LOL (Layers On Layers) - Bypassing endpoint security

Rafal Wojtczuk, rafal@bromium.com

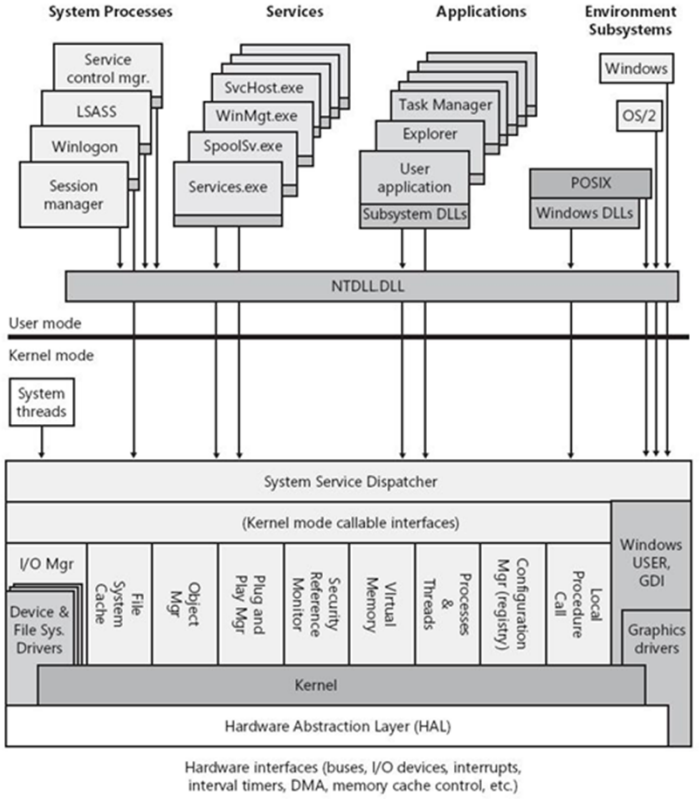
Rahul Kashyap, rahul@bromium.com

Agenda

- Describing the Layers - Architecture
  - AV, HIPS, EMET, Sandboxes, Rootkit Detectors, SMEP
- Exploitation discussion
- LOL

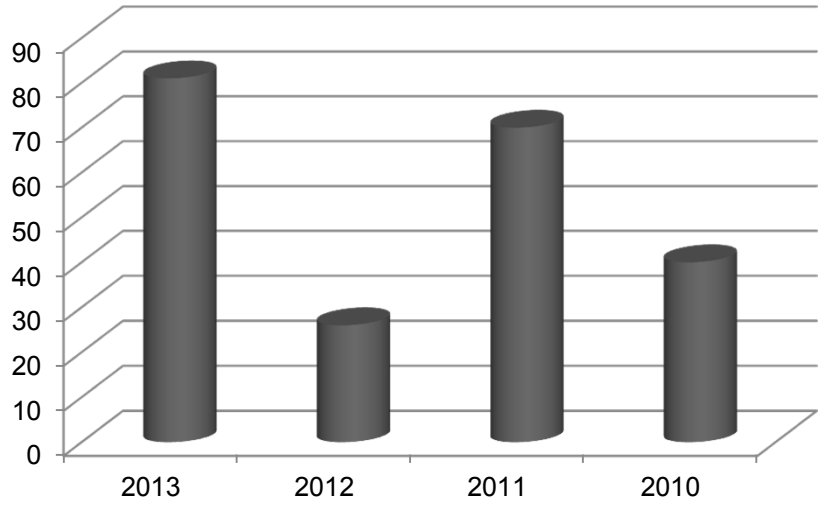
Kernelmode vs usermode

- Most Endpoint Security solutions focus on user mode protection
- Kernel mode is a huge attack surface with limited coverage
- Even the kernel mode focused protections are ill-equipped to defend



Kernel Vulnerabilities

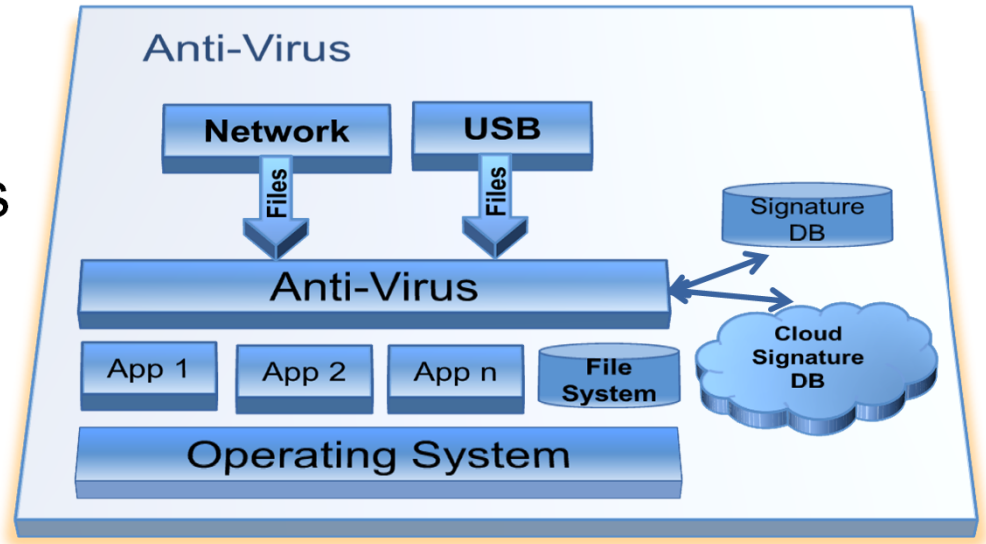
- 200+ Kernel CVE's from Microsoft since 2010
- Stuxnet, Duqu, Gapz, Gameover, CVE-2013-5065 (NDProxy.sys), TDL4 – (to name a few) uncovered in the wild



Kernel (in)-security trends

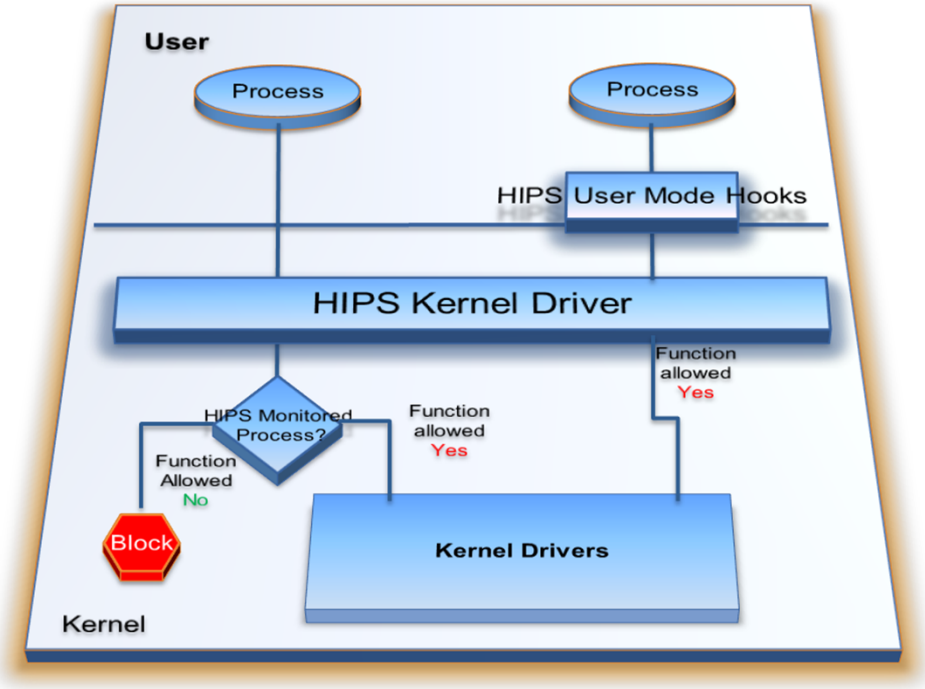
Layer 1: Anti-Virus

- Signature detection
- Heuristic detection
- File based access controls
- Reputation based controls



Layer 2: Host IPS

- Usermode exploitation prevention (out of scope)
- Extra logging of user's actions
- System integrity checks (similar to Patchguard)
- Limit abilities of user processes, by custom kernel code, not relying on security boundaries enforced by the OS
  - forbid stopping HIPS usermode services
  - forbid loading non-whitelisted kernel drivers
  - forbid injecting code into "protected" processes, e.g. lsass.exe
- 'Vulnerability' signatures are more effective than 'exploit' signatures.



