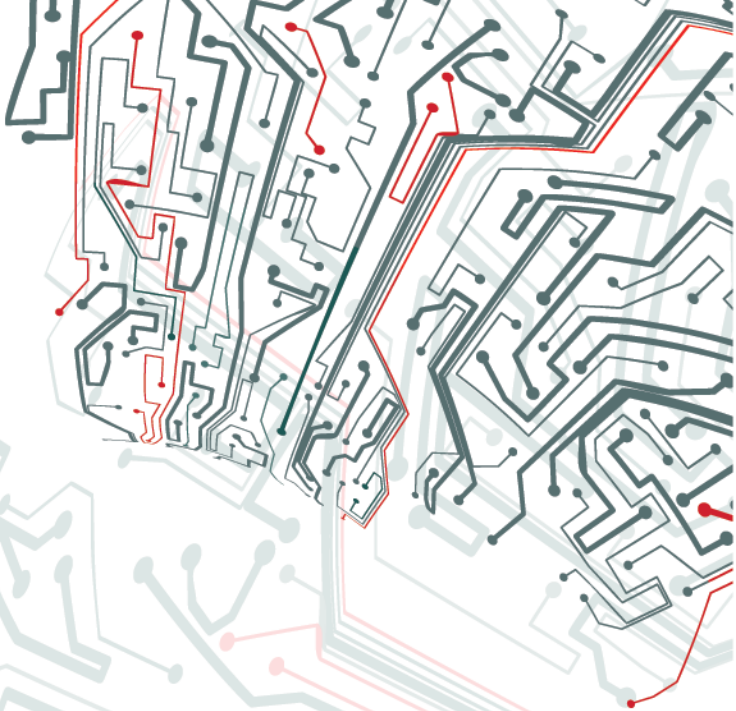
Killing the rootkit - perfect physical memory process detection

Shane Macaulay
Director of Incident Readiness



IOActive
Hardware | Software | Wetware
SECURITY SERVICES

Perfect? Sort of…

- Typical Rootkit/APT method for hiding processes
  - Unlink kernel structures “DKOM”
- New 64bit detection
  - System/Platform independent
  - Linux/BSD/Windows/ARM64/ADM64
    - Ports on the way
- Works by analyzing physical memory & properties of MMU Virtual Memory system

IOActive™

Ideals

- As best as possible, figure out all running code
  - We focus on establishing our understanding through real world targets: Hypervisor monitored guests.

- Combine protection pillars
  1. physical memory traversal (hardware/structure layout)
  2. structure analysis (logical OS interaction)
  3. integrity checking (white listed)

Use a VM

- Hypervisor reduces bare metal pains
  - Establishes verifiability of device state (i.e. not worried about platform attacks e.g. BIOS/firmware/UEFI)
  - Games in fault handler do not work on snapshot, even just extracting physical memory can be hard
  - Protection from virtualized (Dino Dai Zovi), that is serious/obvious impact to performance **when nested**.

IOActive™

What's a Process?

- A Process is an **address space configuration**
  - The configuration "file" is the *page table*
  - A container for threads which are executed on a CPU.
  - Threads share address space.
  - Hard to know if you have all processes.

- Wait, wait?
  - Can't I inject a library/thread to an existing process?
    - Code overwrite or injection is an integrity issue
      - Hash Check

In Memory Process Detection

- Dumping memory is a pain physically
- Scanning VS. List traversal
- Scanning
  - Can be very slow
  - Tends to be high assurance

- Link/Pointer Traversal
  - Easily confused (DKOM attacks)
  - Super Fast !

