Forensics for System Administrators

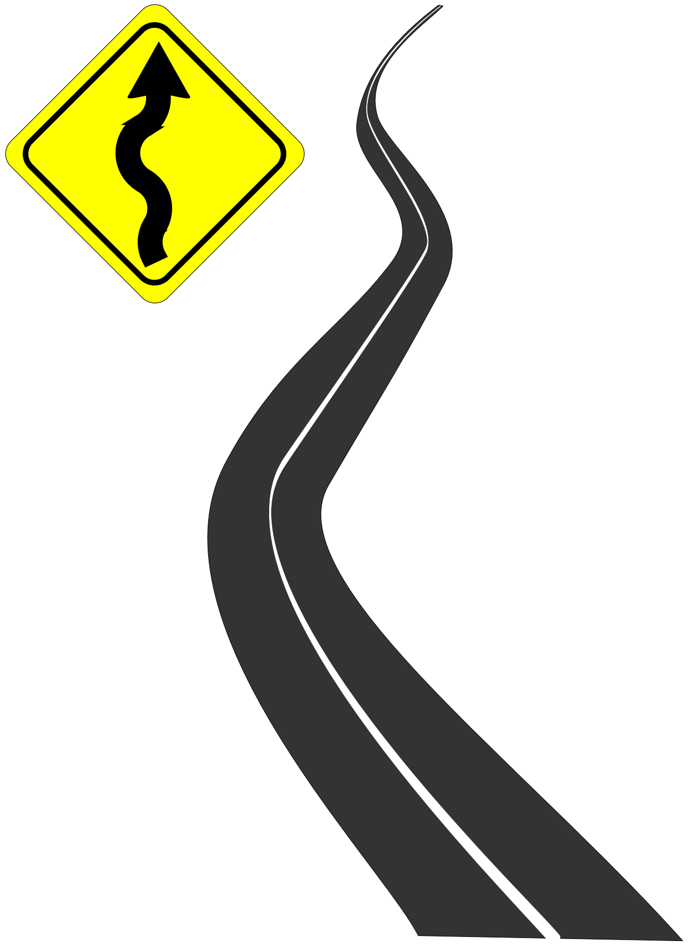
Memory Acquisition I

Klaus Möller
WP8-T1

Webinar, 9th of December 2021

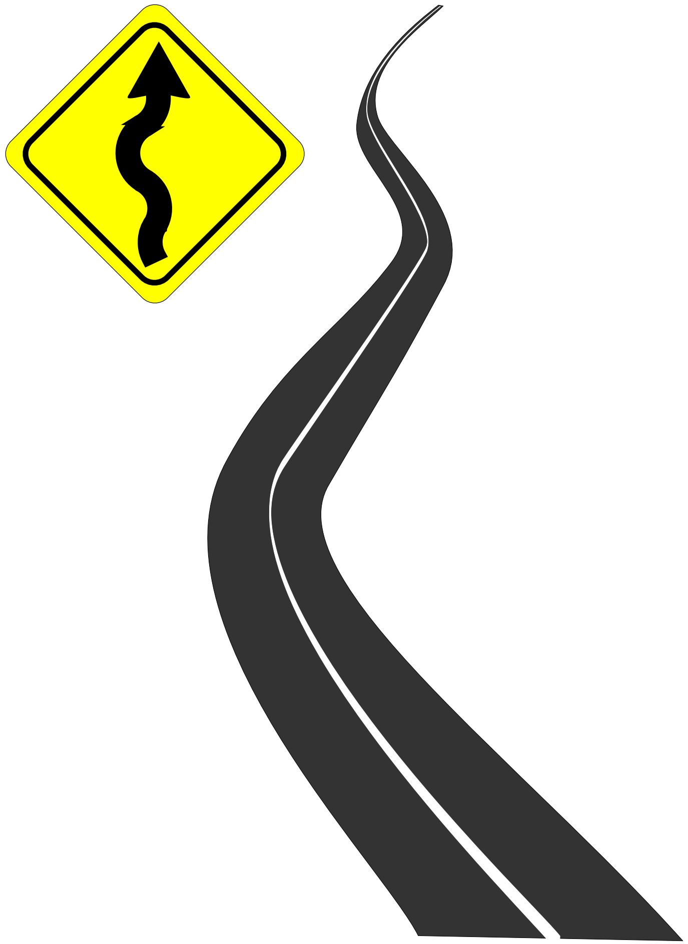
Public

Agenda - Part I



- Motivation
- Technical Basics
  - Virtual and Physical Memory
- Main Memory Dumps
  - Simple
  - Kernel Module
- System Crashdumps
  - Linux Kdump
  - Windows Crashdumps

Agenda - Part II



- Collection of Virtual Machine Memory
  - VMware
  - VirtualBox
  - Linux KVM/QEMU
- Swap & Hibernation
  - Linux Swap files/partitions
  - Windows pagefile, hibernation file
- Single Process Memory Dumps
  - Corefiles
  - Process Explorer

Motivation

Volatile System State - What are we looking for?