Layer2: Host IPS (contd)

- Some Host IPS block attacks leveraging signatures at the TCP/IP layer
  - These are essentially signatures/heuristics by DPI at the protocol layer
    - Advantages: it's less intrusive (limited endpoint hooking), better performance.
    - Disadvantages: that these can be evaded with protocol/pattern based evasions

Layer 3: EMET

- Probably one of the best anti-exploitation mitigation tool (and…it's free!)
- Heavily focused on user mode, no protection against kernel mode exploitation

EMET Security Mitigations	Included
Data Execution Prevention (DEP) Security Mitigation	✓
Structured Execution Handling Overwrite Protection (SEHOP) Security Mitigation	✓
NullPage Security Mitigation	✓
Heapspray Allocation Security Mitigation	✓
Export Address Table Filtering (EAF) Security Mitigation	✓
Mandatory Address Space Layout Randomization (ASLR) Security Mitigation	✓
Bottom Up ASLR Security Mitigation	✓
Load Library Check – Return Oriented Programming (ROP) Security Mitigation*	✓
Memory Protection Check – Return Oriented Programming (ROP) Security Mitigation*	✓
Caller Checks – Return Oriented Programming (ROP) Security Mitigation*	✓
Simulate Execution Flow – Return Oriented Programming (ROP) Security Mitigation*	✓
Stack Pivot – Return Oriented Programming (ROP) Security Mitigation*	✓

** Available and applicable only to 32-bit processes*

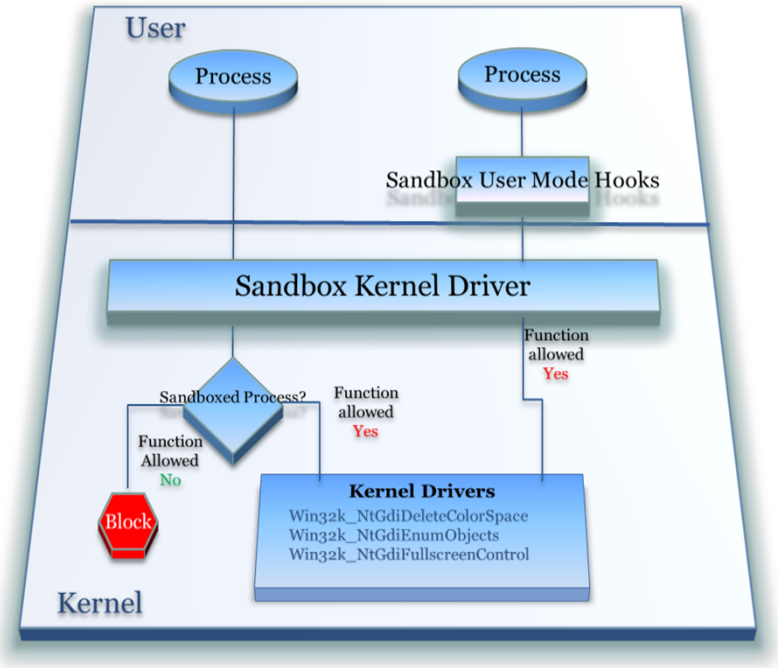
*** EMET supports Windows 7, Windows 8, Windows 8.1, Windows Server 2003 Service Pack 1, Windows Server 2008, Windows Server 2008 R2, Windows Server 2012, Windows Server 2012 R2, Windows Vista Service Pack 1, and Windows XP Service Pack 3.*

Source: Microsoft.com

Layer 4: App Sandboxes

- Two types: App specific (Chrome, Adobe Acrobat) and kernel driver initiated (Sandboxie, Bufferzone, etc)
- All types are vulnerable to kernel mode exploitation
- Chrome has specific hardening to the sandbox, but still exposed to win32k.sys vulns

Ref:
http://labs.bromium.com/2013/07/23/application-sandboxes-a-pen-testers-perspective/



Hidden (bonus?) Layer: Patchguard

- Primarily designed to prevent kernel mode rootkits on x64 bit Windows OS.
- Patchguard is code running in ring0 (just like any other kernel driver)
- It tries to protect the following:
  - NTOS, HAL etc. (key system modules)
  - SSDT, GDT, IDT
  - Certain MSRs (which we discuss later)
- Historically it has been bypassed several times (and fixed..and bypassed)
- Several instances where Patchguard has been disabled recently
- Recommended Read:

http://www.mcafee.com/us/resources/reports/rp-defeating-patchguard.pdf

Kernel mode Rootkits

- Kernel mode rootkits
  - Can intercept native API in kernel mode.
  - Manipulate kernel data structures.
  - Remain ‘hidden’

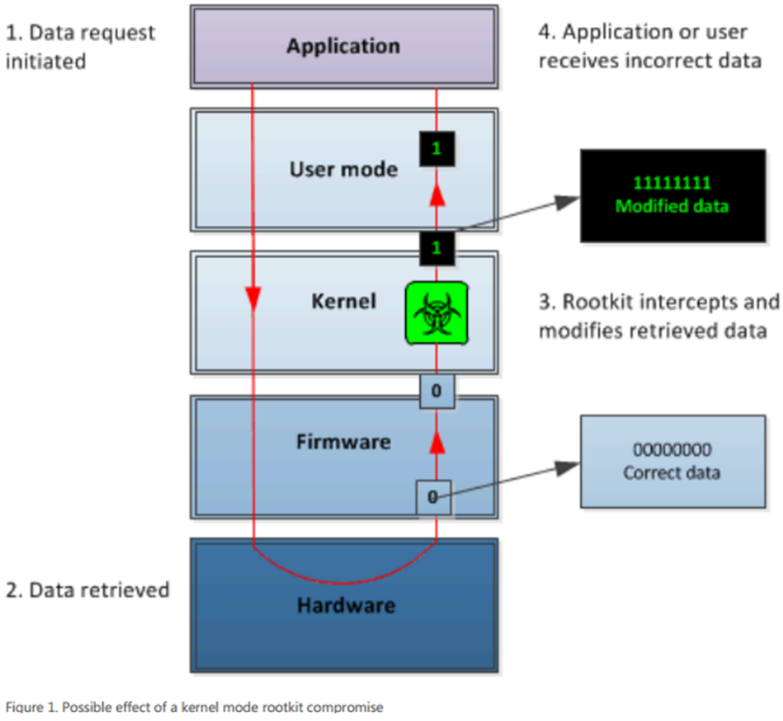
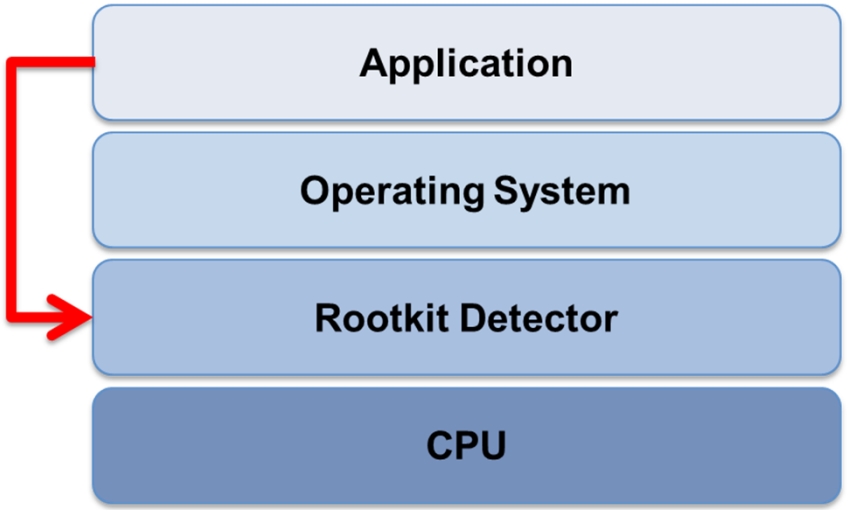


