Rootkits vs. Stealth by Design Malware

Joanna Rutkowska
invisiblethings.org

Black Hat Europe 2006, Amsterdam, March 2nd 2006.

What this talk is going to be about?

- Showing demos of new malware which is Stealth by Design (= no classic rootkit technology used, but still fully stealthy),
- Classifying existing rootkit-like malware and discussing how current anti-rootkit technology works against them,
- Introducing the need for Explicit Compromise Detection (ECD),
- Talking about how difficult is to implement ECD on a Windows box and <u>why MS should help us</u>…

Simple definitions...

- Backdoors – give remote access to the compromised machine (smarter ones typically use covert channels),
- Localstuff – key loggers, web password sniffers, DDoS agents, Desktop camera, eject, etc… (can be more or less fun),
- Rootkits – protects backdoors and localstuff from detection.

- I use the term MALWARE as a common term to describe all backdoors, rootkits, viruses,etc…

- Method of infection – exploit, worm, file infector (virus), etc... – not important from our point of view.
- We will see later that rootkits are not necessary to achieve full stealth…

Different approaches to Compromise Detection…

- Look around in the system
  - Process Explorer, netstat, etc… (this can be done automatically by smart HIDS),
  - Don't be tempted to skip this step as it's easy to overlook very simple malware when focused on advanced kernel detection only.
- Cross view based approaches
  - Look for rootkit side-effects,
  - Detect hidden files, registry keys, processes.
- Signature based approaches
  - Scan for known rootkit/backdoor/localstuff engines.

- Check Integrity of Important OS elements
  - Explicit Compromise Detection (ECD)

What does current malware do..

- Wants to survive system restart
  - Files hiding
  - Registry hiding
- Wants to run *arbitrary* process
  - Process hiding
  - Win32 Services hiding
- Hides sockets
- Hides kernel modules and/or DLL modules
- Hides kernel filter drivers

…but we don't need it all! And still can be stealthy!

Surviving the reboot?

- Should malware really care?
  - Forensic experts love to investigate for all in-disk evidences!
- In many companies people do not turn their computers off at night…
  - And even if they do, how much damage can be done when having a backdoor for several hours and not being able to detect it?
  - Servers are very rarely restarted
- But if you’re not convinced…
- And we also have worms…
- And how about BIOS subversion? – see John Heasman’s BH Federal talk.
- And how about advanced file infection (a file which is not digitally signed by MS, but is often executed)

Surviving the reboot

- First, we have worms…
- But if we don't like the idea to relay on some vulnerability, then…
- How about BIOS subversion? – see John Heasman's BH Federal talk
- How about advanced file infection?
  - a file shouldn't be digitally signed but still often executed (forget about all system files)

What about hiding other stuff?

- Process Hiding?
- Win32 Services hiding?
- Sockets hiding?
- Kernel module/DLL hiding?
- Kernel filter drivers hiding?

Stealth malware without rootkits

- We don’t need all those rootkit technologies, but still we’re capable of writing powerful malware!
- Imagine a backdoor which
  - uses covert channel
  - has its own TCP/IP stack implementation
  - has its own implementation of all useful ‘shell’ commands (ls, mkdir, ps, kill, put, get, etc…)
  - has ability to manually create short-life processes (not hidden)
  - Implemented as relocate-able code – no extra module in the kernel.
- No need to hide anything! (process, sockets, modules, services)
- Let’s see the demo now…

DEMO: deepdoor

- Introducing the backdoor
- Showing tcpdump trace from another machine

Video Clip

- Showing no traces in the system log
- Showing no signs of kernel module reminders (modGREPER)

Video Clip

- Showing no hidden processes detected
- Bypassing Personal Firewalls
  - Norton PFW

