Rootkits vs. Stealth by Design Malware

Joanna Rutkowska
invisiblethings.org

Black Hat Europe 2006, Amsterdam, the Netherlands

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.

- 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...

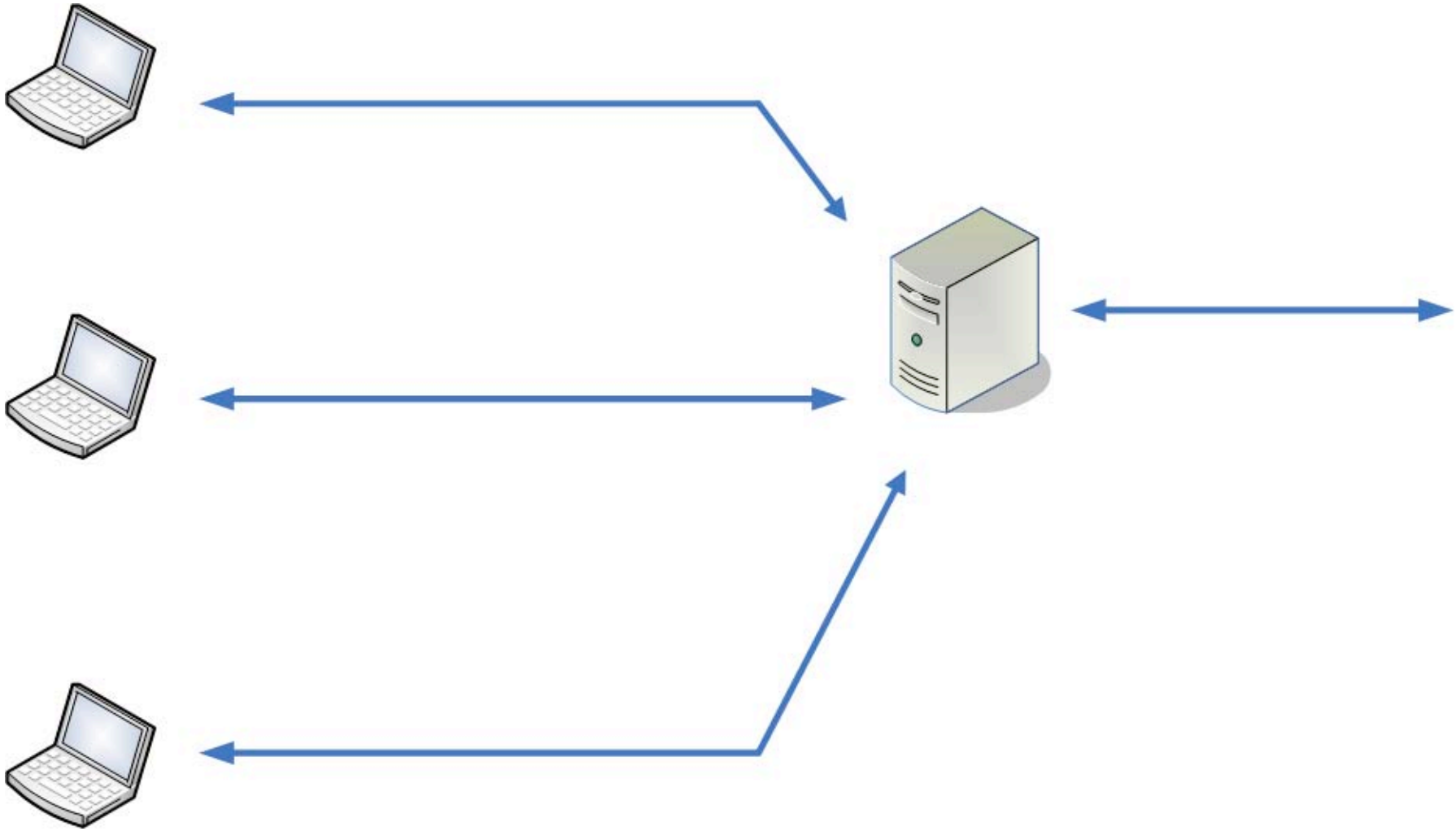
- Look around in the system
  - Process Explorer, netstat, etc… (this can be done automatically by smart HIDS),
  - Don't be tempted to skip this step as it's easy to overlook very simple malware when focused on advanced kernel detection only.
- Cross view based approaches
  - Look for rootkit side-effects,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

