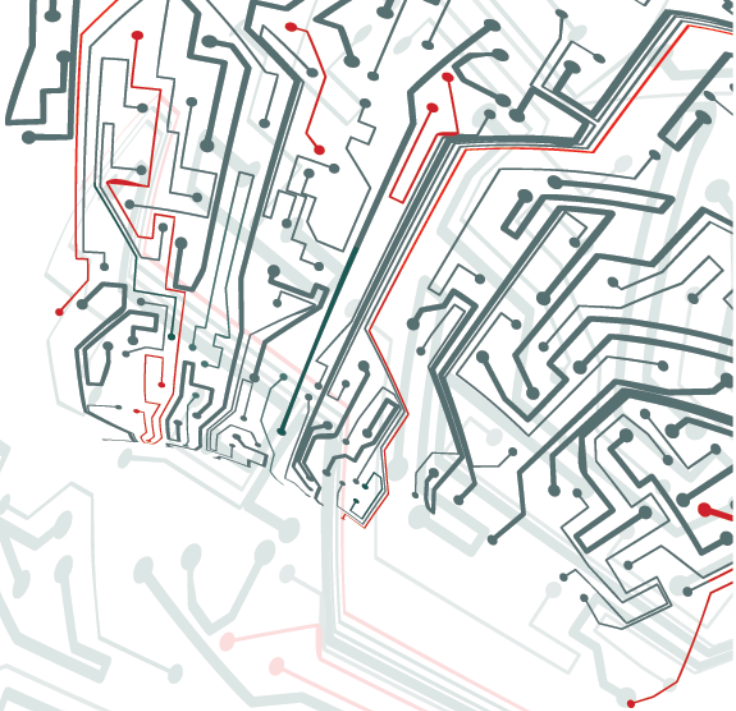
Weird-Machine Motivated Practical Page Table Shellcode & Finding Out What's Running on Your System

Shane Macaulay
Director of Cloud Services



IOActive
Hardware | Software | Wetware
SECURITY SERVICES

Killing the Rootkit! And how to find everything running on your system!!!

- Rootkit/APT technique for hiding processes
  - Unlink kernel structures “DKOM”

- New 64bit detection technique ! DC22 exclusive ☺
  - System/Platform independent technique
  - Linux/BSD/Windows/ARM64/ADM64

- Works by analyzing physical memory & properties of MMU Virtual Memory system

IOActive™

The Long Road

- Barnaby Jack, forever in our hearts and minds.

  “It’s about the journey not the destination.”

IOActive™

13 Years since ADMMutate (slide URL)

http://1drv.ms/1rEBMJF

- ADMmutate (last DC talk was about polymorphic shellcode)
- The more things change
  - The more they stay the same
- Thought about PT shellcode with ADMMutate

- Attack is [hard/stress/]**fun!!&$&%*:P;p;P**
- Defense is hard/stress

Abusing x for fun & profit!

- It's usually the QB that get's the headlines, offensive bias in hacker scene!

- Defense is grind's it out for little glory.
  - Let's energize the "D" here, have some fun!!

- A Defensive exploit
  - Ultimately today were killing process hiding rootkits cross 64bit OS/Platforms **TODAY**! ☺
  - DKOM IS DEAD! Process hiding is DEAD!

IOActive™

Also 13 Years ago

- What else was going on back then?
  - **x86 assembler in Bash**

"cLIeNUX"

"shasm is an assembler written in GNU Bash Version 2, which may work in other recent unix-style "shell" command interpreters."

IOActive™

Ideals

- As best as possible, figure out all running code
  - Code/hacks/weird machine's included/considered
  - When have we done enough?
- We focus on establishing our understanding through real world targets: Hypervisor monitored guests.
- Combine protection pillars; structure analysis, physical memory traversal and integrity checking.

IOActive™

Practical concepts

- Attacks: WeIrD MaChinE
  - Lots of fun!
    - Much esoteric/eclectic More fantastical!!!
- Defense: Detecting * ← That means everything
  - Home field == USE THE “FORCE” A **HYPERVISOR!**
    - 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™

Practical Page Table ShellCode Motivations

- An attack devised to understand memory protection systems
  - Development necessitated comprehensive understanding of inner workings, system fault handling complexities and some of the lowest level (brain melting, see reference below) interaction of software and hardware on modern 64bit platforms.
  - Until Windows 7, page tables directly executable
    - NonExecutable is opt-in/non-default

– **The page-fault weird machine: lessons in instruction-less computation**

  - Julian Bangert, Sergey Bratus, Rebecca Shapiro, Sean W. Smith from WOOT'13 Proceedings of the 7th USENIX conference on Offensive Technologies