Figure 1. Possible effect of a kernel mode rootkit compromise

Source: Microsoft

Layer 5: Kernel Rootkit Detector

- Preventing and logging write attempts to the system's interrupt descriptor table (IDT) and the system service dispatch table (SSDT)
- Stopping changes to the processor system transitioning table
- Preventing modifications to the direct kernel object manipulation (DKOM) list and threads
- Eliminating malicious attachments to kernel mode drivers
- Prohibiting malicious inline hooking to kernel code sections along with key device drivers
- Stopping malicious modifications to drivers' import address table (IAT) hooking
- Preventing malicious modifications to kernel export address table (EAT)
- Stopping malicious I/O calls from device drivers
- Detecting malicious changes to drivers' dispatch routines

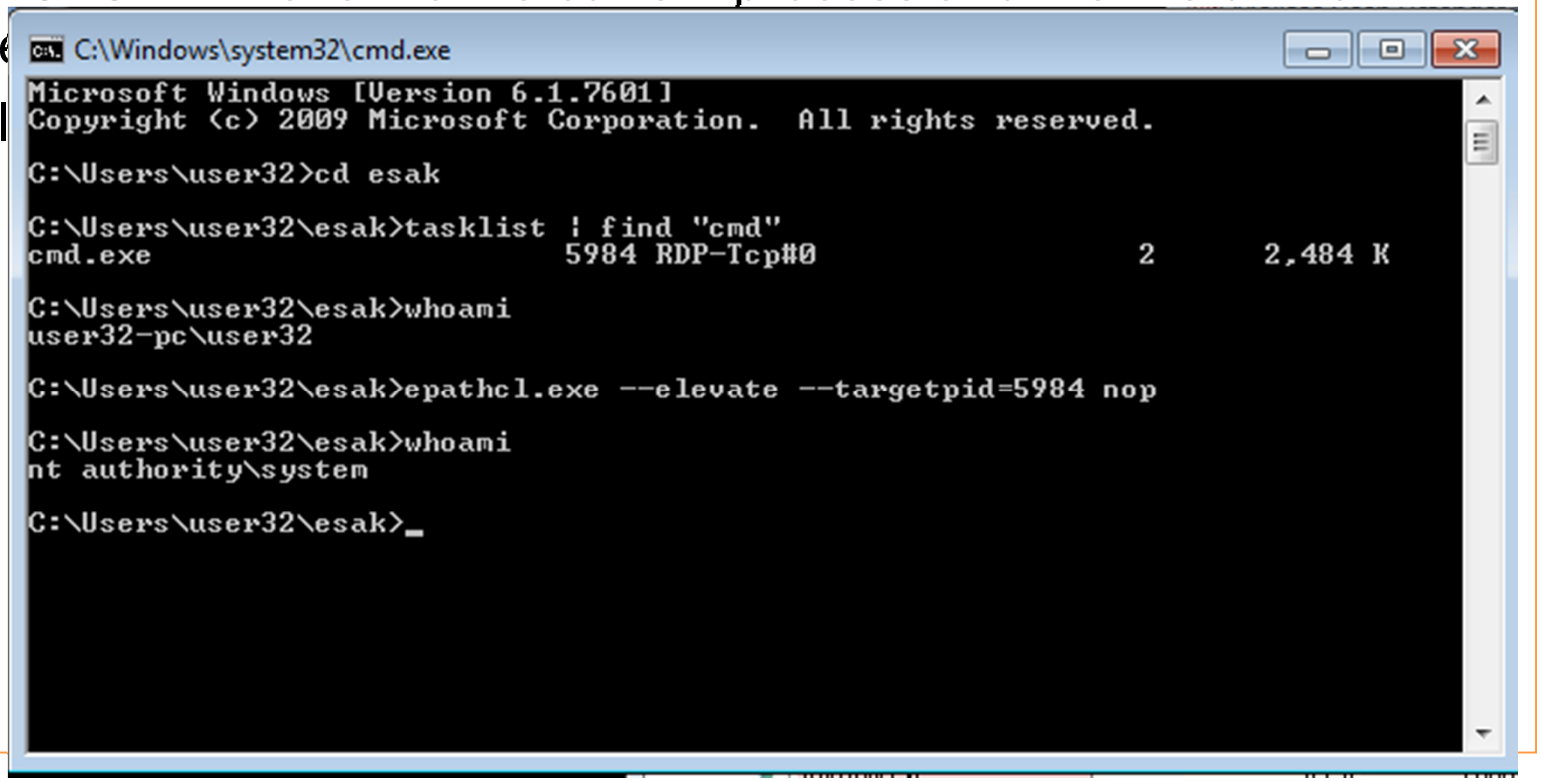


Kernel mode exploits – Quick Review

- Achieve code execution in usermode app (e.g. browser)
- Run kernel exploit code
- Run useful kernelmode payload

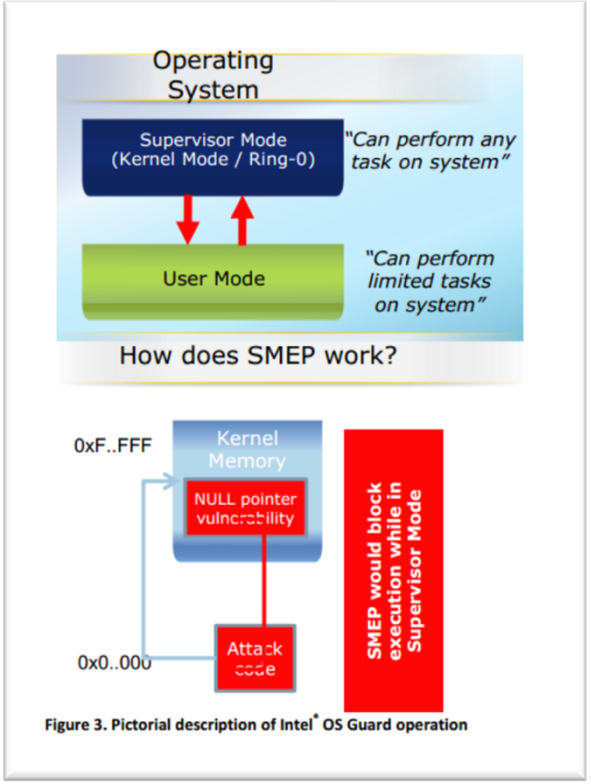
Typical kernel mode payload

- Grant the SYSTEM token to the current process and then return to usermode
- Almost al



Layer 6: SMEP (Supervisor Mode Execution Protection)

- Forbid running code from usermode page in kernel mode, by setting relevant SMEP bit in CR4 CPU register
- Assumption is: no way for a user to create arbitrary executable code in kernel pages (broken on 32bit Windows btw)
- Generic bypass - kernelmode ROP, possibly via SMEP-disable (that clears SMEP bit)



Source: Intel

Kernel mode ROP

- Traditional Kernel mode payloads use shellcode in user space memory
- Leverage the classic ret-2-libc conceptually (to bypass SMEP) and create gadgets to stay in ring0
- The first gadget should clear SMEP

```
1. mov eax, cr4
2. btr eax, 20
3. mov cr4, eax
4. jmp 0x0baaaaad
```

- Several ways to do this..