- Running processes
  - Path, command line arguments
  - Program code (executable, scripts)
  - Internal state (keys, passwords, kerberos tickets, etc.)
  - List of open files/sockets/network connections (w/ IP-addresses)
- Kernel
  - Version/executable, loaded modules/drivers
  - System call table, interrupt table, disk encryption keys, etc.
- Name caches: DNS, NIS, NetBIOS, …
- Currently logged in users
- Temporary filesystems (tmpfs)

Volatile System State - How to get it

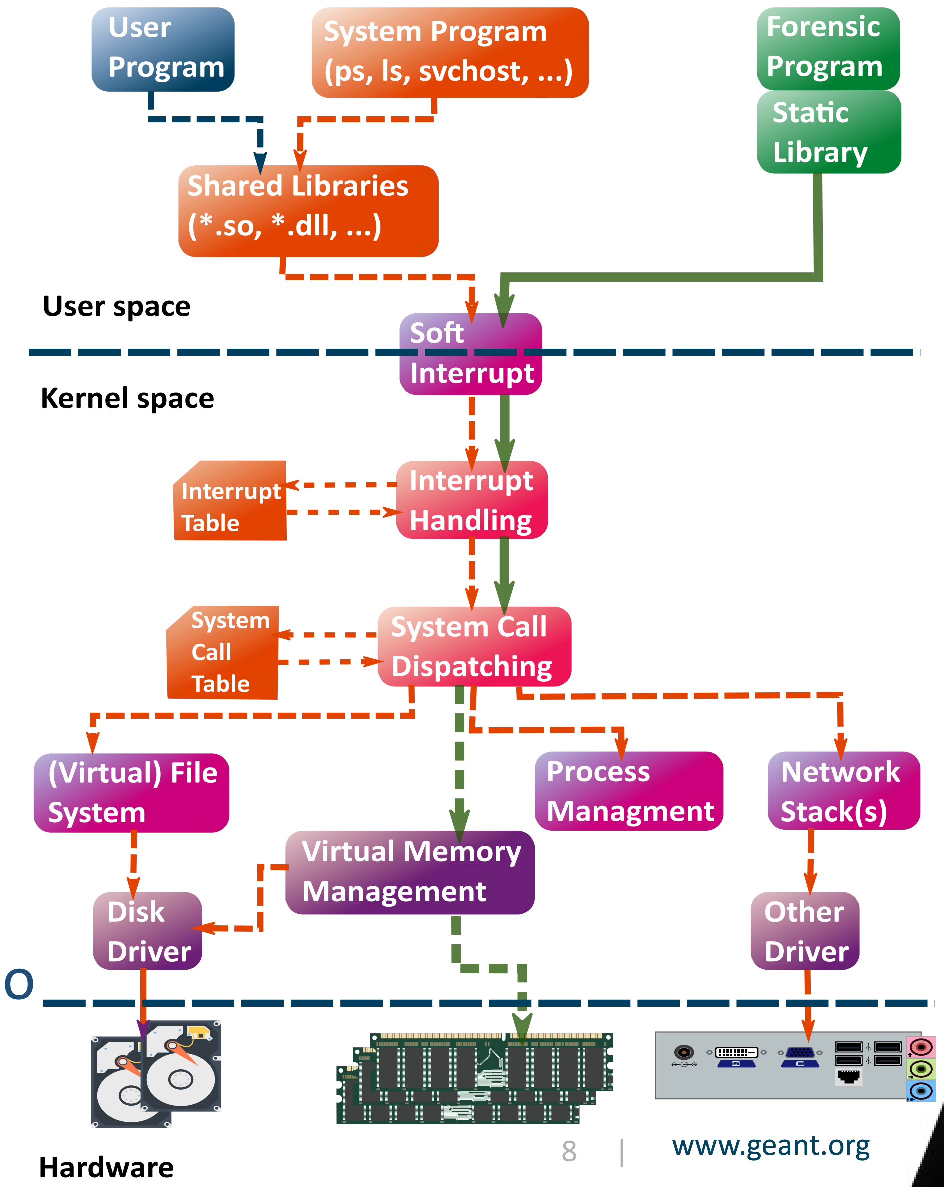
- Easy, isn't it?
  - Run **`ps, lsof, ss, lsmod, uname, date, uptime, …`**
  - And save the results ***somewhere***
- ***Somewhere?***
  - Not on the local disk or memory - that would change system state (more then necessary)
  - Better: Attached additional storage (e.g. USB-Stick)
  - Or save through the network to another machine
    - Use `netcat, cryptcat, socat, ssh`, etc.
- That's what some live response tools do

What about Rootkits?