IOActive™

Process Detection

- Volatility to the rescue! https://code.google.com/p/volatility/wiki/CommandReference#psxview
  - It compares the following **logical** identifiers:
    - PsActiveProcessHead linked list
    - EPROCESS pool scanning
    - ETHREAD pool scanning (then it references the owning EPROCESS)
    - PspCidTable
    - Csrss.exe handle table
    - Csrss.exe internal linked list (unavailable Vista+)

Tool	Virtual Address Translation in Kernel Space	Guessing OS version and Architecture	Getting Kernel Objects
Volatility Framework	2 factors: _DISPATCHER_ HEADER and ImageFileName (PsIdleProcess)	1 factor: _DBGKD_DEBUG_ DATA_HEADER64	2 factors: _DBGKD_DEBUG_ DATA_HEADER64 and PsActiveProcessHead
Mandiant Memoryze	4 factors: _DISPATCHER_ HEADER, PoolTag, Flags and ImageFileName (PsInitialSystem Process)	2 factors: _DISPATCHER_ HEADER and offset value of ImageFileName (PsInitialSystem Process)	None
HBGary Responder	None	1 factor: OperatingSystem Version of kernel header	1 factor: ImageFileName (PsInitialSystem Process)

46

Takahiro Haruyama -- April 2014, discuss his BH Europe 2012 talk with respect to Abort Factors.

IOActive™

64bit Process Detection

- Earlier presentation for kernel code
  - E.g. CSW14 Diff CPU Page table & Logical kernel objects (to detect hidden kernel modules, “rootkit revealer”)

- Also uses page tables “Locating x86 paging structures in memory images” https://www.cs.umd.edu/~ksaur/saurgrizzard.pdf
  - Karla Saur, Julian B. Grizzard
- New process detection technique is faster - single pass
  - Similar to “pmodump”, enhanced with 64bit & additional checks (64bit scan has much more verifiability)

IOActive™

64bit Process Detection Integrity

- Not easily attacked
  - Many modifications result in BSOD
  - Able to extract candidate memory for integrity checking of memory pages to fully qualify
  - Always room to grow with respect to countermeasures and performance

IOActive™

X64 Self MAP

Self pointer

A pointer to self is very powerful

```
7: kd> .formats FFFFF6FB7DBEDF68
Binary: 1111111111111111111101101111011011110110111101101111011011101101000
```

Sign extend	1111111111111111	
PML4 offset	111101101	== 0x1ED
PDP offset	111101101	== 0x1ED
PD offset	111101101	== 0x1ED
Page table offset	111101101	== 0x1ED
Physical page offset	111101101000	== 0xF68 (0xF68 / 8 == 0x1ED)

IOActive™

X64 Kernel Virtual Address Space

http://www.codemachine.com/article_x64kvas.html

Start	End	Size	Description	Notes
FFFF0800`00000000	FFFFF67F`FFFFFFFF	238TB	Unused System Space	**WIN9600 NOW USE & CAN CONTAIN +X AREAS**
FFFFF680`00000000	FFFFF6FF`FFFFFFFF	512GB	PTE Space	**-X used to be executable Win7**
FFFFF700`00000000	FFFFF77F`FFFFFFFF	512GB	HyperSpace	8.1 *seems* to have cleaned up here, 9200 had 1 +X page
FFFFF780`00000000	FFFFF780`00000FFF	4K	Shared System Page	
FFFFF780`00001000	FFFFF7FF`FFFFFFFF	512GB-4K	System Cache Working Set	
FFFFF800`00000000	FFFFF87F`FFFFFFFF	512GB	Initial Loader Mappings	Large Page (2MB) allocations
FFFFF880`00000000	FFFFF89F`FFFFFFFF	128GB	Sys PTEs	
FFFFF8a0`00000000	FFFFF8bF`FFFFFFFF	128GB	Paged Pool Area	
FFFFF900`00000000	FFFFF97F`FFFFFFFF	512GB	Session Space	
FFFFF980`00000000	FFFFFa70`FFFFFFFF	1TB	Dynamic Kernel VA Space	
FFFFFa80`00000000	*nt!MmNonPagedPoolStart-1	6TB Max	PFN Database	
*nt!MmNonPagedPoolStart	*nt!MmNonPagedPoolEnd	512GB Max	Non-Paged Pool	**DEFAULT NO EXECUTE**
FFFFFFFF`FFc00000	FFFFFFFF`FFFFFFFF	4MB	HAL and Loader Mappings	

IOActive™

Self Map detection Windows AMD64

- Self Map exists for each process (not only kernel:)
- Examining a page table - !process 0 0 → dirbase/cr3

!dq 7820e000

#7820e000 00800000`60917**867**

[physical addr] [value]

*!dq 7820e000+**0xf68***

#**7820e**f68 80000000`**7820e**863

^-- *current PFN found* --^ **(PFN FTW)**

IOActive™

PFN FTW Trick! (or Defensive exploit!!)