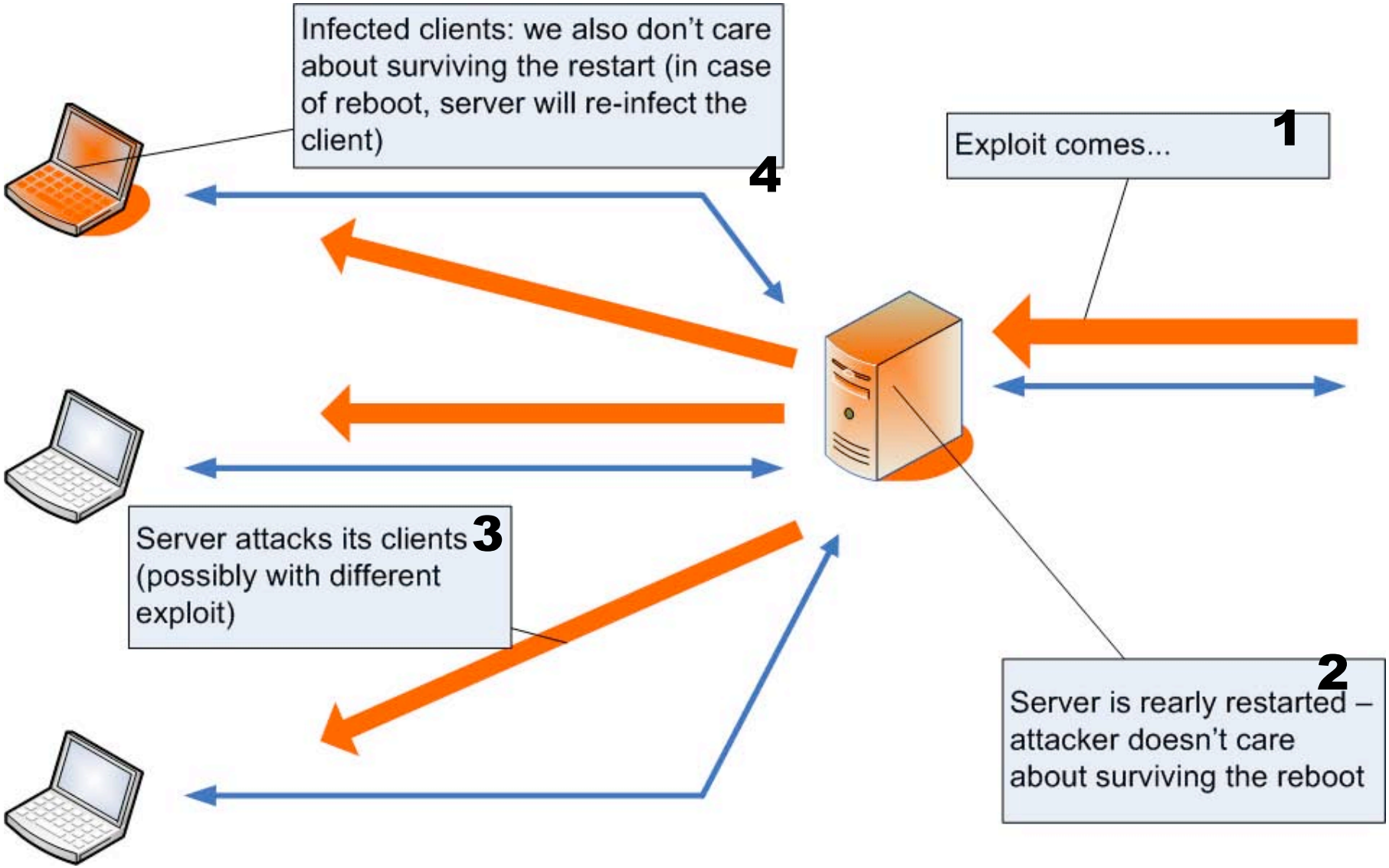
Surviving the reboot?