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™

Page Table ShellCode weird-machine

- Win7- and earlier
  - Can we emit intended shellcode into PTE area?
    - Call VirtualAlloc() from user space results in executable memory in kernel
      - Just reserving memory causes a code-write operation into kernel space

```
PXE at FFFFF6FB7DBEDF68    PPE at FFFFF6FB7DBEDF88    PDE at FFFFF6FB7DBF1008    PTE at FFFFF6FB7E201EA0
contains 0000000000187063  contains 0000000134C04863  contains 0000000100512863  contains 000000002DC3B863
pfn 187       ---DA--KWEV  pfn 134c04    ---DA--KWEV  pfn 100512    ---DA--KWEV  pfn 2dc3b     ---DA--KWEV
```


IOActive™

PT SC WM Died with Win8 (below)

- This works earlier than Win7, interesting to examine fault handling, but ultimately Win8 this is dead! 😐

```
Child-SP          RetAddr           Call Site
ffffd000`2b34ecf8 fffff800`16066ee1 nt!LOCK_WORKING_SET
ffffd000`2b34ed00 fffff800`1603f5ad nt!MiSystemFault+0x911
ffffd000`2b34eda0 fffff800`1615af2f nt!MmAccessFault+0x7ed
ffffd000`2b34eee0 fffff6fb`77fde37a nt!KiPageFault+0x12f
ffffd000`2b34f078 fffff800`01e423fe 0xfffff6fb`77fde37a
ffffd000`2b34f080 fffff800`163ae3e5 SIoctl!SioctlDeviceControl+0x27e
ffffd000`2b34f9b0 fffff800`163aed7a nt!IopXxxControlFile+0x845
ffffd000`2b34fb60 fffff800`1615c4b3 nt!NtDeviceIoControlFile+0x56
ffffd000`2b34fbd0 00007ff9`c1b265ea nt!KiSystemServiceCopyEnd+0x13
0000003a`ba9bf8f8 00007ff9`bef92c83 ntdll!NtDeviceIoControlFile+0xa
```

What about new tool (wanted ptshellcode thingy)?

- Was going to do a talk with an expansion of the PT shellcode concept
  - Was it going to be an ADMmutate update? .NET Compiler thingy some set of C macro's or little script host RoP builder/engine/host?

- Application of technique is mostly dead, requires an info leak(maybe) and what about use bash to write it?

Some peace of mind – really!

- cross platform AMD64 process detection technique
  - obsoletes *process hiding* techniques used by all rootkits/malware!
    - Process hiding rootkits/malware technology being typical of APT

- Detection can be used as an attack (defensive attack pattern)
  - Defensive Exploit against ALL ROOTKITS!

IOActive™

The big picture ProcDetect

- Ultimately decided on a more advanced, and useful, tool for release today
  - Hear it for the D!

- ProcDetect should be with DefCon materials
  - Signed code example for AMD64 Windows
    - Other platform/OS to follow ☺

IOActive™

Attack v Defense

- Defensive Window of opportunity
  - Closing the door/window today!
- Defensive tactics can be new classes of defensive attack techniques
  - Offensive Forensics / Automation
  - Use the process detection here to post process and detect any/every hidden process ever spawned for all TIME! ☺
    - Keep interesting/known memory dumps around

    Right now; there are no possible attacks against *this technique* (“WE FOUND YOU!”)

IOActive™

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
  - Super Fast !

IOActive™

What's a Process?

- A Process is an **address space configuration**
  - A container for threads which are executed on a CPU.
  - Threads share address space.
  - Hard to know if you have them all.

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

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)

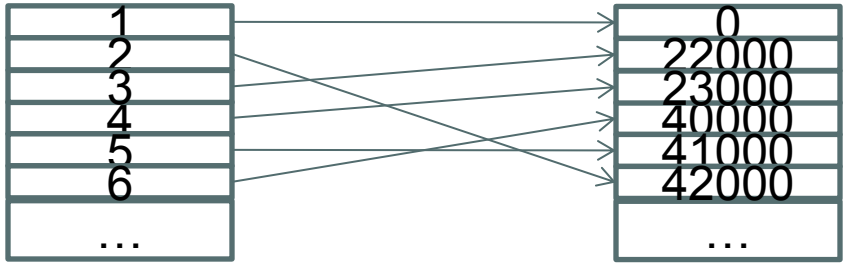
IOActive™

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
    - Can make “non-abortable” if willing to do slower check
    - Current check is really good
  - Always room to grow with respect to countermeasures and performance

IOActive™

Physical/Virtual Memory



Page Frames
4k physical blocks
A Page Frame Number (PFN) is the physical memory “address”

