProcessGetInformation(        VmmPy_UtilFillHexAscii(
```

Demo: Remote Malware Memory Analysis

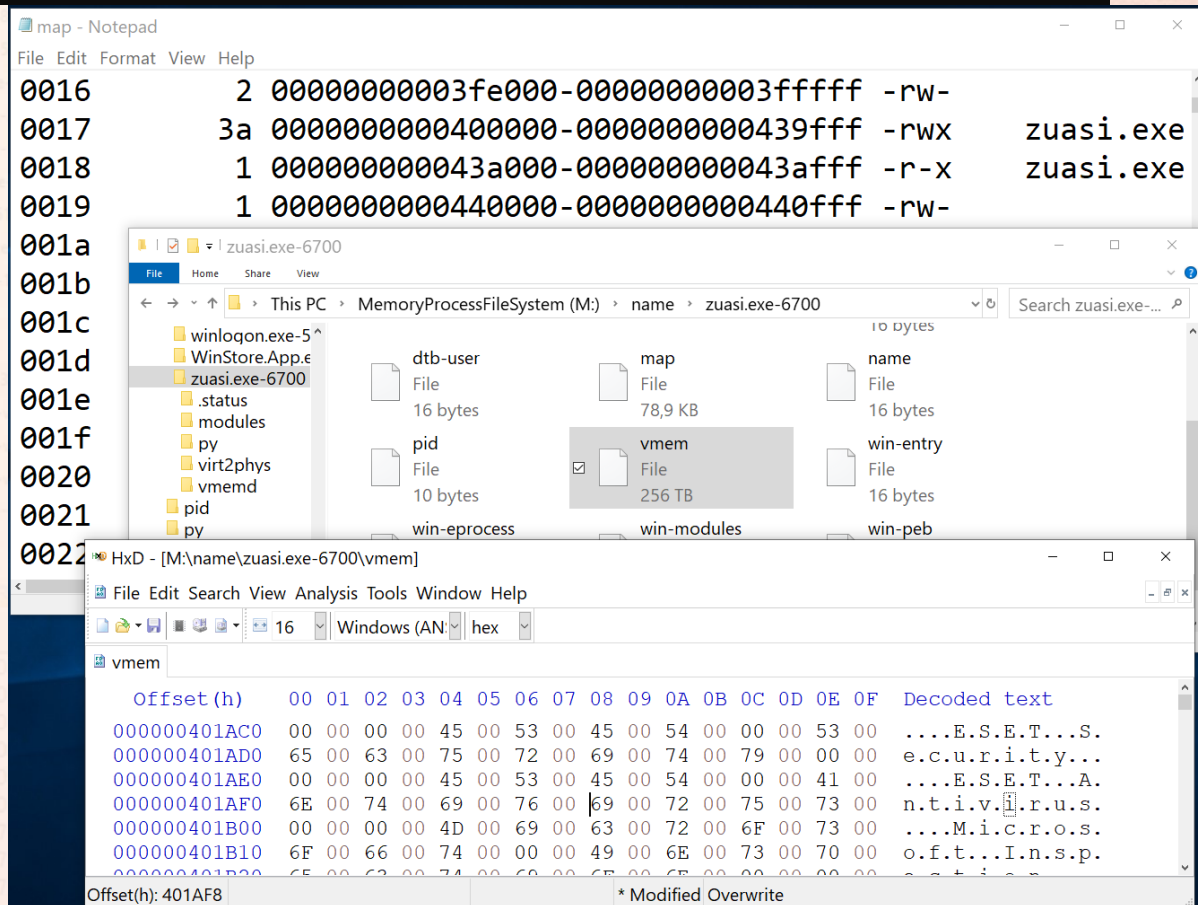
Command Prompt

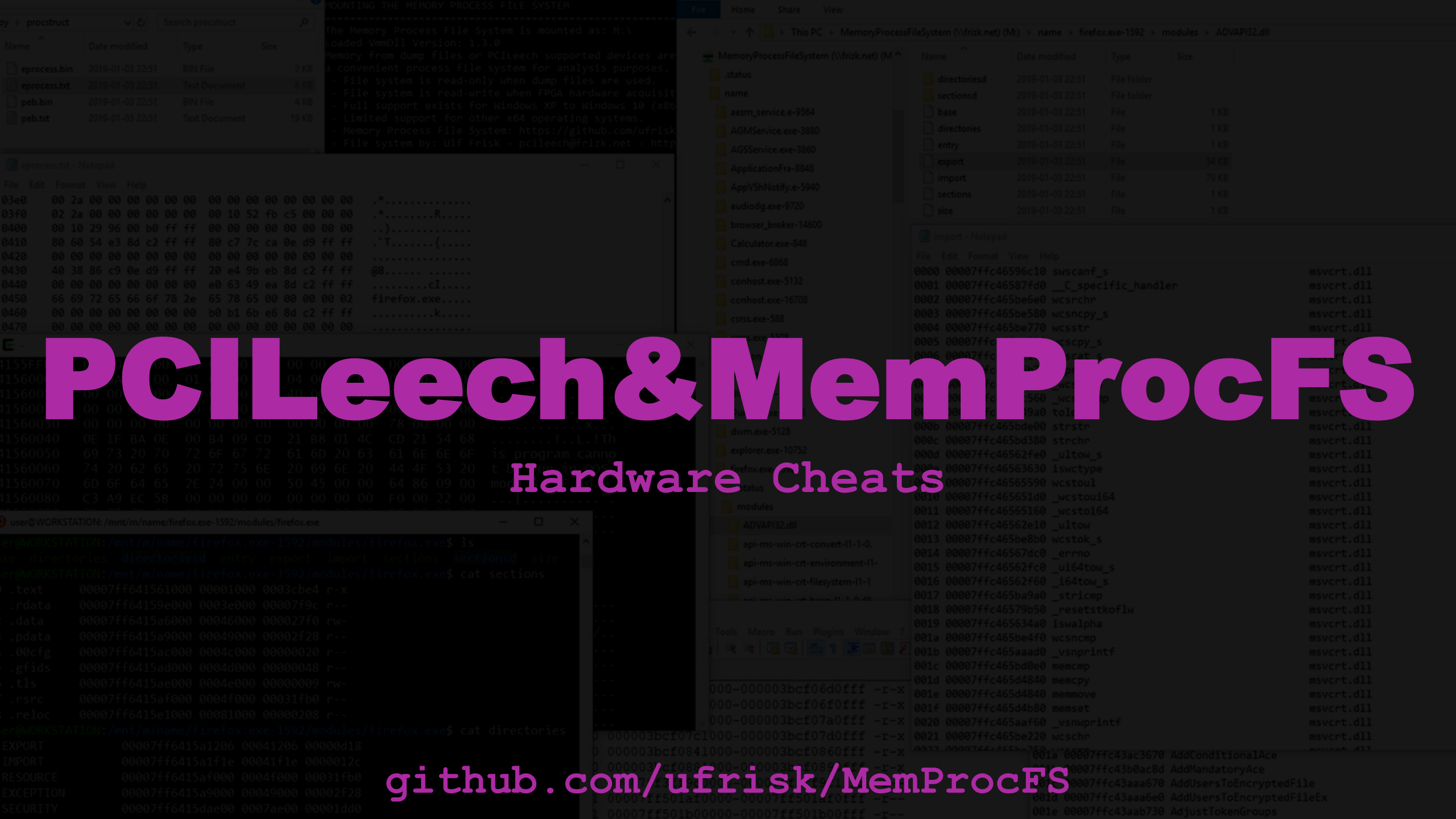
```
Q:\>MemProcFS.exe -device dumpit -remote rpc://kerberos-spn-remote-user:10.9.15.104