- Their primary purpose is try to hide intruder presence/activity
  - Processes, files, network connections, etc.
- User space rootkits
  - Replacing system commands or shared libraries
  - Injecting malicious code directly into processes
- Kernel space rootkits
  - Manipulate Interrupt Table or Interrupt Handler code or System Call Table or System Call Code
  - Manipulation of kernel data structures
- What about *"as little trust as possible in a compromised system"*?
  - Point is, we cannot trust a compromised system

How to bypass Rootkits

- User space
  - Use tools from a trustworthy source
  - Put them on a CD/DVD or USB-Stick with hardware read-only switch
  - Statically linked libraries (or add clean libraries to medium)
- Kernel space
  - Bypass system-call chain as much as possible
  - Check the kernel-data structures carefully for manipulation
  - Not perfect, but the best we can do



DEUTSCHES FORSCHUNGSNETZ DFN DFN CERT

Solution

- Access OS data structures directly, bypassing syscalls

  → Kernel debugger

- Copy the memory contents and analyse them later on another system

  → Hardware, DMA through IOMMU
  - PCIe cards
  - Firewire, Thunderbolt, USB-4 interface

  → Software
  - Copying from `/dev/mem` or `\\.\Device\PhysicalMemory`
  - Crash dumps
  - Copying virtual machine (VM) memory from the Hypervisor
  - Swap/Hibernation partition/file

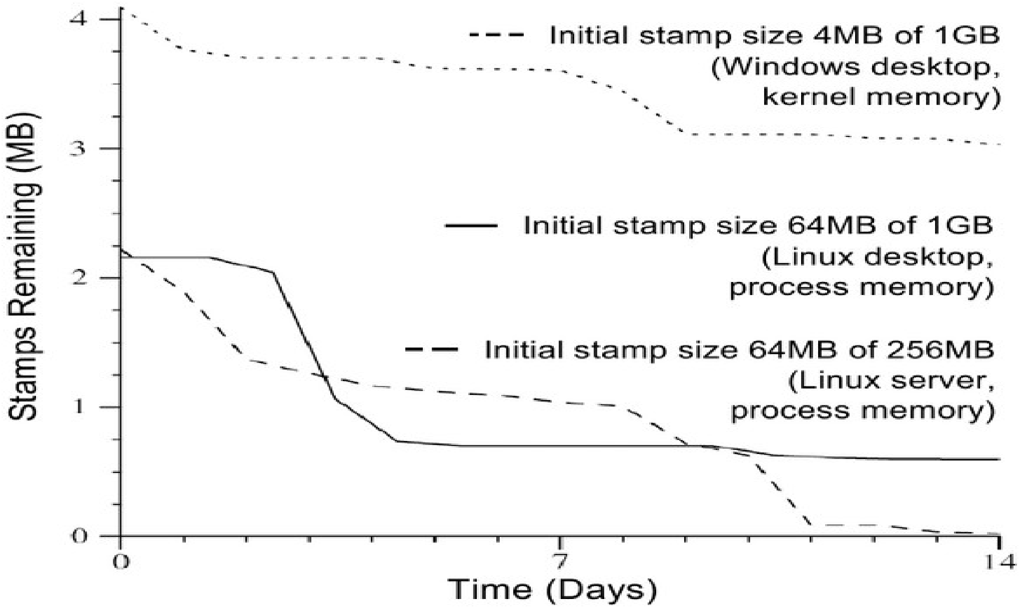
Computer Memory

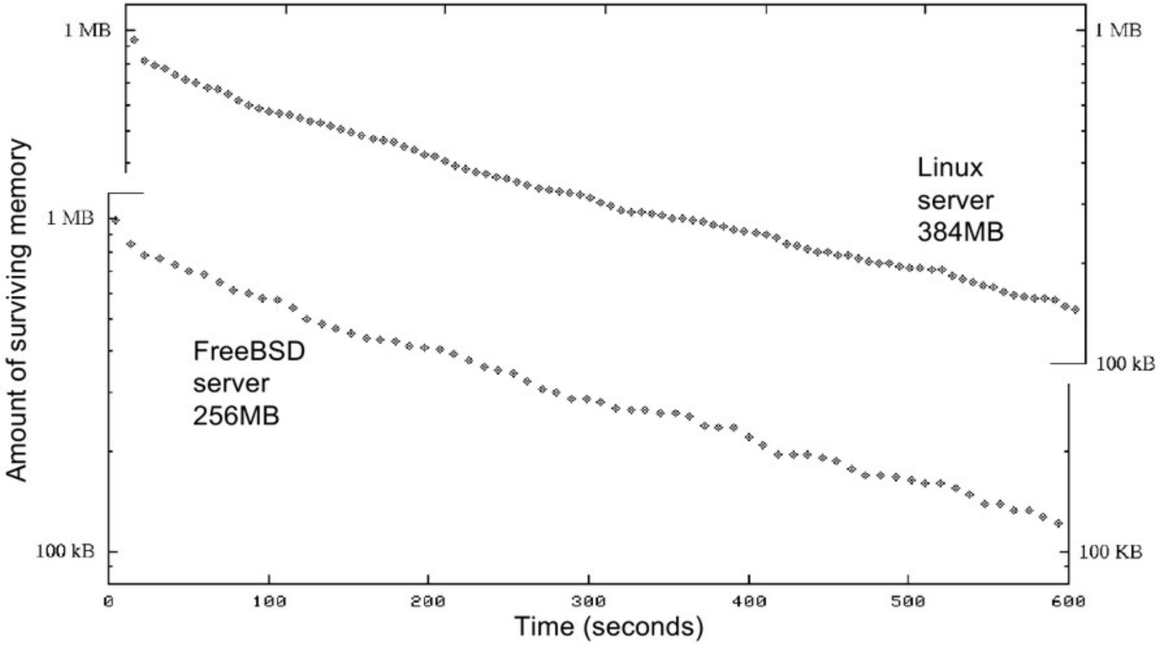
www.geant.org

Volatility of Traces in RAM

Memory contents of terminated processes

(Venema, 2005)