- Should malware really care?
- In many companies people do not turn their computers off for the night,
- And even if they do, how many damage can be done when having a backdoor for several hours and being unable to detect it?
- Servers are very rarely restarted,
- And also we have worms…

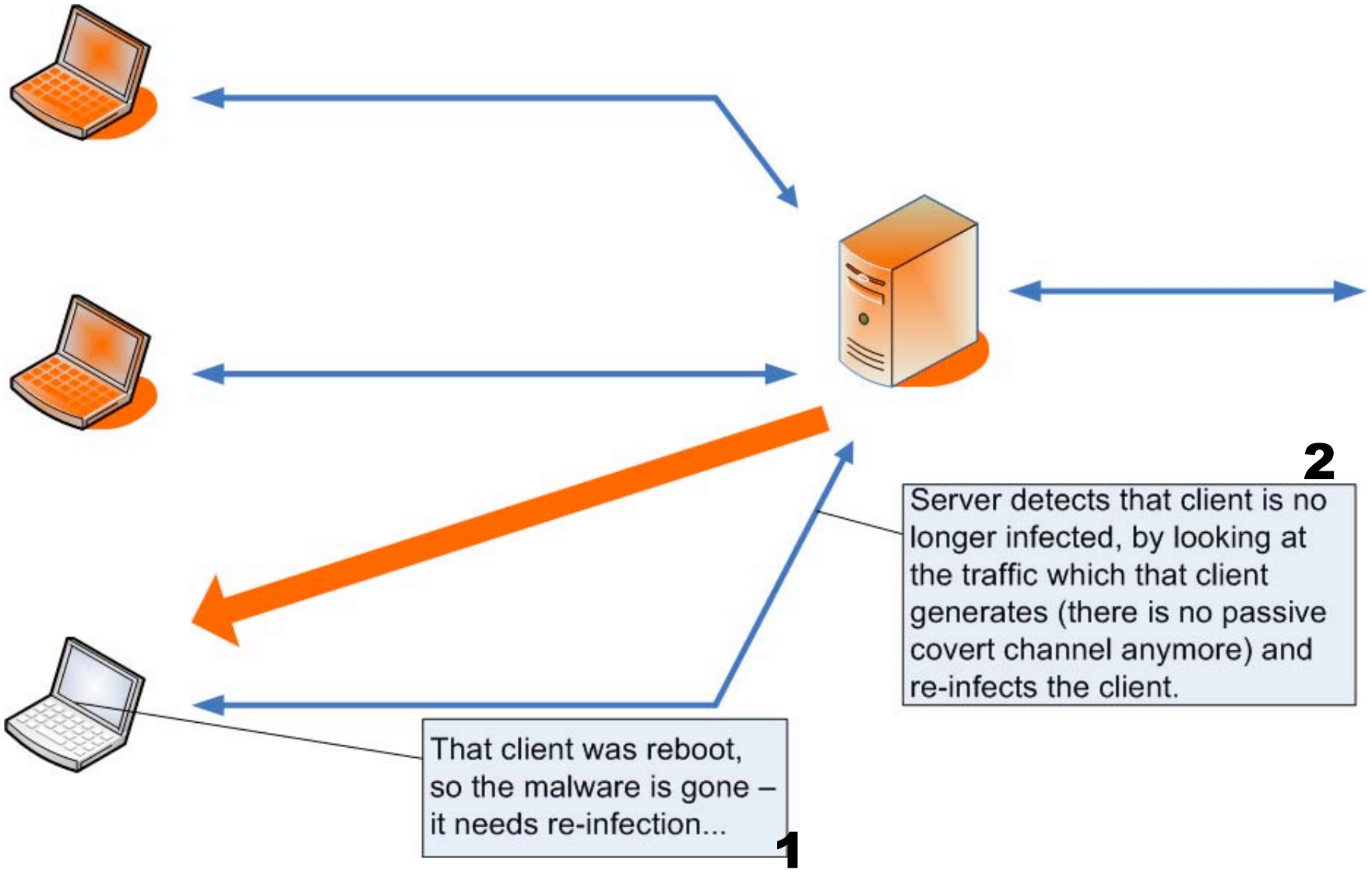
Theoretical Scary Scenario...



