Rootkits:
What they are and how to find them

or

check yo self before you wreck yo self!

Part 1

Xeno Kovah –2010

xkovah at gmail

Ice Cube is a Friendly Rootkit Advocating for Rootkit Detection!

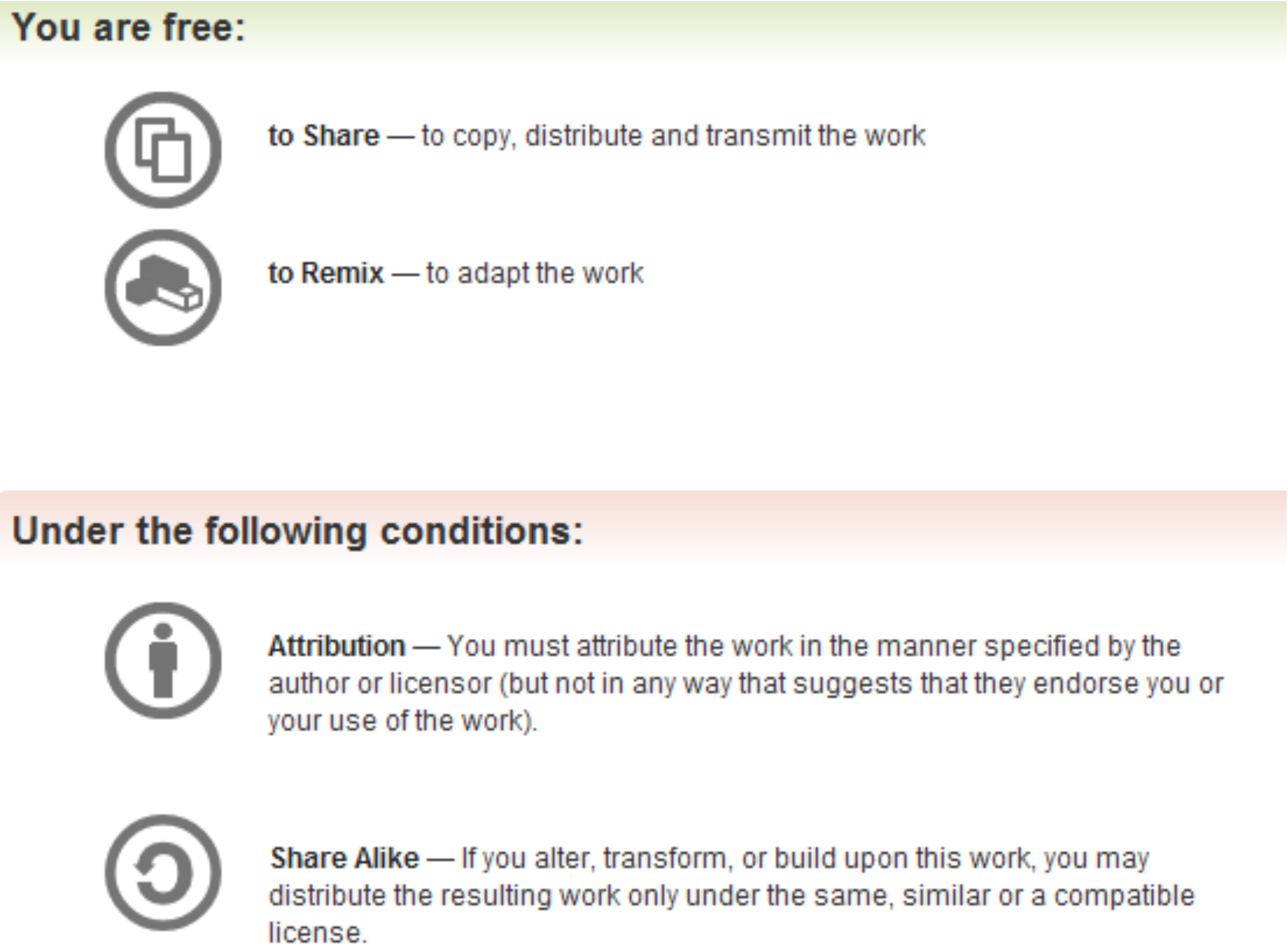
You betta check yo self
fore you wreck yo self
cause I'm bad for your health
I come real stealth

: O

http://www.youtube.com/watch?v=AJR62vsAg-0

All materials is licensed under a Creative Commons “Share Alike” license.