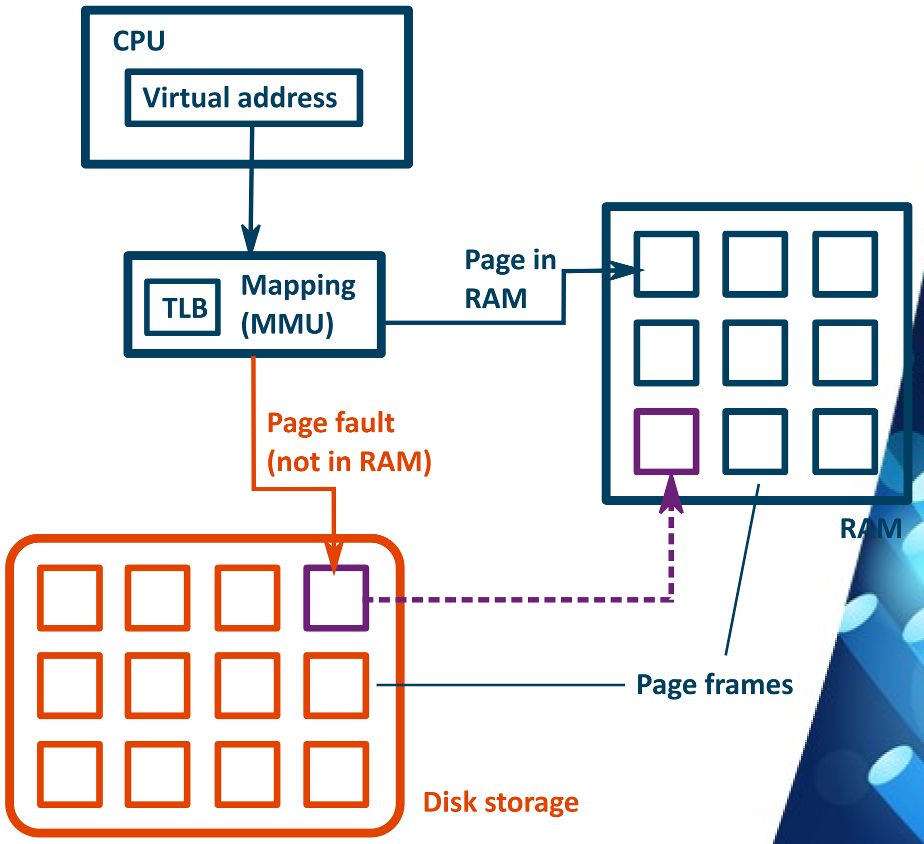


Memory contents of running processes

(Chow et. al, Usenix Security 2005)

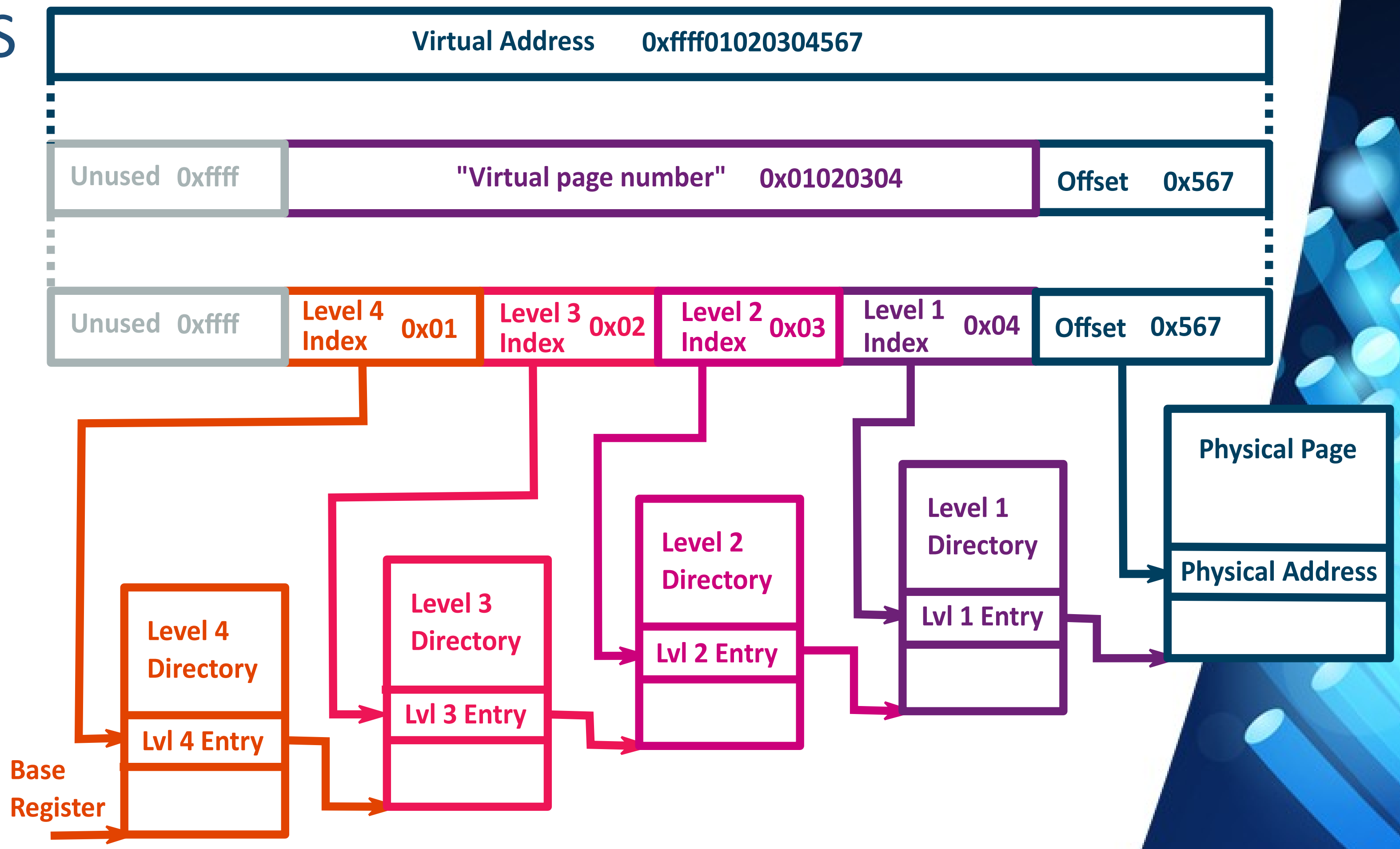
Virtual Memory (VM)