Network infected



Client re-infection



Digression: Passive Covert Channels

- Passive Covert Channels idea:

  http://invisiblethings.org/papers/passive-covert-channels-linux.pdf
- NUSHU (passive covert channel POC in TCP ISNs for Linux 2.4 kernels):

  http://invisiblethings.org/tools/nushu.tar.gz
- How to detect NUSHU (and how to improve it so it will not be detectable) by Steven Murdoch et al:

  http://www.cl.cam.ac.uk/users/sjm217/papers/ih05coverttcp.pdf

- Maybe network based detection (not signature based!) is the future?

Surviving the reboot...

- Still unconvinced that we shouldn't care about restart survival?
- Ok, we want to place a trigger somewhere on the file system, but we don't want to be caught by X-VIEW detection (ala RkR or Black Light)…
- Of course it's trivial to cheat those tools (in more or less generic way), but we want a "stealth by design" solution…
- So, why not try using a good polymorphic file infector for this?
  - Mistfall engine is several years old, but still is considered among AV people as one of the most challenging file infector!
  - Should we sleep well and not worry that in the meantime somebody could/can write something better?
  - Watch out for files which are digitally sign (all system binaries)!

What about hiding other stuff?

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

Hidden Processes?

- It's convenient to have a possibility to run (in a stealthy manner) an arbitrary process…
- However, it should be *always* possible to find extra hidden process executing inside OS, as the OS should be aware of this process:
  - scheduler (but look at smart PHIDE2)
  - Object manager
- So, do we really need hidden processes?
- Maybe we can use injected threads into some other processes to do the job? (compile your favorite tools with .reloc sections)
- Or even better – if we have a smart backdoor (e.g. kernel NDIS based) why not build most of the functionality into it? [see the demo later]

Hidden Win32 Services?

- Services are very easily detectable – much easier then just ordinary processes.
- But, if we agreed that we don't need processes then it should be obvious that we don't need services too.

Hidden Sockets?

- That was *always* a very bad idea!
- Hiding something which is still visible from a network point of view is a bad idea.
- Use covert channels (passive if possible)
- If you need to do it in a traditional way, use knock scenario and connect back.

Hidden modules (kernel and DLLs)?

- Very bad idea – very easy to find.
- It's even better not to hide kernel modules at all (just place them in system32\drivers so they look not suspicious)!
- And if one wants the real stealth – why to use modules at all?
- Load, allocate a block of memory, copy and relocate and unload the original module (no traces left in kernel).
- Or do the same when exploiting kernel bug.

- Related thing: resistance to signature based scanners
  - Shadow Walker,
  - Cut and Mouse (detect when somebody starts reading memory near you and relocate),
  - How to do it without touching IDT?

Hidden kernel filters?

- People use them usually to:
  - hide files (but not registry)
  - hide sockets
  - Implement simple network backdoors
  - install key loggers
- We don’t need them!
- No need to bother to hide them.