- http://creativecommons.org/licenses/by-sa/3.0/



You are free:

to Share — to copy, distribute and transmit the work

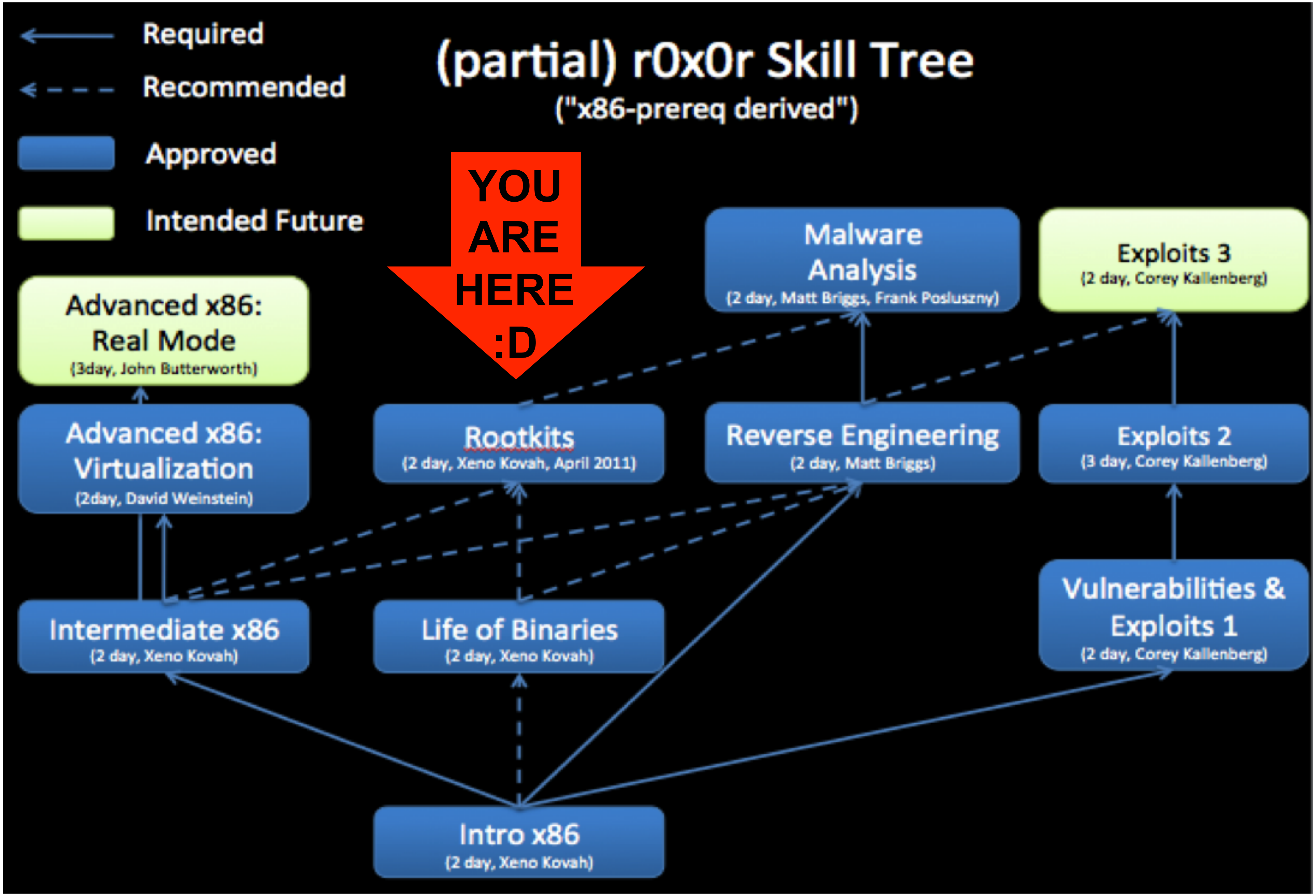
to Remix — to adapt the work

Under the following conditions:

Attribution — You must attribute the work in the manner specified by the author or licensor (but not in any way that suggests that they endorse you or your use of the work).

Share Alike — If you alter, transform, or build upon this work, you may distribute the resulting work only under the same, similar or a compatible license.

May your skill tree overgroweth…



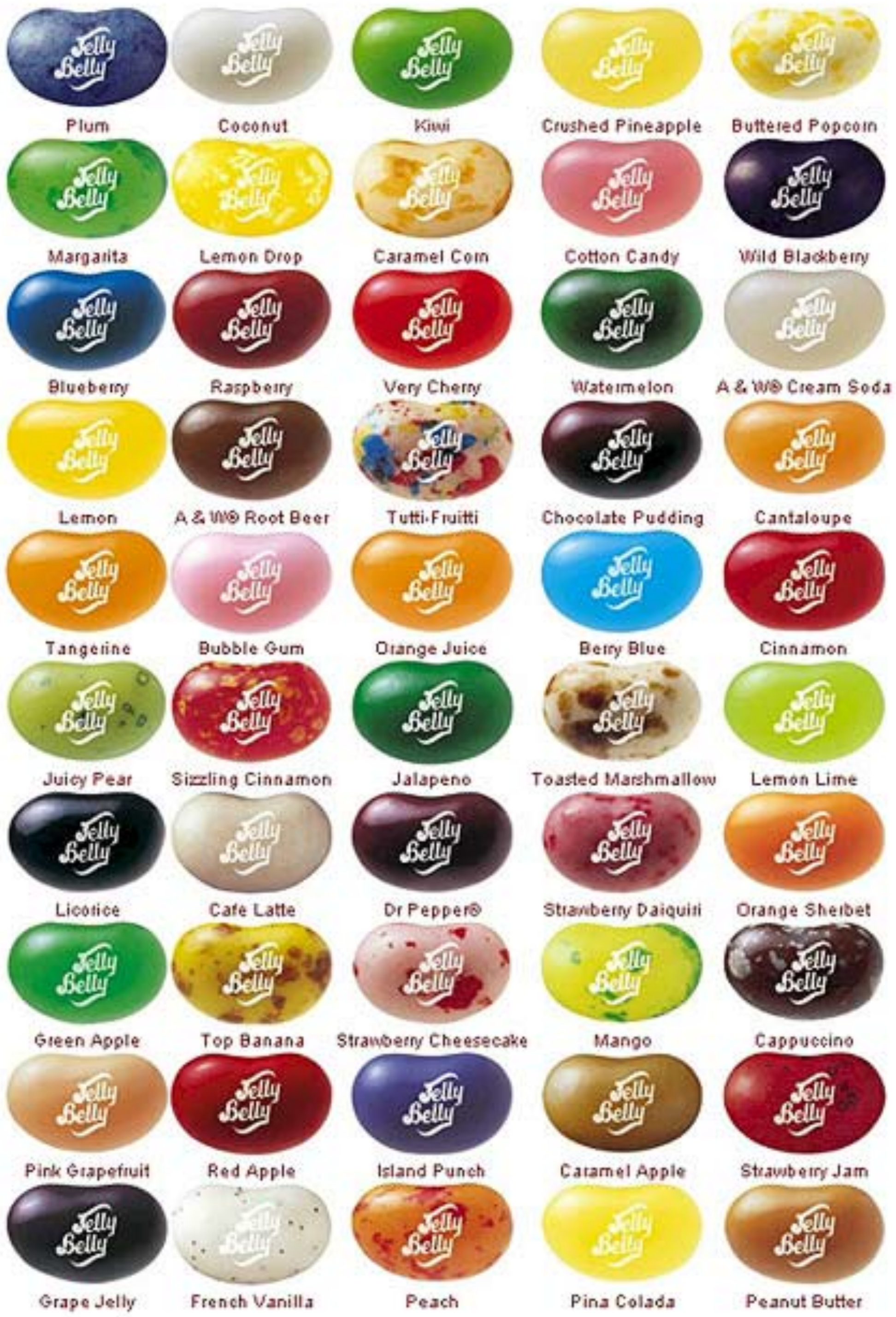
← Required
← - - - Recommended
Approved
Intended Future
(partial) r0x0r Skill Tree
("x86-prereq derived")
YOU ARE HERE :D
Malware Analysis (2 day, Matt Briggs, Frank Posluszny)
Exploits 3 (2 day, Corey Kallenberg)
Advanced x86: Real Mode (3day, John Butterworth)
Advanced x86: Virtualization (2day, David Weinstein)
Rootkits (2 day, Xeno Kovah, April 2011)
Reverse Engineering (2 day, Matt Briggs)
Exploits 2 (3 day, Corey Kallenberg)
Intermediate x86 (2 day, Xeno Kovah)
Life of Binaries (2 day, Xeno Kovah)
Vulnerabilities & Exploits 1 (2 day, Corey Kallenberg)
Intro x86 (2 day, Xeno Kovah)