- Read the blog by j00ru on this http://j00ru.vexillium.org/?p=783

There are many other options...

- Once attacker has gained code execution in the kernel context, all security measures implemented in kernelmode are bypassable
- Attacker might need to thoroughly reverse engineer a given product in order to disable it entirely in a stable manner; but there are a few generic methods to cripple/ignore the protection
  - Clear kernel callback tables – make the watcher blind
  - Migration/code injection to arbitrary usermode processes
  - Not guaranteed to work against every security solution, but expected to work against many

Exploitation

CVE-2013-3660
(a.k.a EPATHOBJ)

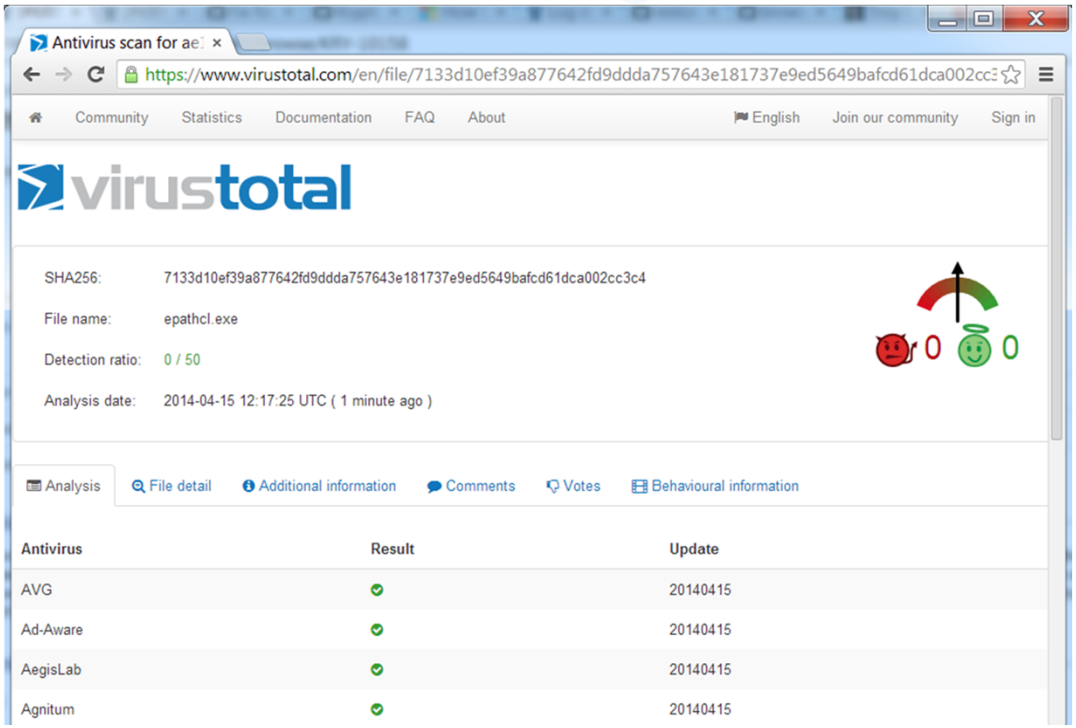
Stages of an attack – AV

- Achieve code execution in usermode app (e.g. browser)
- Run kernel exploit code
- Run useful kernelmode payload

Layer 1: AV

- “AV” == scan for signatures, in usermode
  - So, not much relevant to the topic of kernel vulnerabilities
- It just does not work for 0days
- Straightforward to remove offending patterns from the code; e.g. do not store cleartext metasploit shellcode in the binary, encode it

AV


Antivirus scan for ae
https://www.virustotal.com/en/file/7133d10ef39a877642fd9ddda757643e181737e9ed5649bafcd61dca002cc3
Community
Statistics
Documentation
FAQ
About
English
Join our community
Sign in
virustotal
SHA256: 7133d10ef39a877642fd9ddda757643e181737e9ed5649bafcd61dca002cc3c4
File name: epathcl.exe
Detection ratio: 0 / 50
Analysis date: 2014-04-15 12:17:25 UTC (1 minute ago)
0
0
Analysis
File detail
Additional information
Comments
Votes
Behavioural information
Antivirus
Result
Update
AVG
20140415
Ad-Aware
20140415
AegisLab
20140415
Agnitum
20140415

Layer 2: EMET

- Achieve code execution in usermode app (e.g. browser)
- Run kernel exploit code
- Run useful kernelmode payload

EMET

- EMET is basically a set of heuristics meant to catch common usermode shellcode behavior
  - Thou shalt not VirtualProtect +x the stack
  - So, not much relevant to the topic of kernel vulnerabilities
- User mode bypass – see: “Bypassing EMET 4.1”