- Make it appear as if every process has the whole memory for itself
  - No need to care about other processes data, or the kernel
  - Each process (and the kernel too) has one large linear address space
  - Broken up into chunks, called pages
- Even better, make it appear, as if all of it is actually available
  - I.e. more memory than physically installed RAM
  - Everything not kept in RAM has a copy on disk
    - In the filesystem (executables, shared libraries, memory mapped files)
    - Or swap space (file or partition)



Virtual Memory Management (VMM)

- On each memory address access, the OS needs to
  - Translate from virtual addresses to physical addresses
  - Hardware support in form of the Memory Management Unit (MMU)
  - Translation Lookaside Buffer (TLB): Cache to speed up page table lookups



Impact on Memory Analysis

- When doing the analysis off-line
- Addresses (i.e. pointers) we see in the memory dump are ***virtual*** addresses
- The offsets into the memory dump file are ***physical*** addresses
- During analysis, we have to go back and forth between the two, i.e. we have to re-do the MMUs task
- Thankfully, the forensic tool takes care of this
- The page tables will always be present in RAM and thus in memory dump
- Otherwise the task would be undoable

Main Memory Dumps

www.geant.org

Memory Collection on Linux

- Basic approach
  - `dd if=`***`/dev/xxx`*** `| netcat` ***`target-host target-port`***
  - Fails after reading 1 Megabyte under Linux
- Newer (since 2003) Linux/Windows versions do not allow reading full kernel memory from user space

```
> grep DEVMEM /boot/config-$(uname -r)
CONFIG_DEVMEM=y                         # has /dev/mem
CONFIG_ARCH_HAS_DEVMEM_IS_ALLOWED=y
CONFIG_STRICT_DEVMEM=y                  # restrict access to PCI & BIOS
CONFIG_IO_STRICT_DEVMEM=y               # restrict to idle IO regions
```

- Need special driver (module) to access memory from kernel space

Memory Imaging Process

1. Preparation
    a) Build a profile for volatility or other tool (if needed)
    b) Compile the collection tool/kernel module
2. Collection
    a) To disk or over the network to a remote systems disk
3. Checking the image
    a) Testing the checksum

Profiles?

- Without additional information, ...
  - We would have no idea what kind of data is at a given address
    - Integer, float, string, structure, …
  - Or what it is used for
    - Process, socket, file, directory, etc.
- What’s needed is the symbol table from the compiler
  - Can be used directly for debuggers
- Some forensic tools build more abstract, condensed structures from it
  - Volatility terminology: Profile

Linux: Building a Volatility (2.x) Profile

Live Demo

1. Determine kernel version

```
> uname -r
5.3.18-lp152.47-default
```

2. Clone repository

```
> git clone https://github.com/volatilityfoundation/volatility.git
```

3. Compile

```
> cd volatility/tools/linux/
> make
```

4. Pack

```
> zip newprofile.zip module.dwarf /boot/System.map-$(uname -r)
```

Linux: Building a Volatility 3 Profile

Live Demo

1. Clone repository

```
> git clone https://github.com/volatilityfoundation/dwarf2json.git
```

2. Compile

```
> cd dwarf2json
> go build
```

3. Generate profile (Linux & Mac OS X only)

```
> dwarf2json linux --system-map /boot/System.map -$(uname -r) \
  $(uname -r).json
```

Linux: Compiling the Kernel Module

Live Demo

1. Clone LiME repository

```
> git clone https://github.com/504ensicsLabs/LiME/
```

2. Compile

```
> cd LiME/src
> make clean
> make
```

Linux: Collecting the Memory (to disk)

Live Demo

- Raw image

```
# insmod lime.ko "path=/tmp/testdump.raw format=raw"
```

- Image in LiME format

```
# insmod lime.ko "path=/tmp/testdump.raw format=lime"
```

- Compressed image

```
# insmod lime.ko "path=/tmp/testdump.raw format=lime compress=1"
```

- Everything together (with checksum)

```
# insmod lime.ko "path=/tmp/testdump.raw format=lime compress=1
digest= sha512"
```

Remember to not write to local disk,
use another medium or the network!