About Me

- Security nerd - generalist, not specialist
- Been following rootkits for quite a while, but mostly as just a side thing to keep an eye on. But therefore I was ready to strike when some work came up in the area.
- Mostly made of 4 elements - Carbon, Hydrogen, Nitrogen, and…Oxygen!
- http://www.youtube.com/watch?v=d0zION8xjbM#t=2m21s

About You?

- Name & Department
- Why did you want to take the class?
- Which jelly belly flavors do you hate? (Because I decided the "which is your favorite" is too hard a question)


Plum
Coconut
Kiwi
Crushed Pineapple
Buttered Popcorn
Margarita
Lemon Drop
Caramel Corn
Cotton Candy
Wild Blackberry
Blueberry
Raspberry
Very Cherry
Watermelon
A & W® Cream Soda
Lemon
A & W® Root Beer
Tutti-Fruitti
Chocolate Pudding
Cantaloupe
Tangerine
Bubble Gum
Orange Juice
Berry Blue
Cinnamon
Juicy Pear
Sizzling Cinnamon
Jalapeno
Toasted Marshmallow
Lemon Lime
Licorice
Cafe Latte
Dr Pepper®
Strawberry Daiquiri
Orange Sherbet
Green Apple
Top Banana
Strawberry Cheesecake
Mango
Cappuccino
Pink Grapefruit
Red Apple
Island Punch
Caramel Apple
Strawberry Jam
Grape Jelly
French Vanilla
Peach
Pina Colada
Peanut Butter

Agenda

- Day 1 - Part 1 - Rootkit stuff
- Day 1 - Part 2 - More rootkit stuff
- Day 2 - Part 3 - ???
- Day 2 - Part 4 - Profit!

Miss Alaineous

- Questions: Ask ‘em if you got ‘em
  - If you fall behind and get lost and try to tough it out until you understand, it’s more likely that you will stay lost, so ask questions ASAP.
- Browsing the web and/or checking email during class is a good way to get lost ;)
- 2 hours, 10 min break, 1.5 hours, lunch, 1 hour w/ 5 min break thereafter

What does it all mean?!?!

- Try to have a little more practical class
- Practical in the sense that one way or another you'll learn about new tools and how you can use them to detect rootkits.
- But simultaneously I want to reinforce how much better off you are for having taken the other classes ;)
- Don't have enough time to get heavy into the attribution of changes. That would be things like "What module allocated this memory? Where in the module is the code which causes the changes?" etc
  - Also need the RE class for that. You DID register for the RE class already didn't you?

why, Why, WHY!?!?