Virtual Address/Pages
Page protection is applied to virtual pages/address ranges

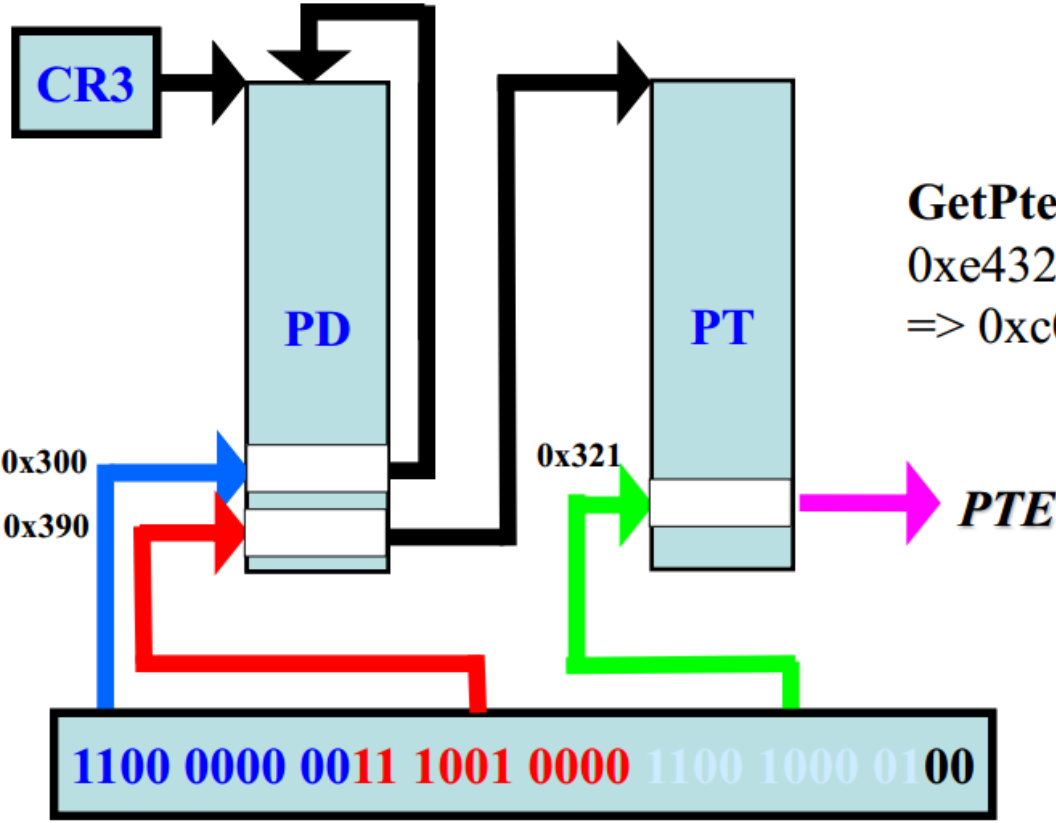
IOActive™

A quick indirection

- Slides 37-39 from Dave Probert (Windows Kernel Architect, Microsoft)
  - Windows Kernel Architecture Internals
- Next slide show's a big hint, can you guess? It's an example of process page table layout/configuration.
  - You have to love all of those arrow's ☺

Self-mapping page tables

Virtual Access to PTE for va 0xe4321000



GetPteAddress:
0xe4321000
=> 0xc0390c84

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™

Self Map process detection Windows AMD64

- Self Map exists for each process (not only kernel:)
- Examining a page table - !process 0 0 → dirbase/cr3 (e.g. 7820e000)

```
!dq 7820e000
#7820e000 00800000`60917867
!dq 7820e000+0xf68
#7820ef68 80000000`7820e863
     ^-- 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

IOActive™

These ARE the bit's 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™

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)*

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 (*scream 187 to a mo…*) 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™

Future Weird Machine overload ? ☹

- Microsoft Research
  - Tracking Rootkit Footprints with a Practical Memory Analysis System -- Weidong Cui, Marcus Peinado, Zhilei Xu, Ellick Chan
  - *"The goal of MAS is to identify all memory changes a rootkit makes…. MAS does so in three steps: static analysis, memory traversal and integrity checking"*

- Seems really hard problem (source code used in MAS), how can *we* verify this level of state?

IOActive™

Public symbols to the rescue'ish ☺

- Public symbols, RTTI or other type inference technique to find/root(tree/linked) all pointers
  - Thread stack return into verifiable code
    - Anti RoP Attack
  - Advanced methods kernel pool (does not require source) verification
    - Integrity Checking of Function Pointers in Kernel Pools via Virtual Machine Introspection
  - At least kernel alerts, logs and various tracing can be trusted if we have code integrity, process/thread detection.
  - Future is not too bad for Defense!

Summary

- Attacks: WeIrD MaChInE
  - Worst case scenario most weird machine activity can hopefully be detected through simple tracing, logging and monitoring tools
    - What about the next GPU/UEFI backdoor? → use a hypervisor guest to establish device/low layer trust capability
- Defenses: Detecting hidden 64bit processes
  - Deep future holds deep verifiability for more devices ☺ (get free The Memory Cruncher™ TMC & BlockWatch ™)
    - Active Protection System (APS)
- FINALLY DEFENSIVE **FUN & PROFIT!** With the D!

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☺)!

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! ☹

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™