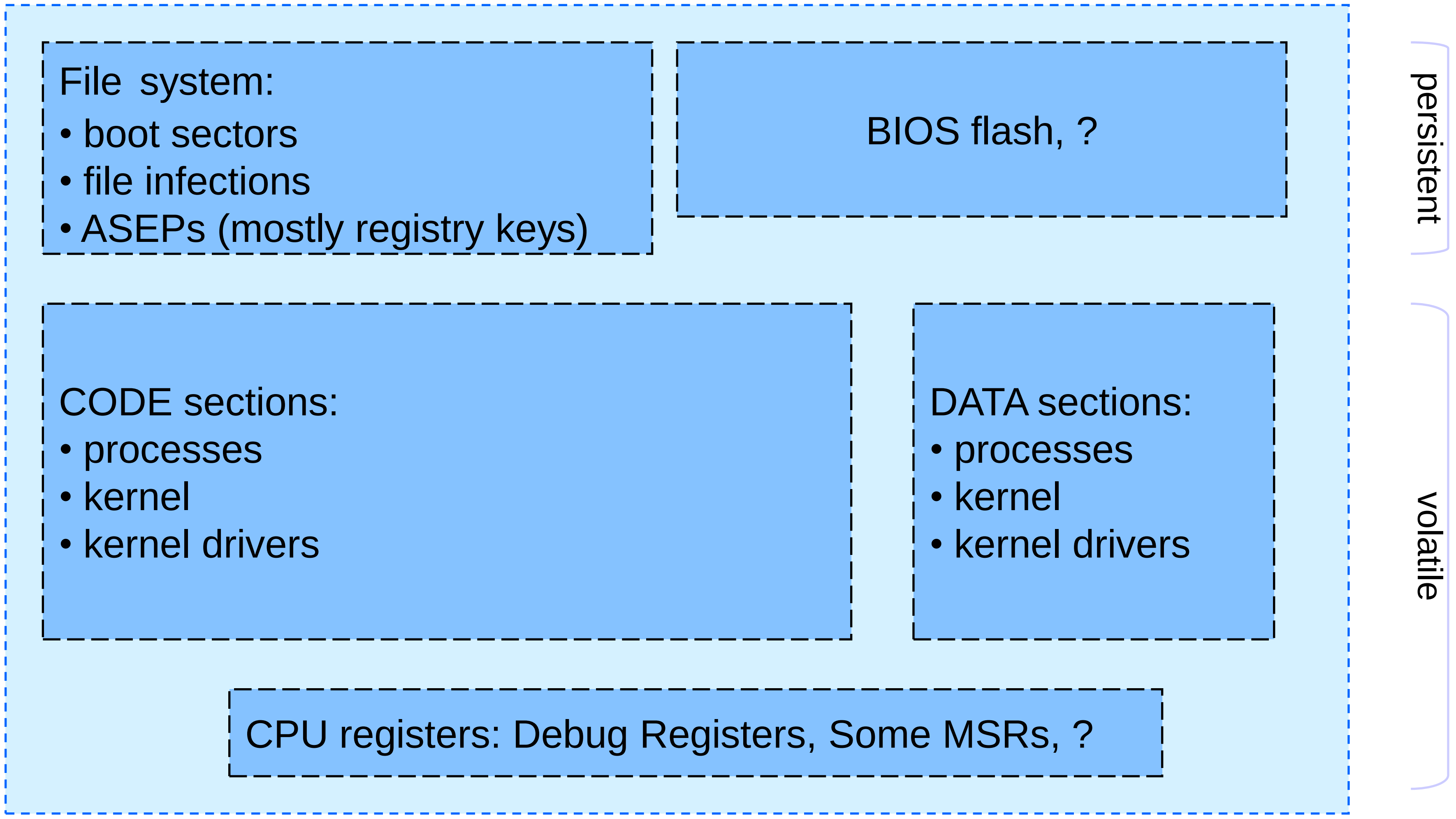
    Video Clip
  - ZA PFW

    Video Clip

Detection

Let’s think about detection of such malware…

Things which can be subverted



File system:
- boot sectors
- file infections
- ASEPs (mostly registry keys)

BIOS flash, ?