#7820ef68 80000000`7820e863

^----------^

64Bit is a more powerful check

Valid PFN will be bounded by system physical memory constraints

Valid self map **address** will always increase from previous

IOActive™

These are the BITs your looking for…

```
typedef struct _HARDWARE_PTE {
    ULONGLONG Valid : 1;                  ← Indicates hardware or software handling (mode 1&2)
    ULONGLONG Write : 1;
    ULONGLONG Owner : 1;
    ULONGLONG WriteThrough : 1;
    ULONGLONG CacheDisable : 1;
    ULONGLONG Accessed : 1;
    ULONGLONG Dirty : 1;
    ULONGLONG LargePage : 1;              ← Mode2
    ULONGLONG Global : 1;
    ULONGLONG CopyOnWrite : 1;
    ULONGLONG Prototype : 1;              ← Mode2
    ULONGLONG reserved0 : 1;
    ULONGLONG PageFrameNumber : 36;       ← PFN, always incrementing (mode 1&2)
    ULONGLONG reserved1 : 4;
ULONGLONG SoftwareWsIndex : 11;           ← Mode2
ULONGLONG NoExecute : 1;
} HARDWARE_PTE, *PHARDWARE_PTE;
```


IOActive™

Base PageTable offsets

Below example of 512-way page table

PTEntry	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

...

Offset 0x1ED below

0	0	0	0	0	0	0	**SELF PTEntry**	0	0
0	PTEntry	0	PTEntry	0	0	0	0	PTEntry	PTEntry

These are the OFFSETS your looking for.

- 512 way Table (512 * 8 = 0x1000, a page)
  - PFN Offset 0 configured and valid bit
  - PFN Offset 0x1ed Point's to self and valid bit
    - This allows us to identify *current position

- Mode2 has more checks for typical page table
- Mode1 is for heightened assurance
  - Both work together to extract PFN & MEMORY_RUN gaps
  - http://blockwatch.ioactive.com/MProcDetect.cs

IOActive™

Self Map Detection Attacks

- Attacks against performance
  - If we de-tune performance we can validate spoof entries and various malformed cases
  - Windows zero's memory quickly (no exiting processes, so far:)
- **!ed** [physical] can be done to assess evasive techniques
  - Simply destroying self map results in **BSOD!!** ☺
  - *Looking for feedback testing to identify better more comprehensive PTE flag checks (edge cases, missed tables or extra checks)*

IOActive™

Implementation (basically in 1 line)

```
// scan every page from lpMapping to lpMapping+MAP_SIZE
for(unsigned long long i=0; i < WinLimit; i+=512)
{
    // first entry of table should not be null and end in 0x867
    // lower bits 0x867 configured
    if(lpMapping[i] != 0 && (lpMapping[i] & 0xfff) == 0x867)
    {
        // self map should be at index 0xf68/8 == 0x1ed
        ULONGLONG selfMap = lpMapping[i+0x1ED];

        // if we can find a possiable self map, extract current PFN
        ULONGLONG low12Bits = selfMap & 0xfff;
        if(selfMap != 0 && (low12Bits == 0x863 || low12Bits == 0x063))
        {
            ULONGLONG offset = CurrWindowBase+(i*8);
            MMPTE_64 selfPTE;
            selfPTE.u.Long.QuadPart = selfMap;

            ULONGLONG shift = (selfPTE.u.Hard.PageFrameNumber << PAGE_SHIFT);
            ULONGLONG diff = offset > shift ? offset - shift : shift - offset;

            printf("Possiable Directory Base Register Value = [%llx]  File Off
```