http://labs.bromium.com/2014/02/24/bypassing-emet-4-1/

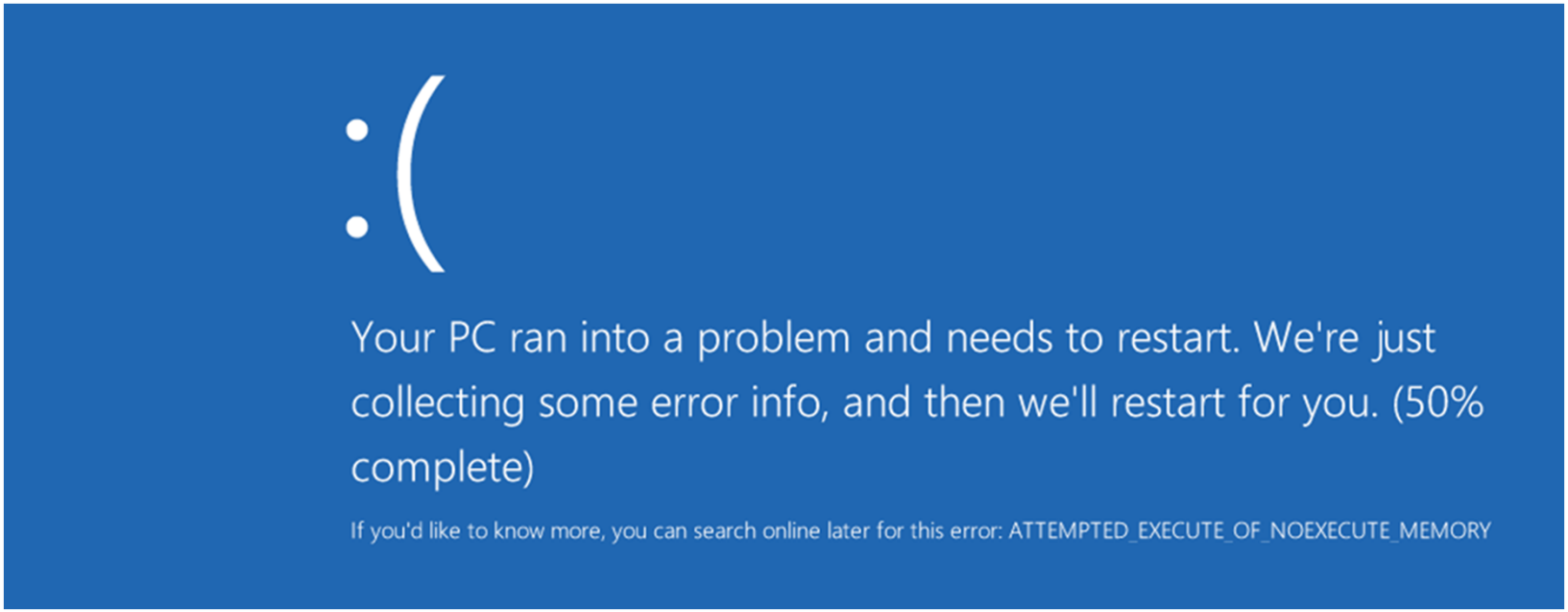
Layer 3: SMEP

- Achieve code execution in usermode app (e.g. browser)
- Run kernel exploit code
- Run useful kernelmode payload

CVE-2013-3660 (EPATHOBJ) and SMEP

- Vulnerability primitive - overwrite arbitrary memory location (in the kernel) with an address of a kernel buffer
- Tavis's PoC overwrites a kernel code pointer (in nt!HalDispatchTable) with an address of a trampoline in kernel memory that jumps into usermode payload - SMEP catches it

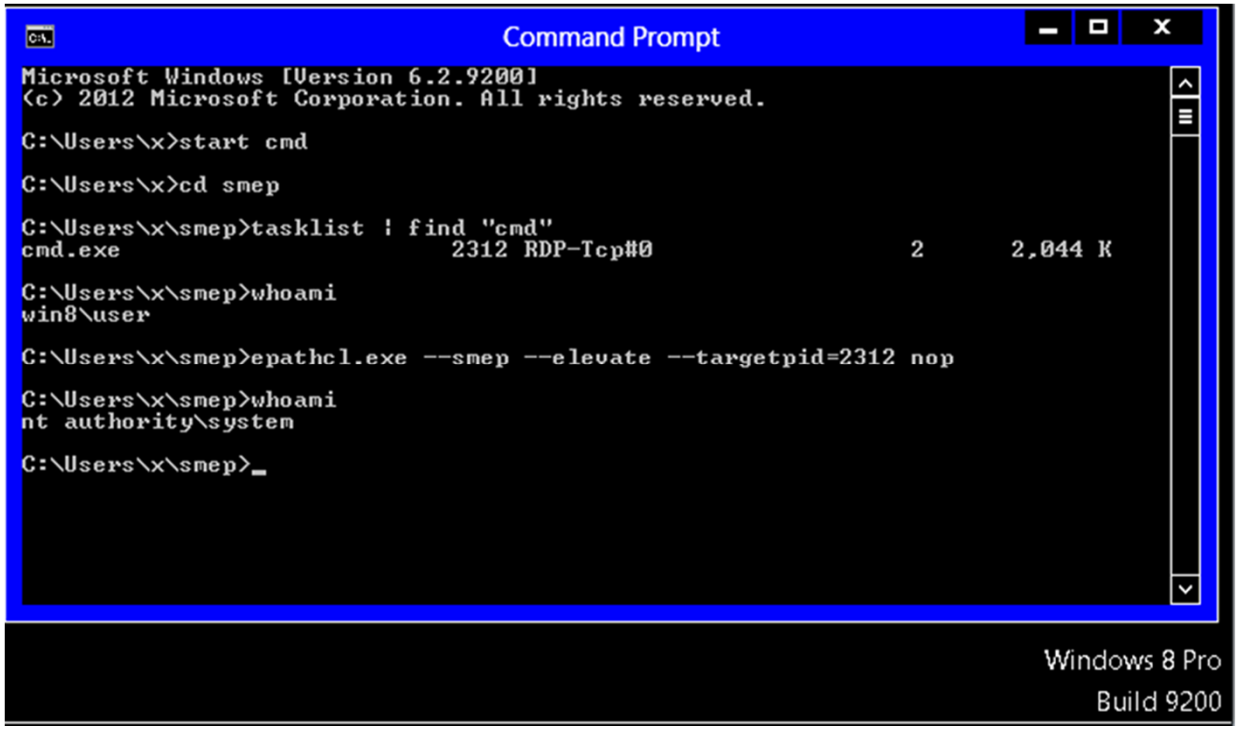
SMEP bugcheck on Windows 8



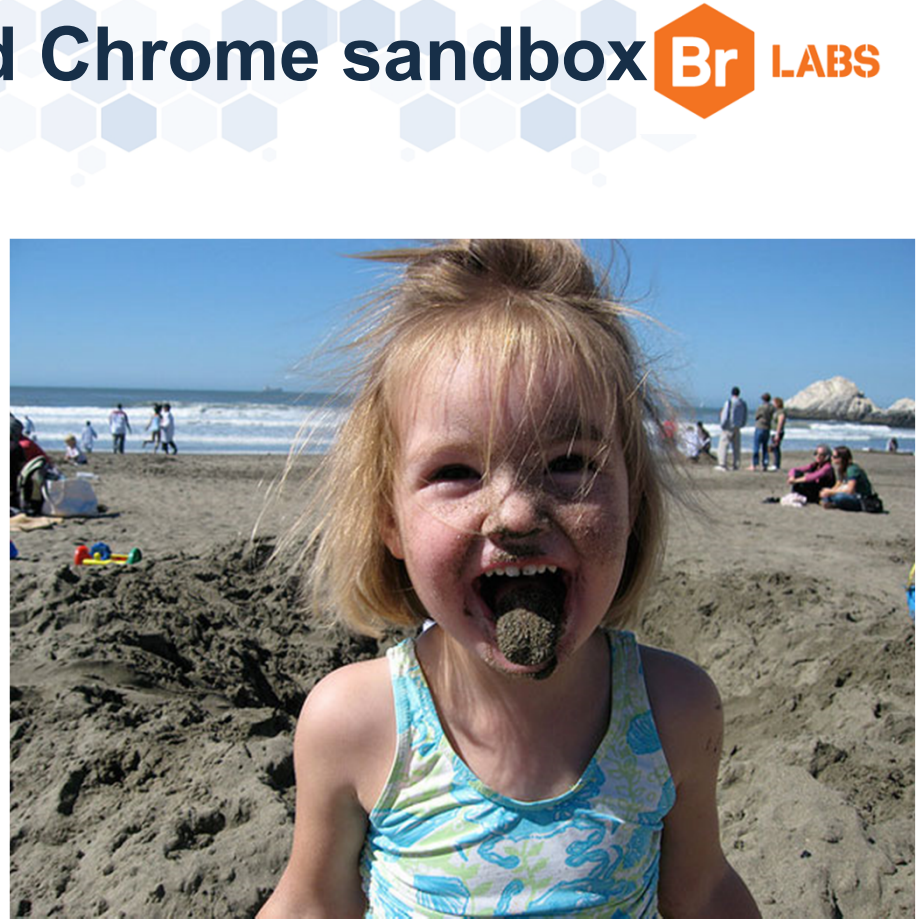
CVE-2013-3660 and SMEP

- Exploit the vulnerability to overwrite nt!MmUserProbeAddress
  - Results in ability to overwrite writable kernel locations via e.g. ReadFile(pipeHandle, kernel_address)
- set U/S bit in the page table entry for an usermode address X
- overwrite nt!HalDispatchTable with X