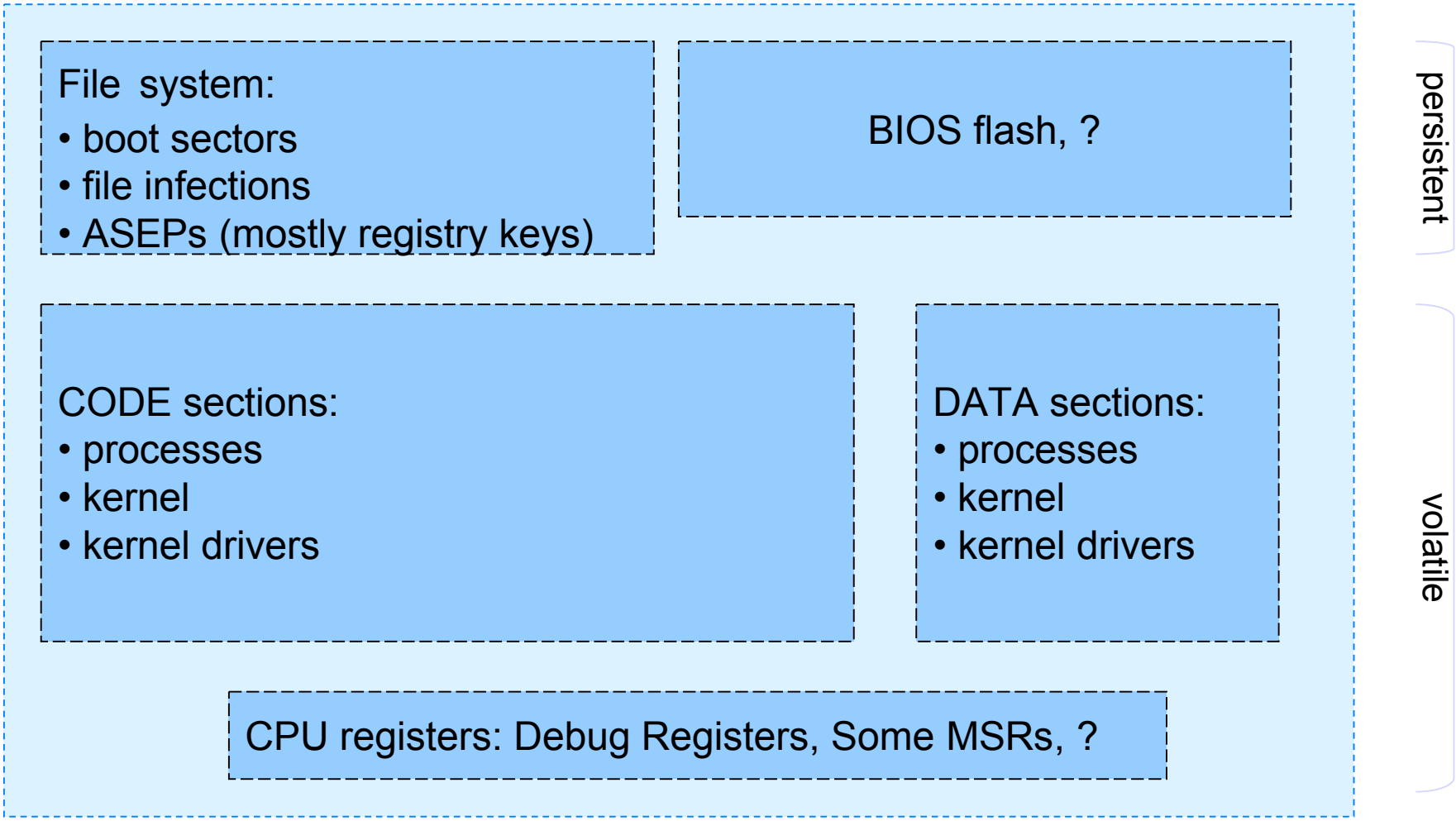
Stealth malware without rootkits

- 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 module in kernel.
- No need to hide anything! (process, sockets, modules, services)
- Let’s see the demo now…

DEMO: Pretty Stealth Backdoor

- Introducing backdoor
- Showing no traces in the system log
- Showing no signs of kernel module reminders (modGREPER)
- Showing no hidden process detected
- Showing tcpdump trace from another machine
- Bypassing Personal Firewalls