CODE sections:
- processes
- kernel
- kernel drivers

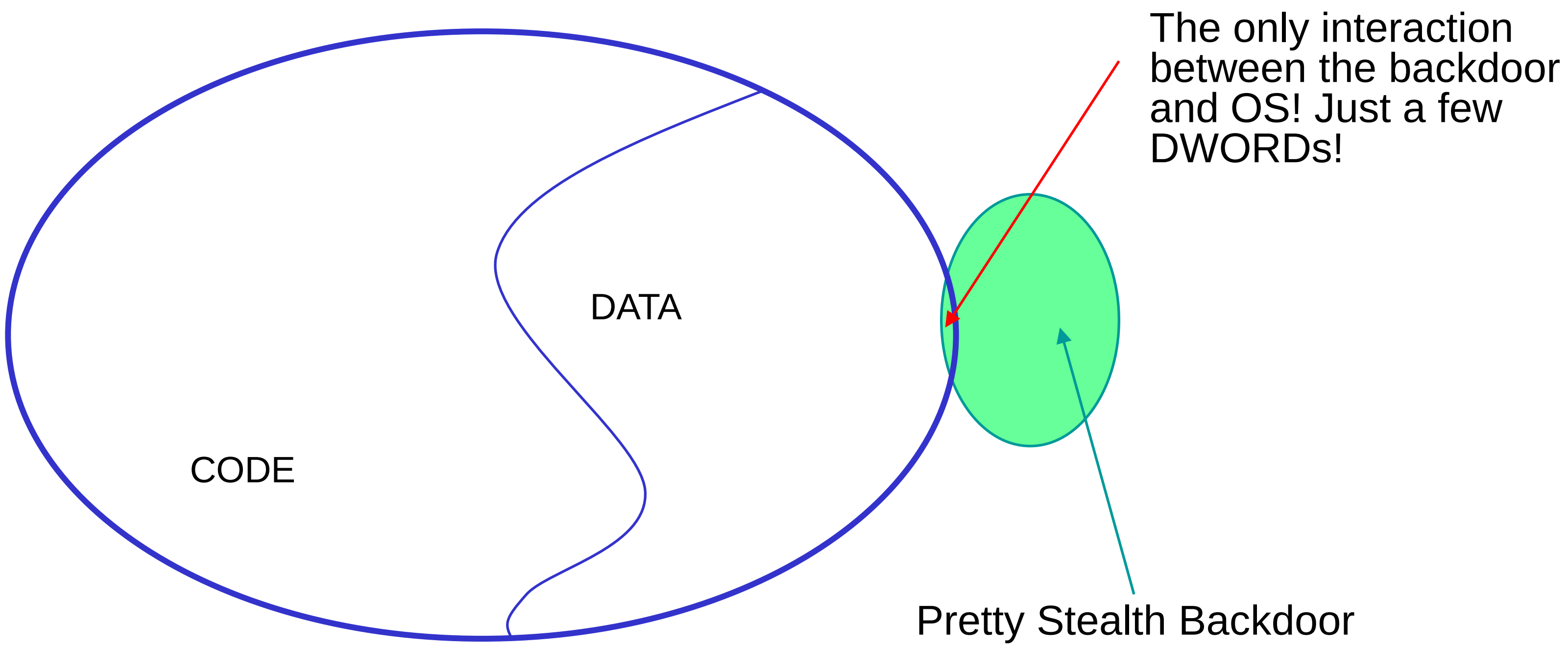
DATA sections:
- processes
- kernel
- kernel drivers

CPU registers: Debug Registers, Some MSRs, ?

Things which can be subverted...

- Persistent storage (file system, etc) subversion is necessary only to reboot survival (nothing more).
- It’s the volatile storage which is crucial to system compromise (we can’t have a backdoor which is not in memory).
  - No matter how the malware is triggered (BIOS, infected files, etc)
- Today many detection tools are focused on file system verification (registry is also file system).

Interaction with OS infrastructure (e.g. deepdoor)


The only interaction between the backdoor and OS! Just a few DWORDs!
DATA
CODE
Pretty Stealth Backdoor

Lessons learned

- Malware doesn’t need to modify code sections (we can always verify code section integrity)
- The real problem is malware which modifies data sections only.
- We saw a backdoor which modified only a few DWORDs somewhere inside NDIS data section!