IOActive™

```
Server 2003 Enterprise x64 Edition SP0-4f22a4c3.vmem"
Starting map scan for file
Possiable Directory Base Register Value = [aab27000]  File Offset = [aab27000], Diff = [0]
Possiable Directory Base Register Value = [aab72000]  File Offset = [aab72000], Diff = [0]
Possiable Directory Base Register Value = [ab40d000]  File Offset = [ab40d000], Diff = [0]
Possiable Directory Base Register Value = [ab69c000]  File Offset = [ab69c000], Diff = [0]
Possiable Directory Base Register Value = [ab992000]  File Offset = [ab992000], Diff = [0]
Possiable Directory Base Register Value = [ac0c0000]  File Offset = [ac0c0000], Diff = [0]
Possiable Directory Base Register Value = [ac2fb000]  File Offset = [ac2fb000], Diff = [0]
Possiable Directory Base Register Value = [ac462000]  File Offset = [ac462000], Diff = [0]
Possiable Directory Base Register Value = [aca8b000]  File Offset = [aca8b000], Diff = [0]
Possiable Directory Base Register Value = [ad3d0000]  File Offset = [ad3d0000], Diff = [0]
Possiable Directory Base Register Value = [ad521000]  File Offset = [ad521000], Diff = [0]
Possiable Directory Base Register Value = [ade8b000]  File Offset = [ade8b000], Diff = [0]
Possiable Directory Base Register Value = [ae184000]  File Offset = [ae184000], Diff = [0]
Possiable Directory Base Register Value = [aea3f000]  File Offset = [aea3f000], Diff = [0]
Possiable Directory Base Register Value = [aec6c000]  File Offset = [aec6c000], Diff = [0]
Possiable Directory Base Register Value = [aed12000]  File Offset = [aed12000], Diff = [0]
Possiable Directory Base Register Value = [af206000]  File Offset = [af206000], Diff = [0]
Possiable Directory Base Register Value = [af397000]  File Offset = [af397000], Diff = [0]
Possiable Directory Base Register Value = [afca4000]  File Offset = [afca4000], Diff = [0]
Possiable Directory Base Register Value = [b0474000]  File Offset = [b0474000], Diff = [0]
Possiable Directory Base Register Value = [b05ff000]  File Offset = [b05ff000], Diff = [0]
Possiable Directory Base Register Value = [b09ab000]  File Offset = [b09ab000], Diff = [0]
Possiable Directory Base Register Value = [b0e64000]  File Offset = [b0e64000], Diff = [0]
Possiable Directory Base Register Value = [b11bd000]  File Offset = [b11bd000], Diff = [0]
Possiable Directory Base Register Value = [b131e000]  File Offset = [b131e000], Diff = [0]
Possiable Directory Base Register Value = [b1380000]  File Offset = [b1380000], Diff = [0]
Possiable Directory Base Register Value = [b15d7000]  File Offset = [b15d7000], Diff = [0]
Possiable Directory Base Register Value = [b1f2d000]  File Offset = [b1f2d000], Diff = [0]
Possiable Directory Base Register Value = [b1f99000]  File Offset = [b1f99000], Diff = [0]
Possiable Directory Base Register Value = [b1fae000]  File Offset = [b1fae000], Diff = [0]
Possiable Directory Base Register Value = [b2827000]  File Offset = [b2827000], Diff = [0]
Possiable Directory Base Register Value = [b4b56000]  File Offset = [b4b56000], Diff = [0]
Possiable Directory Base Register Value = [1181f1000]  File Offset = [d81f1000], Diff = [400000000]
Possiable Directory Base Register Value = [1190001000]  File Offset = [d90001000], Diff = [400000000]
end map scan
detected page tables = 34
```

Example execution (.vmem starts @0 offset), .DMP (0x2000+) or other autodetect header offset ☺

IOActive™

Detected Memory Runs

- Round value by offset to find gap size, adjust to automate memory run detection
  - Takahiro Haruyama blog post on related issue (large memory) and also memory run detection issues from logical sources
- *previous slide, detecting gap, when offset changes;
  - ROUND_UP(0xb4b56000, 0x40000000) = first run end 0xc0000..
  - ROUND_DOWN(0x1181f1000, 0x40000000)