```

Analyze **live malware memory**

From a **remote** infected system

By clicking on files and using API




PCILeech&MemProcFS
Hardware Cheats
github.com/ufrisk/MemProcFS

Hardware Cheats with PCILeech/MemProcFS

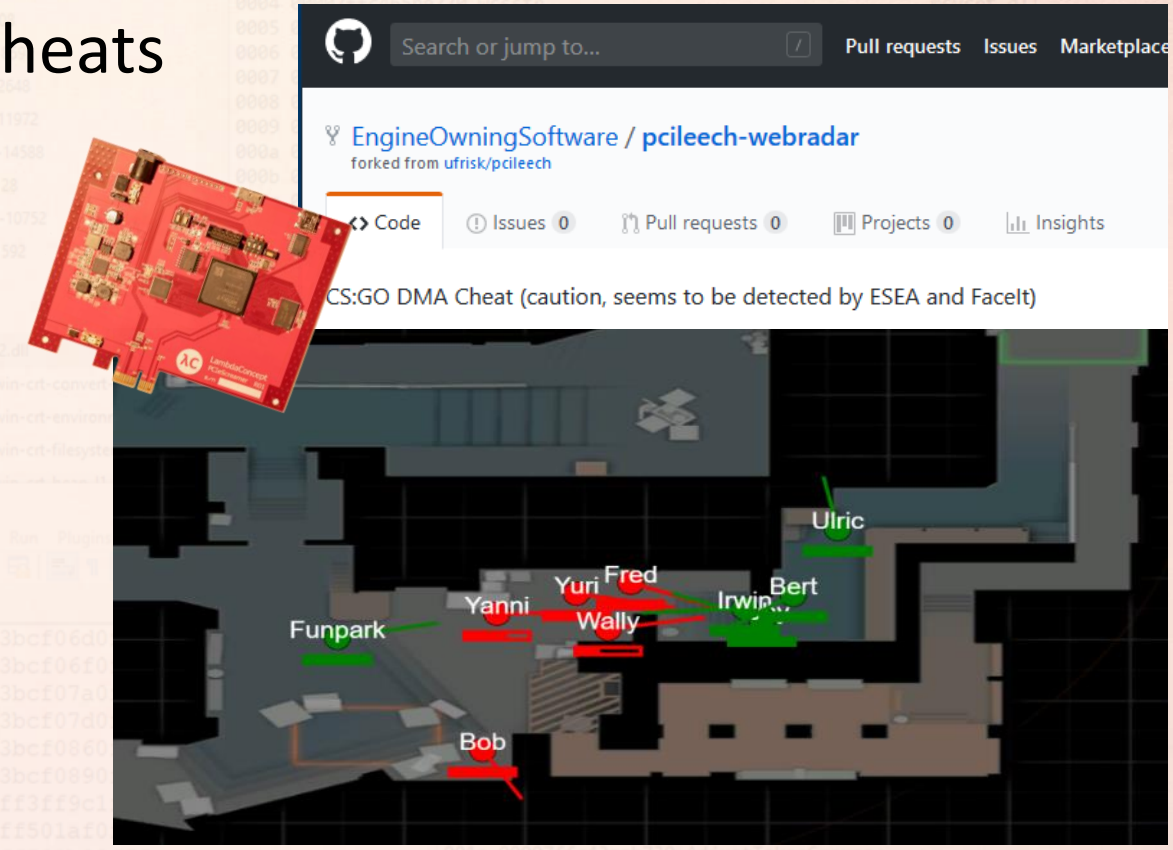