Linux: Collecting the Memory (over the network)

Live Demo

- With **netcat**
  - On the compromised host

```
# insmod lime.ko "path=tcp:12345 format=lime localhostonly=0"
```

  - On the host taking the image

```
> netcat compromised-host 12345 > dumpfile
```

- With **ssh** & **netcat**
  - From the host taking the image (2nd line on the compromised host)

```
> ssh -L 12345:localhost:<target port> <compromised host>
# insmod lime.ko "path=tcp:12345 format=lime"
```

  - On the host taking the image

```
> netcat localhost 12345 > dumpfile
```

Checking the image

- Cryptographic hash sums are used to assert the chain of custody
  - I.e. that the image has not been tampered with (since acquisition)
- Technically
  - Use the build-in hash sum features of the collection tool
    - Faster, one less thing to forget
  - Do not use broken hash algorithms like MD5 or SHA-1
    - SHA256 is OK, SHA512 is better
- Organisationally
  - 4 eyes principle while collecting the memory
  - Store & transfer the checksum apart from the image
    - Or tampering becomes trivial
  - Even better: Cryptographic signatures, PGP or S/MIME, your choice

When Checking the Hash Sum ...

- In combination with compression
  - Using the **build-in checksum** feature, the checksum is that of the **uncompressed image** (i.e. before compression)

```
> sha512sum /tmp/testdump.lime; cat /tmp/testdump.lime.sha512
d4a0047f88fecc5336fb097670ec9ec3cc4...
19e625b5f013443785af58fa224cfa3a9a3...
```

  - Using **external tools,** the checksum is that of the **compressed image** (i.e. after compression)

```
> file /tmp/testdump.lime.sha512
/tmp/testdump.lime: zlib compressed data
> unpigz -c /tmp/testdump.lime | sha512sum; cat /tmp/testdump.lime.sha512
19e625b5f013443785af58fa224cfa3a9a3 … 7d6bff60b5bf0 -
19e625b5f013443785af58fa224cfa3a9a3 … 7d6bff60b5bf0
```

Windows: Collecting Memory & Checksum

- Take the image

```
winpmem_mini_x64_rc2.exe testdump.raw
```

- Take the checksum
  - With **certutil** (Windows build-in tool)

```
certutil -hashfile testdump.raw SHA512
```

  - With PowerShell

```
> Get-FileHash -Path y:\testdump.raw -Algorithm SHA512
```

Crashdumps

Kernel Debugger

- Several facilities for debugging errors in the kernel
  - Error message printing (`printk`) , tracing frameworks (e. g. `dtrace`), debuggers
- Live kernel debugging = Analysis of a running system through an attached debugger
  - Usually through the serial console (JTAG for embedded systems)
  - Network consoles are appearing (Linux kgdboe)
    - Linux: kdb and kgdb
    - Windows: KD, WinDbg
- Post mortem debugging through ***crash dumps***
  - Can also be imported into forensic tools
    - E. g. volatility

Crash Dumps

- Advantages
  - Dump file can be analysed with debuggers
  - Memory state does not change while dump takes place
  - Works with practically every operating system
- Disadvantages
  - Requires preparation of the OS, i.e. crash dump configuration
    - May need to be rebooted for configuration to take effect
  - Triggering a crash dump often will trigger a (subsequent) reboot
- Live dumps (or Live debugging) will usually not trigger reboots

Linux Crash Dump Preparation

- Install **kdump** and **kexec** packages - distribution dependant
- Kernel needs several options enabled
  - `CONFIG_KEXEC=y`
  - `CONFIG_CRASH_DUMP=y`
  - `CONFIG_PROC_VMCORE=y`
  - `CONFIG_SYSFS=y`
- Kernel needs to be booted with **`crashkernel=xxxM`** option
  - **`xxxM`** number of megabytes reserved for crash kernel (64 - 256 usually)
- Configuration files **`/etc/sysconfig/kdump`** and/or **`/etc/kdump.conf`**
- Enable **`kdump.service`** (systemctl)

Linux Crash Dump Execution

- Kernel gets signal to crash and hands over control to the crash kernel via kexec mechanism
- Crash kernel then does the actual dumping of the kernel
- Trigger as root (uid == euid!)

```
echo 1 > /proc/sys/kernel/sysrq
echo c > /proc/sysrq-trigger
```

- Dump file can be written over the network (SSH or NFS)