SMEP-bypassing exploit on Windows 8



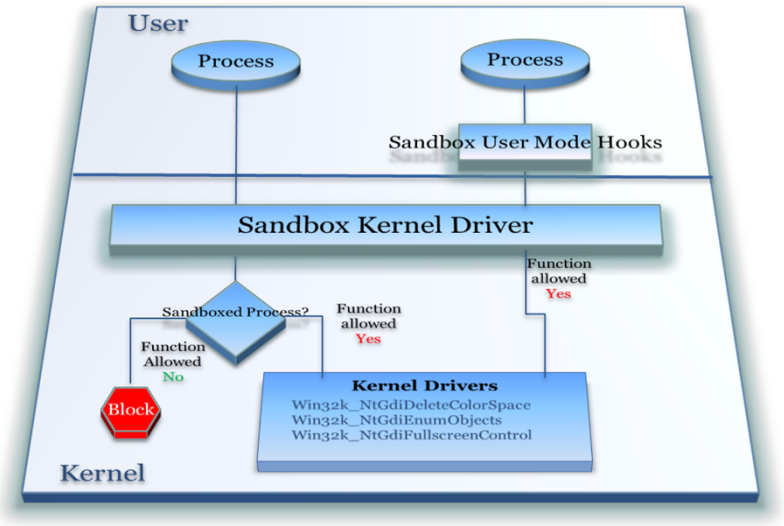
Layer 4: Sandboxie4 and Chrome sandbox



- In both cases, isolation is implemented using usual OS security mechanisms – so normal token stealing kernel payload is all attacker needs from kernelmode
- Admittedly, Chrome sandbox limits the number of usable exploits (but win32k.sys ones are exploitable)

Sandboxie3

- Achieve code execution in usermode app (e.g. browser)
- Run kernel exploit code
- Run useful kernelmode payload

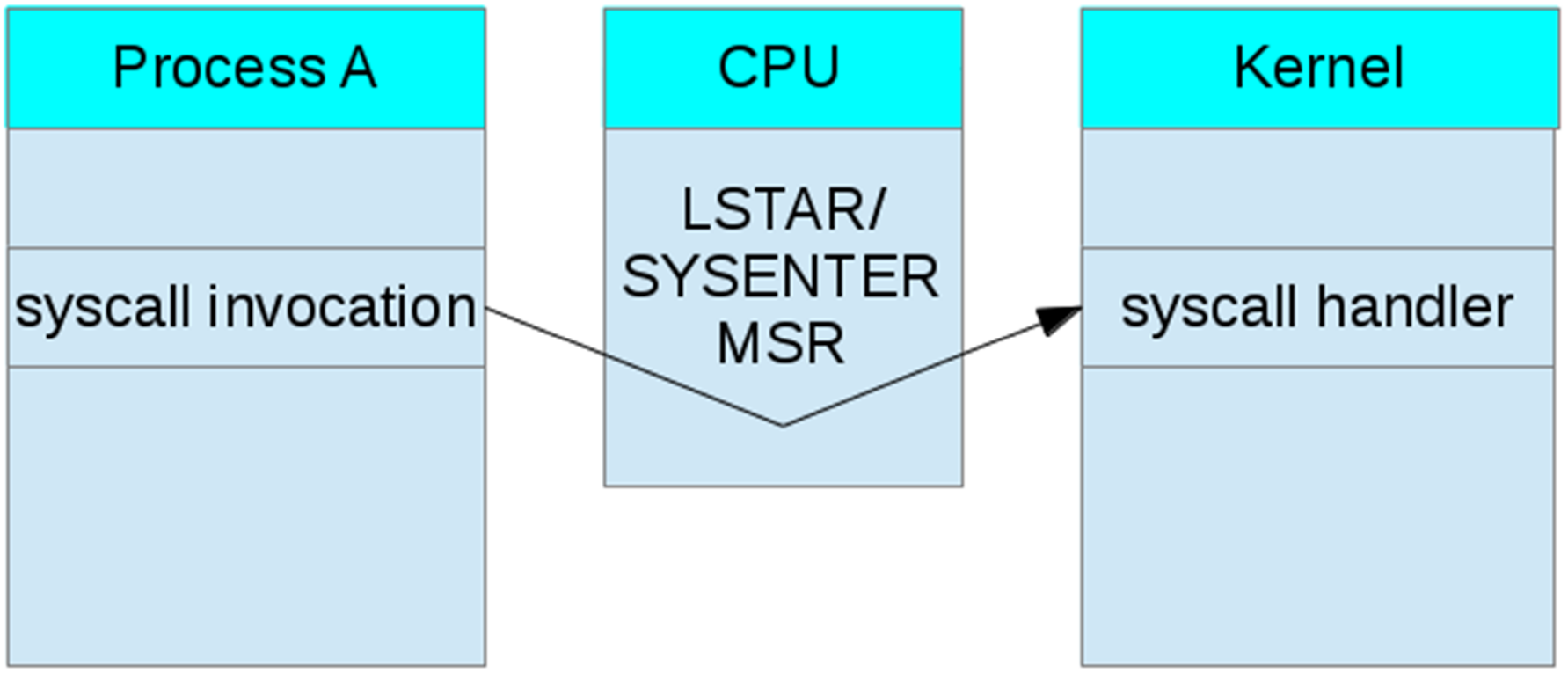


Layer 5: Host IPS

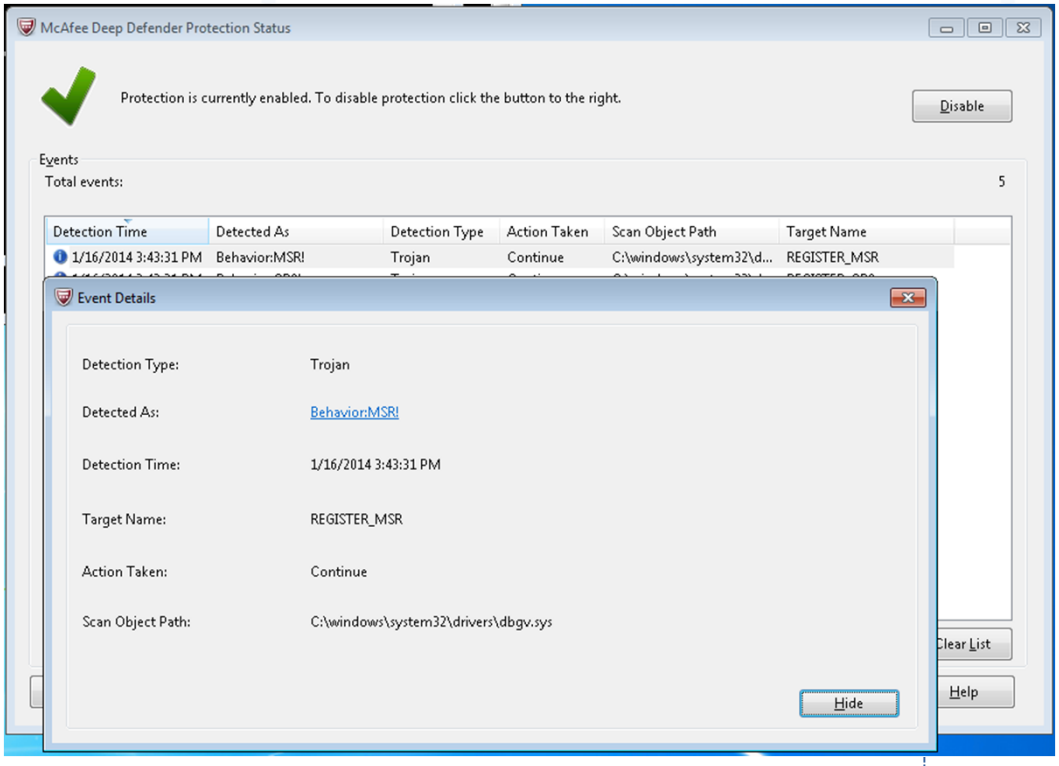
HIPS/Sandboxie3
Usermode code injection from kernel