The **unexpected use case** – cheating in games!

Anti-Cheats – detects software based cheats

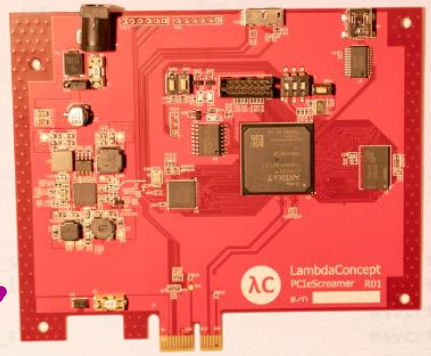
HW Cheat – ”only” a PCIe device ...

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

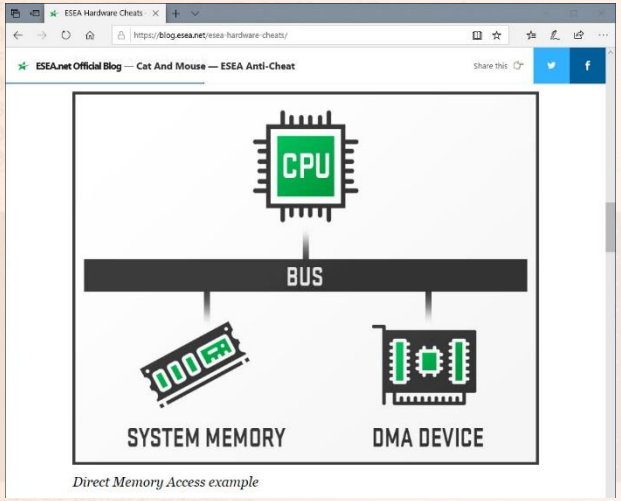
Now detected by anti-cheats.