Malware classification proposal

- Type 0: Malware which doesn't modify OS in any undocumented way nor any other process (non-intrusive),
- Type I: Malware which modifies things which should never be modified (e.g. Kernel code, BIOS which has it's HASH stored in TPM, MSR registers, etc…),
- Type II: Malware which modifies things <u>which are designed to be modified</u> (DATA sections).

- Type 0 is not interesting for us,
- Type I malware is/will always be easy to spot,
- Type II is/will be very hard to find.

Type I Malware examples

- Hacker Defender (and all commercial variations)
- Sony Rootkit
- Apropos
- Adore (although syscall tables is not part of kernel code section, it’s still a thing which should not be modified!)
- Suckit
- Shadow Walker – Sherri Sparks and Jamie Butler
  - Although IDT is not a code section (actually it’s inside an `INIT` section of `ntoskrnl`), it’s still something which is not designed to be modified!
  - However it *may* be possible to convert it into a Type II (which would be very scary)

Fighting Type I malware

- VICE
- SDT Restore
- Virginity Verifier 1.x [see the DEMO later]
- Patch Guard by MS on 64 bit Windows

- Today’s challenge: false positives
- Lots of nasty apps which use tricks which they shouldn’t use (mostly AV products)
- Tomorrow: Patch Guard should solve all those problems with false positives for Type I Malware detection…
- … making Type I Malware detection a piece of cake!

Patch Guard

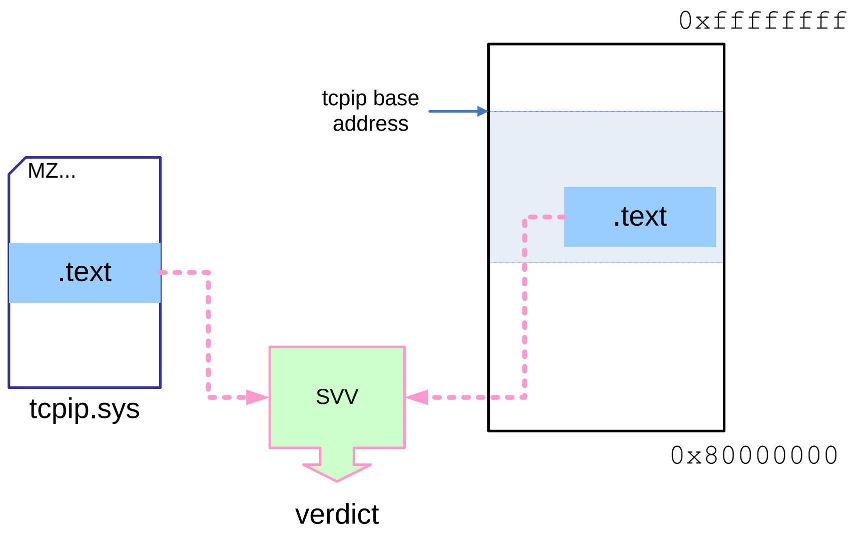
- By Microsoft, to be (is) included in all x64 Windows
  http://www.microsoft.com/whdc/driver/kernel/64bitPatching.mspx
- Actions forbidden:
  - Modifying system service tables
  - Modifying the IDT
  - Modifying the GDT
  - Using kernel stacks that are not allocated by the kernel
  - Patching any part of the kernel (detected on AMD64-based systems only) [*I assume they mean code sections here*]
- Can PG be subverted? Ask Metasploit ;)
- But this is not important!

Patch Guard

- Important thing is: PG should force all the *legal* (innocent) apps not to use all those rootkit-like tricks (which dozens of commercial software use today)
- PG should clear the playground, making it much easier to create tools like SVV in the future,
- It won’t be necessary to implement smart heuristics to distinguish between Personal Firewall-like hooking and rootkit-like hooking.
- It’s unlikely that PG bypassing techniques could be used by serious software companies, because it will give MS the right to treat their products as malware!