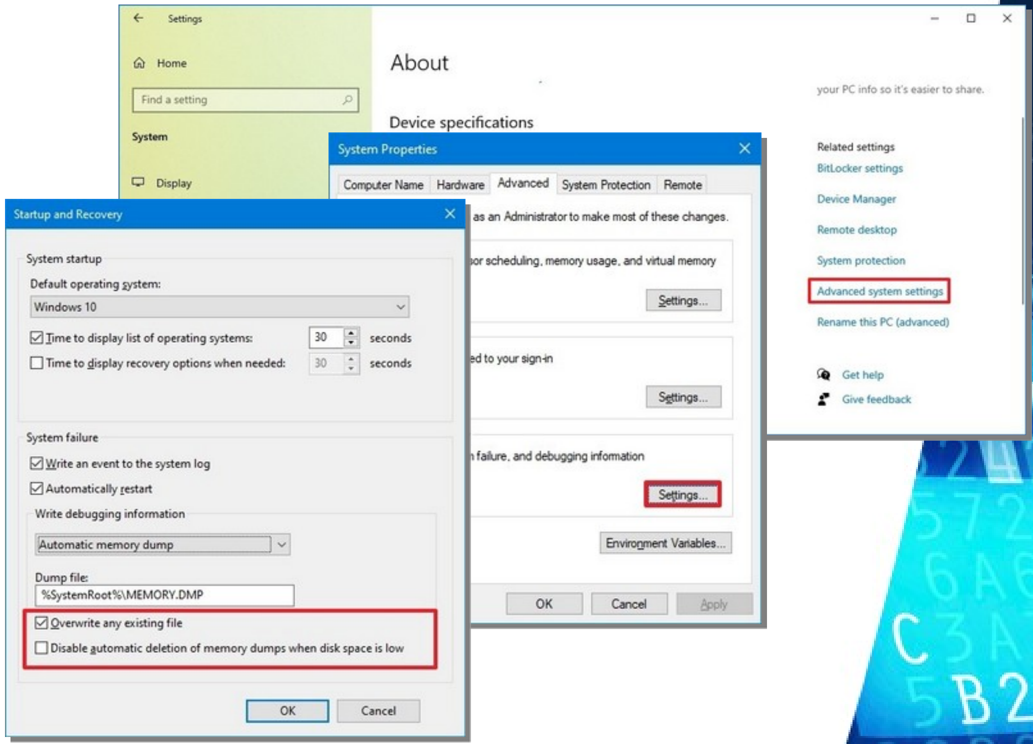
Linux Crash Dump: Live Kernel Dump

- Copy from `/proc/kcore`
  - Copy of the systems memory in ELF format - can be analysed with standard debuggers (Gdb)
  - Huge (terrabytes), but sparse file
  - Need to copy only the occupied pages, see `/proc/iomem`
- Tools:
  - **`getkcore`** from volatility toolkit (`tools/linux/kcore`)
  - **`kcore_dump`** from “schlafwandler”
    - Version that is supposed to work with KASLR for kernel version > 4.8
    - Very little testing, production ready?
- Don’t forget debugging symbols!

Windows 10 Crash Dump: Enable Dump

- Memory Dump Settings (GUI)
  - **Control Panel → System and Security → System**
  - **Advanced system settings → Advanced**
  - **Startup and Recovery → Settings**
  - Select **Kernel memory dump** or **Complete memory dump** under **Writing Debugging Information**
  - Reboot
- CLI

```
wmic recoveros set DebugInfoType=1
wmic recoveros set DebugFilePath=PATH\TO\DUMP
```



Windows 10 Crash Dump: Setting Keyboard Sequence

- To prepare for initiating a crash dump from the keyboard
  - Create one of the following registry keys
  - Depending on your keyboard type

```
HKEY_LOCAL_MACHINE\System\CurrentControlSet\Services\i8042prt\Parameters
Key CrashOnCtrlScroll, Value (DWORD) 0x01      # PS2 keyboards

HKEY_LOCAL_MACHINE\System\CurrentControlSet\Services\kbdhid\Parameters
Key CrashOnCtrlScroll, Value (DWORD) 0x01      # USB keyboards

HKEY_LOCAL_MACHINE\System\CurrentControlSet\Services\hyperkbd\Parameters
Key CrashOnCtrlScroll, Value (DWORD) 0x01      # Hyper-V keyboards
```

Windows 10 Crashdump Execution

- From keyboard (when prepared)
  - Press **right CTRL** key (and hold down) while pressing **SCROLL LOCK twice**
  - To change the key:
    - `https://docs.microsoft.com/en-us/windows-hardware/drivers/debugger/forcing-a-system-crash-from-the-keyboard`
- Alternatively, use the Sysinternals **`NotMyFault`** Tool
  - Part of Sysinternals Suite

```
notMyfault64c.exe /crash reason
```

DFN DEUTSCHES FORSCHUNGSNETZ DFN CERT®

GÉANT

Windows 10 Live Kernel Dump

- Install Windows debugging tools (e.g. from SDK or other source)
- Install LiveKD from Sysinternals

```
 LiveKD.exe
0: kd> .dump /f c:\path\to/dump.dmp
```

DFN
DEUTSCHES FORSCHUNGSNETZ
DFN CERT®

GÉANT

Wrapping Up

Memory Forensic Tool Quality Criteria

- Operating system & Hardware architecture support
- How well does the tool work in adversarial conditions?
  - Rootkits/Anti-Forensics, DRM/Copy-protection SW, faulty memory, etc.
  - Past bugs/vulnerabilities
- GUI, CLI, stand-alone, etc.
- Image file support
  - File types (raw, LiME, etc.)
  - Compression, splitting image over multiple files, …
  - Writing over network (raw, HTTPs)
- Memory footprint?
- Time to capture the memory image? (GiB/s)

DFN DEUTSCHES FORSCHUNGSNETZ DFN CERT

GÉANT

What have you learned?