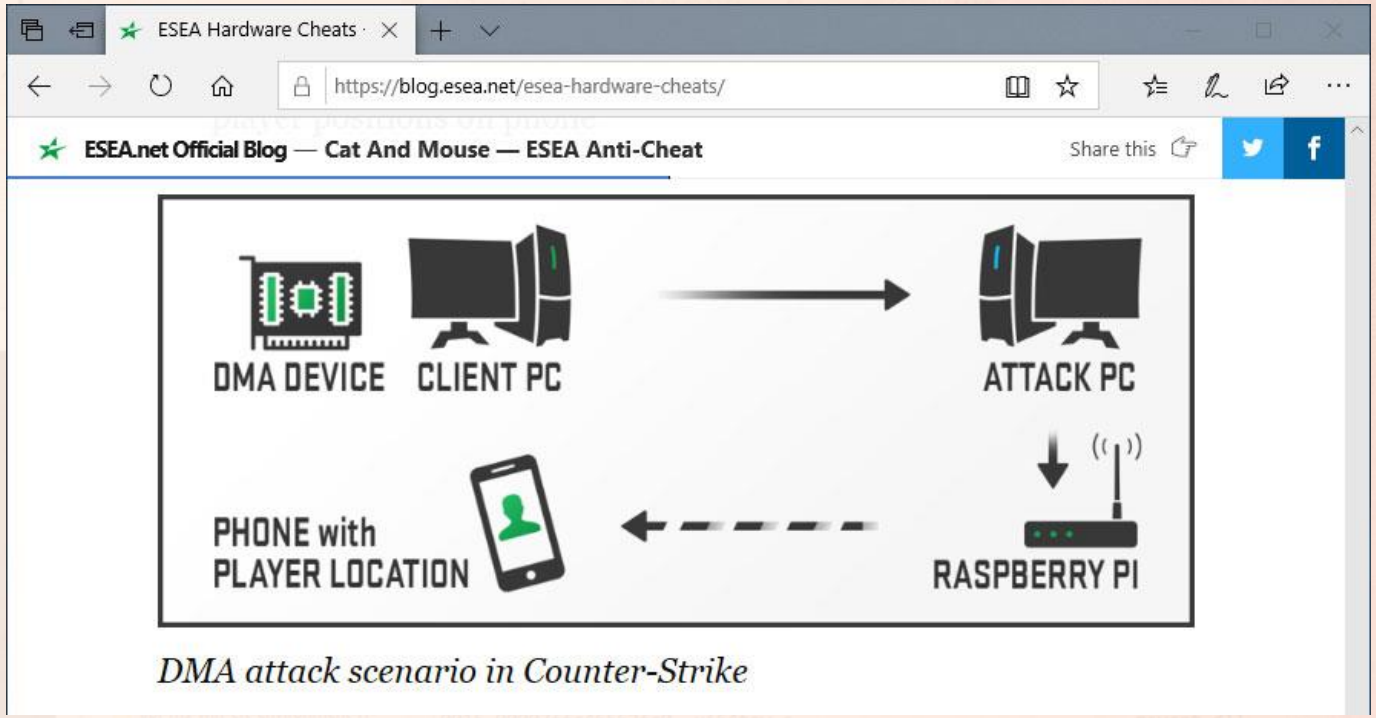
Hardware Cheats



Hardware Cheats



Direct Memory Access example



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. "


MemProcFS
Looking Forward
github.com/ufrisk/MemProcFS

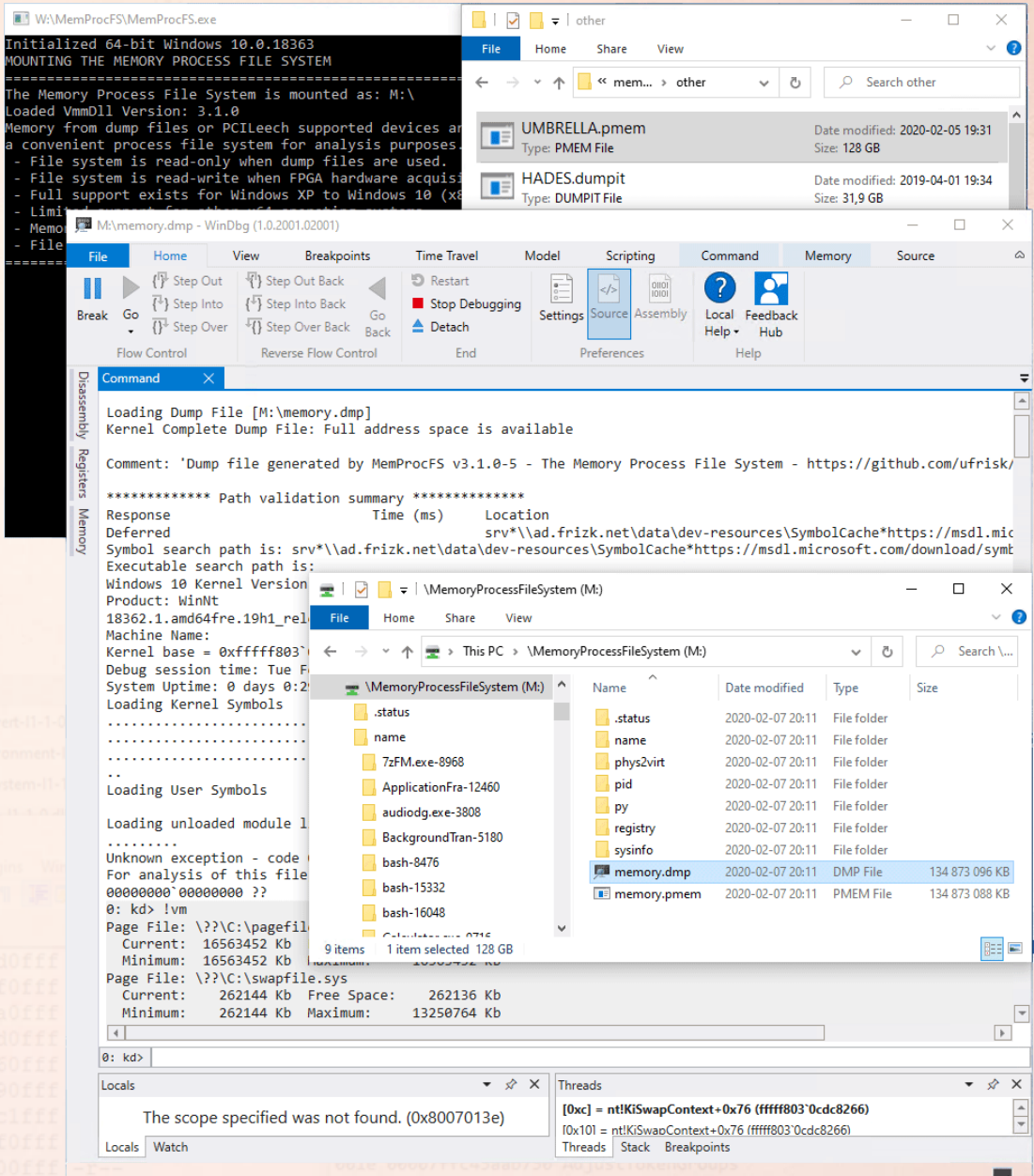
Looking Forward

More functionality:
- plugins …
- process minidump files for windbg
- better networking support
- …

Background scanning of memory dump files:
- page hashing → signature database lookups
- recovery of some free()'d artifacts

Timelining, MFT-files

APIs for Powershell and C#



MemProcFS and Volatility

MemProcFS may be a compliment – **Volatility is awesome** memory forensics!

Windows	Windows, Mac, Linux
Easy-to-use file system, C-API, Python-API	Command-Line, Python
Fast efficient in-memory objects parsing	Slower, more comprehensive, mem scanning
PCILeech live memory analysis&alteration, Windows10 memory compression	malfind, timelining, …

MemProcFS Summary

Fast!

Easy-to-Use – Point and click memory forensics!

Acquisition – memory dump files, driver, fpga, remote agent!

Plugins and **API** – roll your own plugins in **C/C++/Python**!

100% Open Source

github.com/ufrisk/MemProcFS

Thank You!

```
 Current Action: Dumping Memory
 Access Mode:    KMD (kernel module assisted DMA)
 Progress:       8678 / 8678 (100%)
 Speed:          173 MB/s
 Address:        0x000000021E600000
 Pages read:     2050967 / 2221568 (92%)
 Pages failed:   170601 (7%)
Memory Dump: Successful.
```

UlfFrisk

github.com/ufrisk