System Virginity Verifier Idea

- Code sections are read-only in all modern OSes
- Program should not modify their code!
- Idea: check if code sections of important system DLLs and system drivers (kernel modules) are the same in memory and in the corresponding PE files on disk
  - Don't forget about relocations!
  - Skip `.idata`
  - etc…



Extending SVV – SVV 2.3

- Check not only .text sections, because there are more things which should stay untouched…
- Check all the other code sections (PAGE*, etc…)
  - Problems with checking some exotic sections of ntoskrnl.exe!
- IDT verification (only the most important entries)
- MSR registers (syscall hooking on XP and 2003)
- Addresses bugcheck’s 1-byte attack (http://rootkit.com/blog.php?user=bugcheck)

DEMO: Fighting Type I Malware

- Demo showing SVV2 detecting some malware:
  - Apropos Rootkit
  - AFX2005
  - EEYE BootRoot
  - Shadow Walker 1 (special thanks to Jamie for sending me SW1 code!)
- Demo showing how SVV2 handles potential false positives introduced by software like Personal Firewall, etc…
- Demo showing that sometimes it's virtually impossible to distinguish between PF and a rootkit-like hooking

Type II Malware examples

- NDIS Network backdoor in NTRootkit by Greg Hoglund (however easy to spot because adds own NDIS protocol)
- Klog by Sherri Sparks – “polite” IRP hooking of keyboard driver, appears in DeviceTree (but you need to know where to look)
- He4Hook (only some versions) – Raw IRP hooking on fs driver
- prrf by palmers (Phrack 58!) – Linux procfs smart data manipulation to hide processes (possibility to extend to arbitrary files hiding by hooking VFS data structures)
- FU by Jamie Butler
- PHIDE2 by 90210 – very sophisticated process hider, still however easily detectable with X-VIEW...

Fighting Type II Malware

- There are three issues here:
  - To know where to look
  - To understand what we read
  - To be able to read memory

- But… we all know how to read memory, don't we?
- More on this later, now let's look at some demos…

DEMO: Type II Malware Detection

- Demo showing spotting klog using Device Tree and KD

  Video Clip
- Demo showing he4Hook detection using KD

  Video Clip

Type II Malware Detection cont.

- “To know where to look” issue
- On the previous demo, we somehow knew where to look…
- …but there is lots of data inside the OS…
- …how to make sure that we check all the potential places?

Challenge for Type II malware detection

- Create a list of where should we look (NDIS data structures, device IRPs, attached filters, …
- What else? Is the list finite?
- OMCD project
  - Open Methodology for Compromise Detection
  - http://isecom.org/omcd/
- But do we really need *Open* Methodology? Should such a project be public?

SbD malware detection

- SbD malware detection vs. classic rootkit technology detection
- Problem: there are no hidden objects to detect!

ECD vs. ICD…

- ICD (Implicit Compromise Detection)
  - looks for hidden objects
  - Also known as Cross view diff
  - e.g. Rootkit Revealer, Black Light, etc…
  - Useless against SbD malware
- ECD (Explicit Compromise Detection):
  - looks what was subverted in the system by checking the integrity of all important OS elements.
  - Requires ability to read kernel memory (see later)
  - Is more systematic approach to compromise detection – we try to see if all is intact, rather then use tricks to find hidden objects.
  - e.g. SVV, MS Patch Guard, VICE

Memory Reading Problem (MRP)

- Problem: we can't safely read Windows kernel memory
- What about those popular functions:
  - `__try/__except` – will not protect from BugChek 0x50
  - `MmIsAddressValid()` – will introduce a race condition (and we also won't be able to access swapped memory)
  - `MmProbeAndLockPages()` – may crash the system for various reasons, TLB corruption being one of them!
- The truth is: We can't read <u>arbitrary Windows kernel</u> memory without the risk of crashing the system!
- But Why? We're in ring0, we should be able to do everything, right?
- If it's such a problem to read kernel memory, how is it possible that all those Windows machines work?!

MRP cont.

- The problem is not what can we physically do, but rather what we can do from the “protocol point of view”,
- And kernel was not designed to allow 3rd parties to read memory areas which belong to somebody else (reading NDIS data structure by somebody who is not NDIS itself),
- 3rd party reading memory, which it doesn’t own, may be subject to various race conditions or cause TLB corruption,
- So, before we try to read something we really need to think it over to see if we really can safely read it!
- It seems that Microsoft's help is very necessary here.