- There are many way to get to a systems main memory
- Most require some preparation
  - Some even installing hardware beforehand
- Kernel debugging is hard, although very powerful
  - However, requires **a lot** of knowledge & expertise
- Collecting memory through a special kernel module/driver
  - Most generic, with regards to requirements
  - Preparation (i.e. profile building) can be done offline
- Crash dumps can be an alternative
- More coming up: VM hosts, Swap, Hibernation, ...

Thank you

Any questions?

Next Webinar: ***Memory Acquisition II***

December 14th, 2021

www.geant.org

The research leading to these results has received funding from the European Union's Horizon 2020 research and innovation programme under Grant Agreement No. 731122 (GN4-2).

References: Books on Forensics

- Michael Hale Ligh, et al: *The Art of Memory Forensics: Detecting Malware and Threats in Windows, Linux, and Mac Memory*, John Wiley & Sons, Inc. 2014, ISBN: 978-1-118-82509-9
- Bruce Nikkel: *Pactical Forensic Imaging*, No Starch Press Inc. 2016, ISBN-13: 978-1-59327-793-2
- Harlan Carvey: *Windows Forensic Analysis*, Syngress Publishing Inc. 2009

References: Operating System Internals

- Pavel Yosifovich et al: *Windows Internals, Part 1 (System architecture), 7th Ed.*, Microsoft Press 2017, ISBN-13: 978-0735684188
- Allievi Andrea et al: *Windows Internals, Part 2 (Developer Reference), 7th Ed.*, Microsoft Press 2021, ISBN-13: 978-0135462409
- Robert Love: *Linux Kernel Development 3rd Ed.*, Addison-Wesley Professional 2010, ISBN-13: 978-0672329463
- Robert Love: *Linux System Programming: Talking Directly to The Kernel And C Library*, 2nd Ed., O'Reilly2013, ISBN-13 : 978-1449339531
- The FreeBSD Documentation Project: FreeBSD Handbook, `https://docs.freebsd.org/en/books/handbook/`
- The FreeBSD Documentation Project: FreeBSD Developers' Handbook, `https://docs.freebsd.org/en/books/developers-handbook/`
- The FreeBSD Documentation Project: FreeBSD Architecture Handbook, `https://docs.freebsd.org/en/books/arch-handbook/`
- Marshall Kirk McKusick et al.: *The Design and Implementation of the FreeBSD Operating System: Edition 2*, Addison-Wesley Professional 2014, ISBN-13: 978-0321968975

DFN DEUTSCHES FORSCHUNGSNETZ DFN CERT®

GÉANT

References: Images und Testcases

- Computer Forensic Reference Data Sets (CFReDS)
  `http://www.cfreds.nist.gov/`
- Digital Forensics Tool Testing Images
  `http://dftt.sourceforge.net/`
- Digital Forensics Research Workshop (DFRWS)
  `http://www.dfrws.org/`
- Honeynet Project Challenges
  `https://www.honeynet.org/challenges`

References: Memory Imaging Tools (Open Source)

- Microsoft AVML: `https://github.com/microsoft/avml`
- Volatility LiME: https://github.com/504ensicsLabs/LiME
  - Schlafwandlers kcore_dump `https://schlafwandler.github.io/posts/dumping-/proc/kcore/`
- Hal Pomeranz automation script for AVML/LiME: https://github.com/halpomeranz/lmg
- Velocidex Pmem Suite (lin|win|osx)pmem: `https://winpmem.velocidex.com/`
- Moonsols mdd (v 1.3, 2013, for very old Windows versions): `https://sourceforge.net/projects/mdd/`

Sample Forensic Distributions

- SIFT (SAS Investigative Forensic Toolkit): https://www.sans.org/tools/sift-workstation/
- CAINE (Computer Aided Investigative Environment): https://www.caine-live.net/
- GRML Forensic: https://grml-forensic.org/
- ALT Linux Rescue: https://en.altlinux.org/Rescue
- BlackArch: https://blackarch.org/
- BackBox: https://www.backbox.org/
- KALI (formerly Backtrack): https://www.kali.org/downloads/
- Matriux: http://www.matriux.com/
- Safe Boot Disk (Windows based): https://www.forensicsoft.com/help/SAFE_Boot1-1/

References: Standards

- US NIST Special Publication 800-86 *Guide to Integrating Forensic Techniques into Incident Response*, 2006, `https://doi.org/10.6028/NIST.SP.800-86`
- ENISA *Trainings for Cybersecurity Specialists*, `https://www.enisa.europa.eu/topics/trainings-for-cybersecurity-specialists/online-training-material?tab=articles`
- IETF RFC 3227 *Guidelines for Evidence Collection and Archiving*, `https://tools.ietf.org/html/rfc3227`