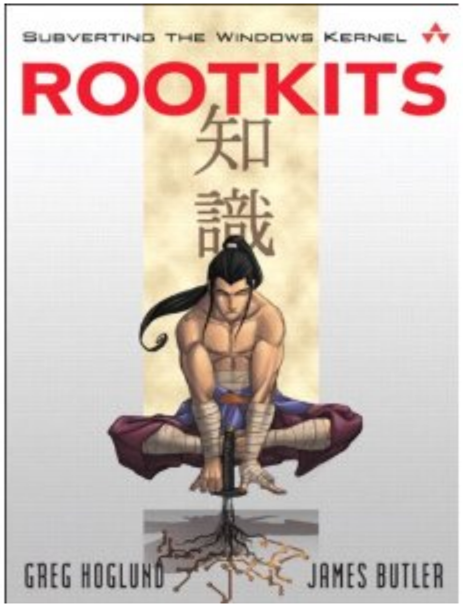
Why have a homework before anyone has learned anything?!

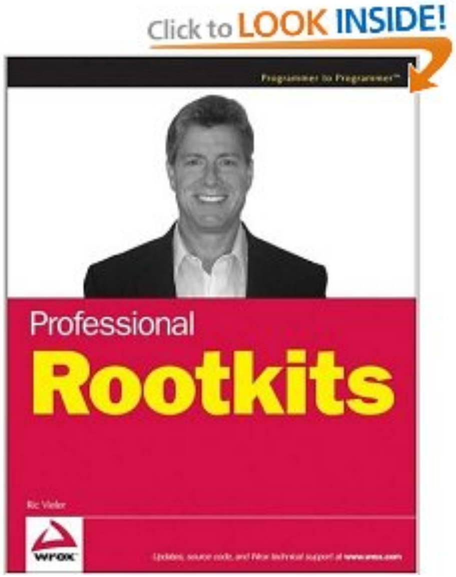
- Understand what people (sponsors/ subordinates/you) would actually go through/ see when trying to detect rootkits (if they even knew to try.)
- Understand that some tools are more equal than others when it comes to detecting things, and the danger of a false sense of security.
- Provide a concrete before-and-after picture of the necessity of this type of information for even being able to understand what the good tools are trying to tell you
- Have the tools in-hand to then apply them to other systems

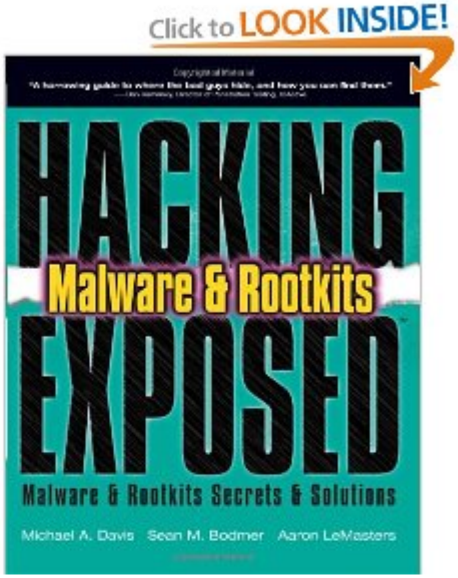
Watchugot? Watchuget?