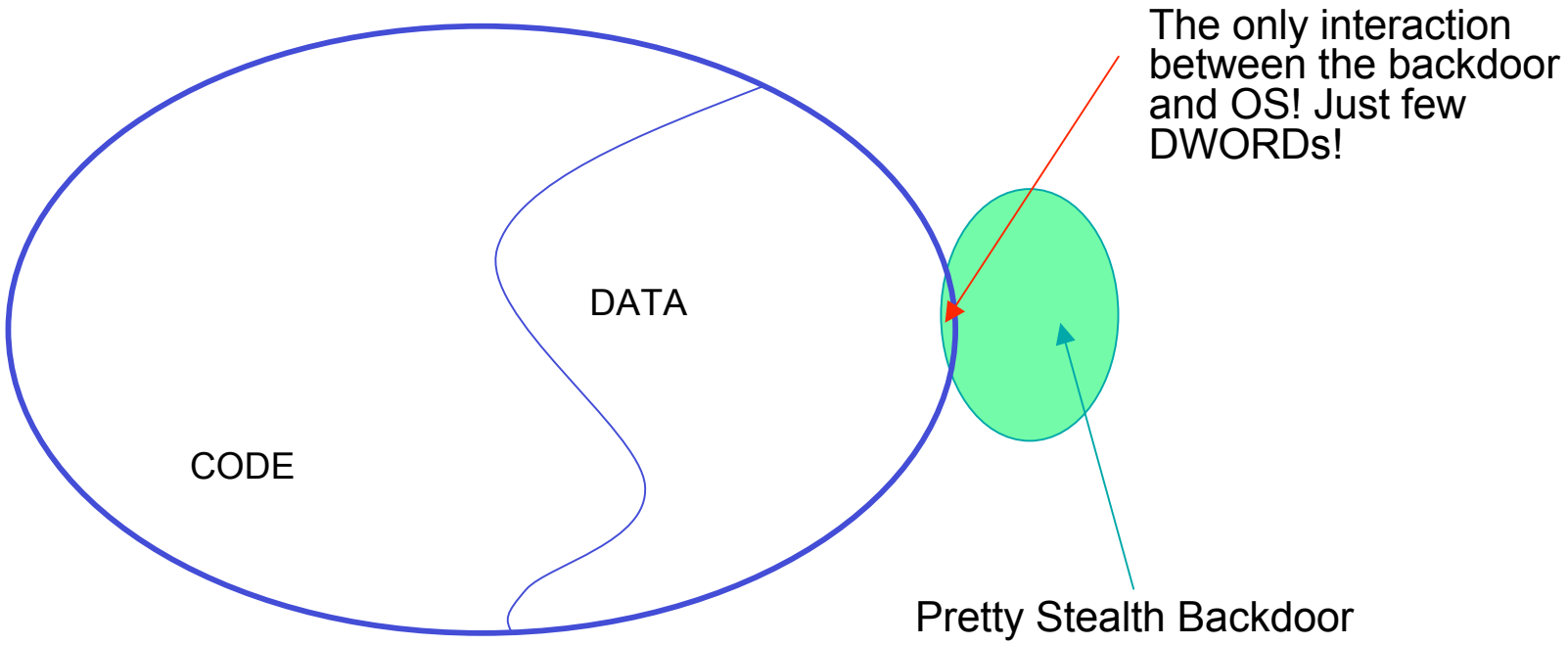
Things which can be subverted



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).
- Today many detection tools are focused on file system verification (registry is also file system).

Interaction with OS infrastructure



Lessons learnt

- Malware doesn't need to modify code sections (we can always verify code section integrity)
- The real problem is malware which modifies only data sections.
- We saw a backdoor which modified only 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 which are designed to be modified (DATA sections).

- Type 0 is not interesting for us,
- Type I malware is/will always be easy to spotted,
- 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 – Sheri Sparks and Jamie Butler
  - Although IDT is not a code section, 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