Detect processes of guests from host dump

- A host memory dump will include page tables for every guest VM process as well as host process entries
  - Lots of room to grow here, deep integration with HyperVisor page mapping data may be straight forward
    - E.g. parsing of MMInternal.h / MMPAGESUBPOOL in VirtualBox
- Issues
  - Hypervisor may not wipe when moving an instance or after it's been suspended (ghost processes)
    - I'd rather detect ghosts than fail ☺
- Nested paging not a problem

IOActive™

```
P:\>ProcDetect.exe c:\Windows\MEMORY.DMP
Starting map scan for file
Possible Directory Base Register Value = [1aa000]  File Offset = [330000], Diff = [186000]
Possible Directory Base Register Value = [14c2000]  File Offset = [1648000], Diff = [186000]
Possible Directory Base Register Value = [15e8000]  File Offset = [176e000], Diff = [186000]
^C
```

Initial values reflective of host system, consistent Diff values

```
Possible Directory Base Register Value = [19cafa000]  File Offset = [47b64a000], Diff = [2deb50000]
Possible Directory Base Register Value = [187000]  File Offset = [4a8890000], Diff = [4a8709000]
Possible Directory Base Register Value = [6a02000]  File Offset = [4b99d4000], Diff = [4b2fd2000]
Possible Directory Base Register Value = [719e000]  File Offset = [4ba257000], Diff = [4b30b9000]
Possible Directory Base Register Value = [8356000]  File Offset = [4bb521000], Diff = [4b31cb000]
Possible Directory Base Register Value = [18b79000]  File Offset = [4cbf8c000], Diff = [4b3413000]
```

Skew is evident for guest instances. An typical kernel PFN is observed (*187*) as the first (large jump 0x2..->0x4…) in a range of skewed diff values (another layer of decoding to adjust, similar to what happens when snapshot is requested and disk memory is serialized)

```
Possible Directory Base Register Value = [b5b06d000]  File Offset = [b13055000], Diff = [48018000]
Possible Directory Base Register Value = [b6b3bd000]  File Offset = [b233a5000], Diff = [48018000]
end map scan
detected process page tables = 170
```

Final host processes identifiable by Diff realignment

IOActive™

Self Map trick in Linux

- **Virtual Memory in the IA-64 Linux Kernel**
  - Stephane Eranian and David Mosberger
    - 4.3.2 Virtually-mapped linear page tables

“linear page tables are not very practical when implemented in physical memory”

“The trick that makes this possible is to place a self-mapping entry in the global directory.”

IOActive™

Issues, Considerations Caveats

- Use a hypervisor – secure the guest/host (very hardened host)
  - Hypervisor escape == you're a high value to risk nice exploit
    - Probably NOT YOU!
    - BluePill type attacks, hopeful still to consider (but perf hit of nesting should be obvious)
- SefMap Detection relies on page table.
  - Maybe "no paging process"– (same as x86 paging paper)
  - TSS considerations, monitor other tables with stacks?
  - Remote DMA?
    - Please no! ☹

IOActive™

Summary

- Always use a VM
  - At least simplify memory dumping
- Use ProcDetect
  - Have fun detecting!
  - Process hiding rootkit is dead
  - 64bits helps peace of mind
- We can detect a process anywhere (host, guest, nested, on the network (probably☺)
- RoP & other attacks? Check out CSW14 and DC22 slides

IOActive™

Attention Wikipedia editors DKOM ☺

“Not only is this very difficult to..”

We have a high assurance capability, applicable cross 64bit platforms (linux/freebsd also arm64, etc…) , for process detection.

Even though threads are distinct execution contexts, the property of shared MMU configuration establishes a verification capability that OS kernel object manipulation can not effect.

IOActive™

Thank you & Questions

- I hope I referenced earlier works sufficiently, this topic is broad and expansive, thanks to the many security professionals who analyze memory, reverse-engineered, dove deep and discussed their understanding.
- References, follow embedded links and their links

IOActive™