MRP cont.

- What about \Device\PhysicalMemory?
  - Removed in w2k3 SP1 (due to TLB corruption problem, btw!)
  - Even if it was safe to use it (and was not removed) it still gives access only to physical memory and we need to read (check integrity) of also swapped memory.
- How about freezing a system?
  - Increase IRQL on all processor and dump memory?
  - Again – only physical memory available

MRP – what Microsoft can do?

- It’s a hard problem – no easy solution exists.
- MS should put some effort into building an infrastructure which would allow 3rd party tools for kernel memory verification/scanning.
- This infrastructure should be easy to verify (e.g. check if it hasn’t been already hooked)
- This “infrastructure” doesn’t have to be an API, it can also be a set of guidelines regarding how to properly synchronize with the Memory Manager and read the memory…

Stealth by Design vs. Type II Malware

- "Stealth by Design" is not the same as "Type II"!
  - Some process hiders (e.g. FU, PHIDE2) are type II but not SbD!
  - The same for some files hiders (e.g. he4hook)
- Some Type I malware is SbD, but only Type I:
  - Eeye bootroot NDIS backdoor

- SbD is about not hiding anything – avoiding cross view detection by design.
- Type II is about making ECD detection difficult.

Stealth by Design vs. Type II Malware Detection

	Type I Malware	Type II Malware
Classic Rootkit Technology	ECD easy and effective. X-VIEW works well too.	ECD may be difficult X-VIEW easier and more effective.
Stealth By Design	X-VIEW useless. ECD easy and effective.	X-VIEW useless. ECD may be difficult. Network based detection may be easier?

- ECD = Explicit Compromise Detection
- X-VIEW = Cross View Based Detection

DEMO: deepdoor again

- Showing that it’s a type II backdoor…
  - Code verification
  - SDT verification
  - IDT verification
  - NDIS protocols (btw, not a strict Type II requirement)
- We’ve already seen it’s a Stealth by Design malware…
- So where is the backdoor?
  - touching the backdoor (using KD)…
  - Having seen this, we still cannot come up with a detection tool, mostly because of the MRP!
  - We cannot also use PFW for preventing this backdoor, as this is “the last one wins” game (not “the first one wins”!)
  - We have seen only few DWORDs of the backdoor, where is the rest? Even if we knew this is not a good method for detection (polymorphism, etc).

Challenge

- Maybe we shouldn’t worry about advancement in malware technology?
- Commercial Hacker Defender shows another trend:
- Implement lots of Simple and Stupid Implementation Specific Attacks (ISA) against all the tools on the market…

- So, all commercial AV products are ineffective against custom malware (which one can buy for $$$),
- Most of that “commercial malware” is detectable by private detectors (which one can buy for $$$$-$$$$$),
- Private detectors can’t cost too little!

What OS vendors can do?

- Make it possible to reliably read kernel memory
  - We (ISVs) cannot do much when we're blind!
  - IsSystemInfected() API is *really* not a good idea!
- Design system in such a way that the crucial parts are easily verifiable:
  - Export symbols like
    - `IDT` (helps to verify IDT integrity)
    - `KiServiceTable`, (SDT integrity)
    - `KiFastCallEntry (MSR_SYSENTER` verification)
  - This will help ISVs with writing system integrity checkers
  - This will *not* make creating rootkits easier, as rootkit authors already know how to find IDT and Service Table and all the other interesting stuff!
- Exploiting hardware to verify kernel memory integrity may be a good idea (TPM?)

Losers and Winners

- Mr. and Mrs. Smith always lose!
- Large companies may win (using private detectors)…
- Authors of ISA-based malware earn money and laugh from AV companies!
- Providers of custom rootkit/compromise detection services laugh from ISA-based malware :)
- AV may (at some point) become providers of those custom detectors for large companies…
- Everybody waits for the next generation OS which will introduce more then two CPU privileges modes (4 years?), hopefully eliminating ISA (but not SbD type II malware…)

Thank you for your time!