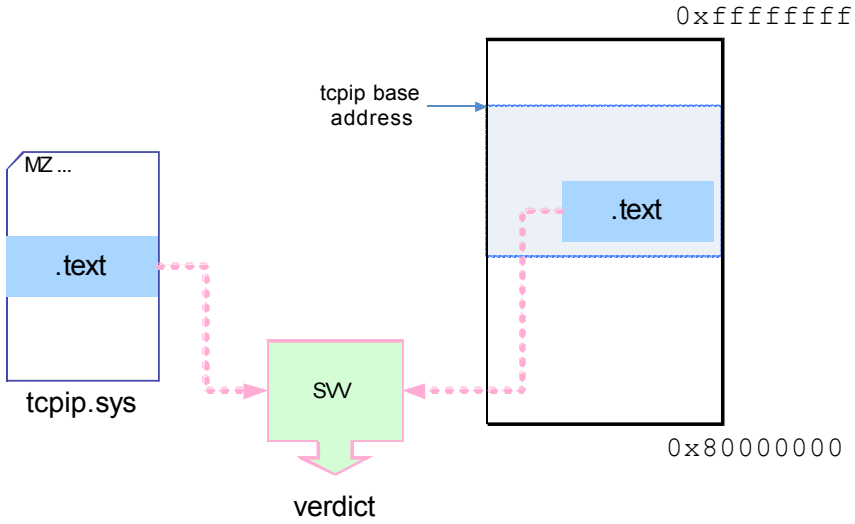
- By Microsoft, to be (is) included in all x64 Windows
  http://www.microsoft.com/whdc/driver/kernel/64bitPatching.mspx
- Actions forbidden:
  - Modifying system service tables
  - Modifying the IDT
  - Modifying the GDT
  - Using kernel stacks that are not allocated by the kernel
  - Patching any part of the kernel (detected on AMD64-based systems only) [*I assume they mean code sections here*]
- Can PG be subverted? Almost for sure.
- 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.
- So, even if we see a POC for bypassing PG (I'm pretty sure we will see sooner or later) in the future, it will not make PG useless…
- It will only proof my statement that it's good to have several detection tools (from different vendors preferably)

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 1.x

- Check not only CODE sections, because there are more things which should stay untouched…
- IDT
- MSR registers
- Debug Registers (need tricks to verify them)
- …?

DEMO: Fighting Type I Malware

- Demo showing SVV detecting some malware:
  - Apropos Rootkit
  - EEYE BootRoot
  - HackerDefender
- Demo showing how SVV handles potential false positives introduced by software like Personal Firewall, etc…

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 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?
- Later on this, now let's look at some demo…

DEMO: Type II Malware Detection

- Demo showing spotting klog using Device Tree and KD
- Demo showing he4Hook detection using KD

Type II Malware Detection cont.

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

Memory Reading Problem (MRP)

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

MRP cont.

- It's not the problem of what can we physically do, but rather of what can we do from the "protocol point of view",
- And kernel was not designed to allow 3rd party 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 race conditions or cause TLB corruption,
- So, before we try to read something we really need to think it over if we really can safely read it!
- It seems that Microsoft's help is very necessary here.

Stealth by Design vs. Type II Malware

- “Stealth by Design” != “Type II”
- Lots of Type II malware today is not SbD:
  - All the process hiders (FU, PHIDE2)
  - Files hider (he4hook)
- Some Type I malware is SbD:
  - Eeye bootroot NDIS backdoor

- SbD is about not hiding anything – avoiding cross view detection by design.
- X-VIEW detection is useless when detecting SbD malware.
- Explicit Compromise Detection (ECD) is useful here.

Stealth by Design vs. Type II Malware

- Type II is about implementing malware so that there is no easy way to detect it by performing integrity scan (of filesystem, code sections, etc...)
- Type II is about avoiding ECD.
- Type II challenge: modify those parts of OS, which are hard to verify that were modified!
- X-VIEW may sometimes work.
- SbD Malware which is of Type II may be extremely difficult to detect
  - X-VIEW doesn't work
  - ECD is usually difficult

File infectors...

- Advanced EPO File Infectors are SbD…
- …but if infected file has a digital signature (like all Windows system files), then even the most advanced virus is Type I only!

Stealth by Design vs. Type II Malware

	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: Pretty Stealth Backdoor Again

- Showing that it's Type II backdoor
  - Code verification
  - SDT verification
  - IDT verification
  - IRP verification
  - NDIS protocols (btw, not a strict Type II requirement)
- We already saw it's Stealth by Design…
- So where is the backdoor?

Challenge

- 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?

- But on the other hand…

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 (SaSISA) 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!

Losers and Winners

- Mr. and Mrs. Smith always lose!
- Large companies may win (using private detectors)…
- Authors of SASISA-based malware earn money and laugh from AV companies!
- Providers of custom rootkit/compromise detection services laugh from SASISA-based malware :)
- AV may start become those providers of custom detectors for large companies at some point in the future…
- Everybody waits for the next generation OS which will introduce more then two CPU privileges modes (4 years?), hopefully eliminating SASISA…

Thank you
for your time!