- ... directly beats the process restrictions
- Generally, if there is a resource X available only for a process Y, then we can grab X by injecting code into Y
- Quite a few methods
  - KeInsertQueueApc() - uses many kernel API, if any is hooked/monitored by HIPS/sandbox, we lose
  - syscall/sysenter MSR overwrite - no kernel API used

How syscalls are dispatched



Layer 6: Deepsafe + MSR overwrite



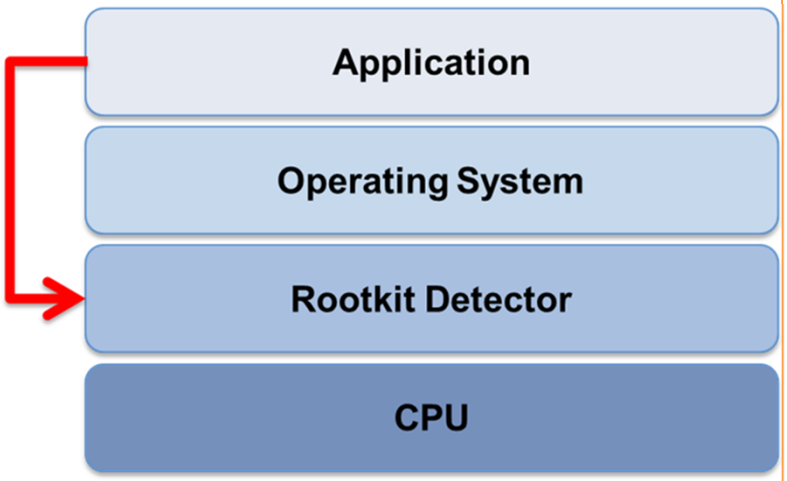
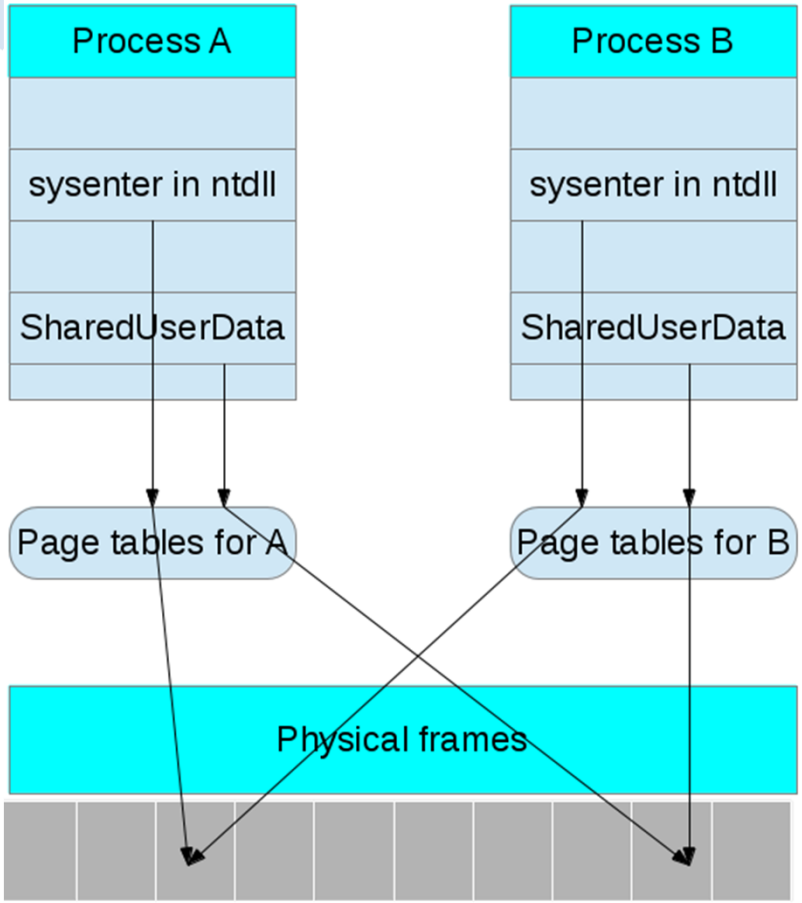


Fig: Exploit triggers MSR alert

Deepsafe continued

- Deepsafe has also the ability to detect the attempt to clear CR4.SMEP – so an exploit should not attempt to bypass SMEP via CR4.SMEP-clearing trampoline
- However, Deepsafe (at least currently) does not detect the mere escalation to kernelmode – the kernelmode payload just needs to be careful to not behave in a way that is covered by Deepsafe detection methods

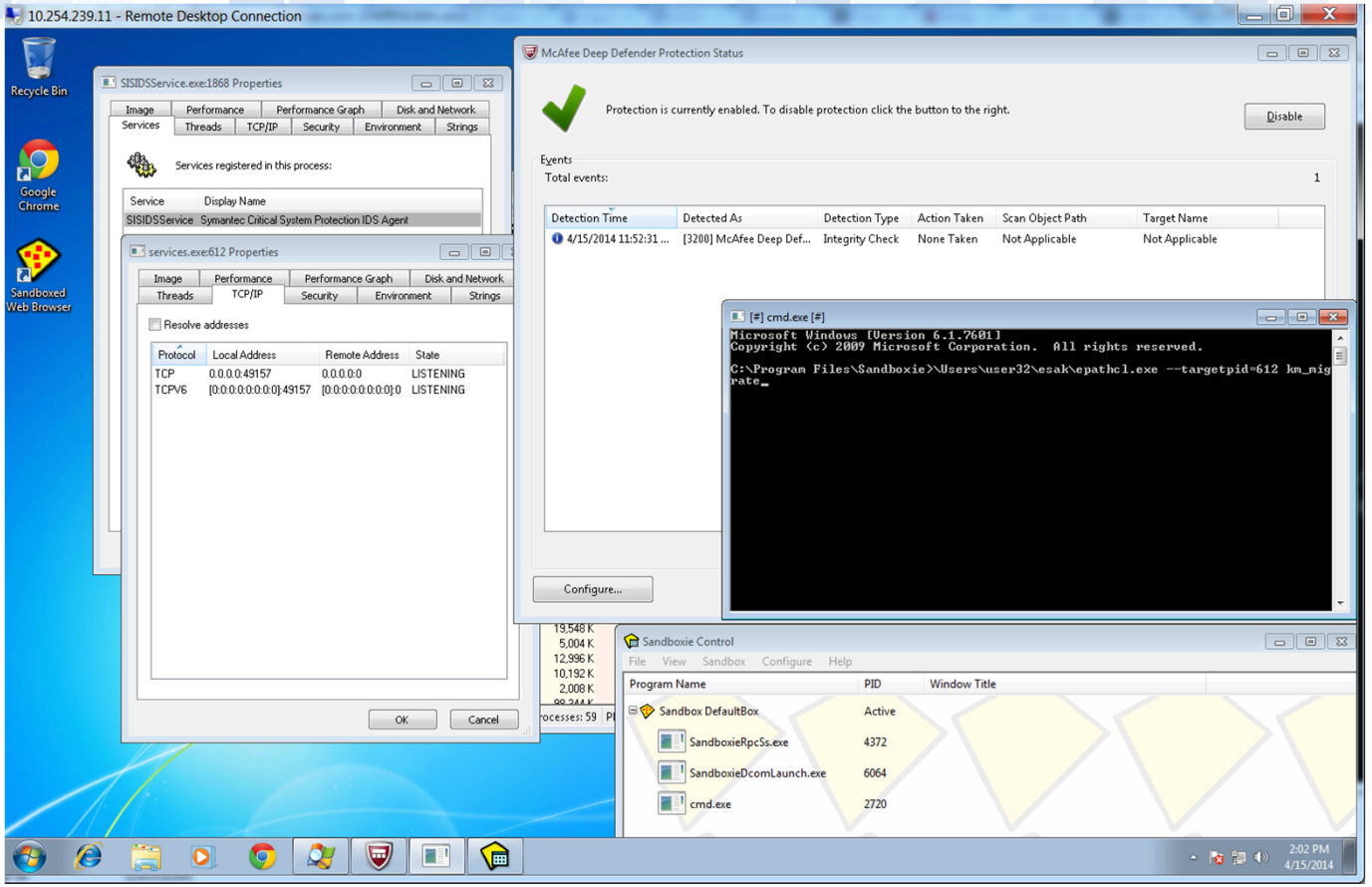
Backing phys frame overwrite



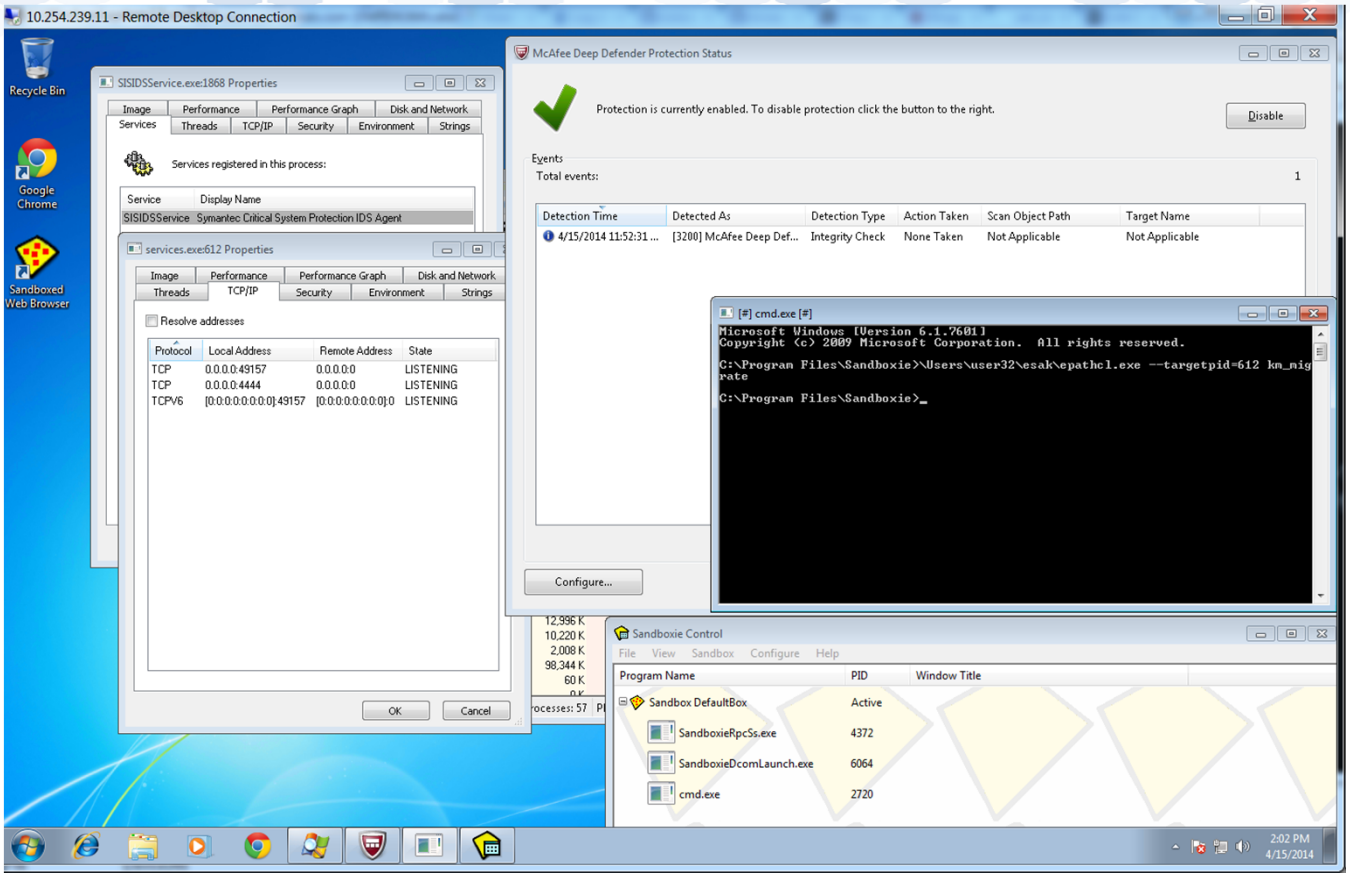
Backing phys frame overwrite cntd

- How to overwrite the backing frame?
  - Just set R/W bit in any PTE for the frame
  - Again, PTE is mapped at the known location
- Ability to inject hook in all processes, no kernel API used
- What to overwrite ?
  - SharedUserData, actual syscall invocation in ntdll
- Where to place the hook code?
  - Unused end of page in any library

Find Waldo, picture 1



Find Waldo, picture 2



WTF??LOL

Conclusion

- Kernel cannot protect against itself
- A reliable kernel exploit can lead to a ‘Swiss army knife’ malware
- Despite various layers, most current solutions have architectural deficiencies to defend against such attacks
- A robust abstraction layer like VMM raises the bar significantly to defend against such attacks

References

- [1] Michael Vincent and Abhishek Singh, „How Advanced Malware Bypasses Process Monitoring”, http://www.fireeye.com/blog/technical/malware-research/2012/06/bypassing-process-monitoring.html
- [2] Jared Demott, „Bypassing EMET 4.1”, http://bromiumlabs.files.wordpress.com/2014/02/bypassing-emet-4-1.pdf
- [3] Mateusz ‘j00ru’ Jurczyk & Gynvael Coldwind, „SMEP: What is it, and how to beat it on Windows”, http://j00ru.vexillium.org/?p=783
- [4] bugcheck & skape, „Kernel-mode Payloads on Windows”, http://uninformed.org/index.cgi?v=3&a=4
- [5] https://labs.mwrinfosecurity.com/blog/2013/03/06/pwn2own-at-cansecwest-2013/
- [6] https://labs.mwrinfosecurity.com/blog/2013/09/06/mwr-labs-pwn2own-2013-write-up---kernel-exploit/
- [7] http://nakedsecurity.sophos.com/2014/02/27/notorious-gameover-malware-gets-itself-a-kernel-mode-rootkit/
- [8] http://www.welivesecurity.com/2012/12/27/win32gapz-steps-of-evolution/
- [9] http://www.fireeye.com/blog/technical/cyber-exploits/2013/12/cve-2013-33465065-technical-analysis.html
- [10] http://labs.bromium.com/2013/10/22/the-latest-tdl4-and-cve-2013-3660-exploit-enhancements
- [11] http://www.mcafee.com/us/resources/reports/rp-defeating-patchguard.pdf

Q&A
Thanks!

@rckashyap

http://labs.bromium.com/