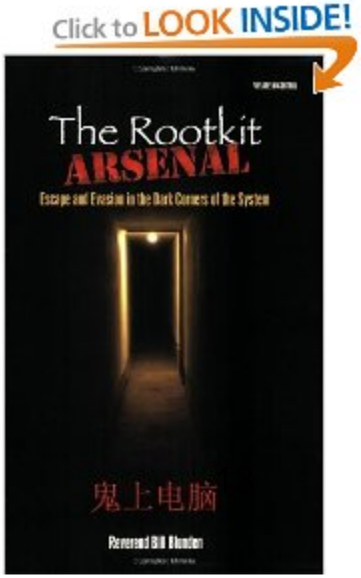
- You’ve got:
  - Rootkits VM
- You're going to get
  - Anonymized homework writeups from everyone in all the instances of this class
  - Rootkit detector capability comparison matrix
  - TiddlyWiki describing how to install the rootkits (targeted at other instructors) + some reverse engineering rootkit material cut from the RE class
  - A collection of more detectors, and a collection of more proof-of-concept rootkits from places like rootkit.com (be warned, some of the PoCs will be detected by AV, so don't use on your work laptop.)
  - Eventually, 2nd "for fun" rootkit VM :D, which still just uses techniques from this class, but takes away most of the easy win detection mechanisms

Textbook pros/cons









2005 - Rootkits: Subverting the Windows Kernel

- Pro: Written by two people who contributed a lot to the foundations of understanding what' s possible with rootkits
- Con: …but starting to show its age, with lack of many newer techniques.
- Con: Without existing OS internals knowledge, could be too much complexity too fast. Windows Internals book by MS definitely helps to explain what they' re talking about at some points.

2007 - Professional Rootkits

- Pro: Builds up a rootkit of increasing capabilities, with explanations of the code
- Cons: Adds nothing new to the field, just basically a reference for example code for the most stable versions of various techniques (not always the most stealthy techniques.)
- E.g. the type of thing which can be used to make the Sony Rootkit style software

2009 - The Rootkit Arsenal: Escape and Evasion in the Dark Corners of the System

- Pro: More inclusive of newer techniques like bootkits than the Hoglund/Butler book.
- Pro/Con: Comes with lots of code, BUT…Doesn’t allow you to download the code from anywhere, so if you want to experiment with it, you have to re-type it (or go find the original)
- Con: A bunch of the code is apparently just re-written from other people’s example code (e.g. files on rootkit.com). Also either doesn’t know how to program (use -> not *. in C!) or he was just trying to further obfuscate ripped off code.
- Con/Pro: Author comes from a forensics background rather than having OS knowledge, and thus he throws in a bunch of forensics stuff (which I question the relevance of, because I consider anti-forensics to be its own separate field from rootkit hiding). But if you haven't had exposure to anti-forensics, then it's a pro as you can learn more.

2010 - Hacking Exposed: Malware & Rootkits

- Pro: Good up to date reference which covers rootkits as they are seen in the wild, with many references to specific malware instances
- Pro/Con: Overall does a decent job, but while rootkits are sexy and therefore get cover billing, they’re still a minority content area (around 120 pages of how rootkits work and 34 pages of detection).
- Con: A lot of the detection recommendations are un-actionable, though that’s a problem for anyone talking about the area.
- Con: Almost no source code

What is a rootkit?

(or more importantly, how will I define it for this class)

- It’s an overused term is what it is
- It's neither a root, nor a kit
- An attacker tool
- NOT how they get root
- "A rootkit is a set of programs which *PATCH* and *TROJAN* existing execution paths within the system. This process violates the *INTEGRITY* of the TRUSTED COMPUTING BASE (TCB)." - Greg Hoglund, http://www.phrack.com/issues.html?issue=55&id=5
- The only universal truth about rootkits is that *they are trying to **hide** the attacker’s presence*
- 2 basic categorization schemes though

http://spennypost.blogspot.com/2010/10/fbu-bonfire-night-strike.html

Lord of the rings around the rosie

- Ring 3 – Userspace-Based
- Ring 0 – Kernel-Based
- “Ring -1” – Virtualization-Based
  - Intel VT-x(Virtualization Technology for x86), AMD-V (AMD Virtualization), Hypervisor subverted
- "Ring -1.5?" - Post-BIOS, Pre OS/VMM
  - e.g. Master Boot Record (MBR) "bootkit"
  - Peripherals with DMA(Direct Memory Access) (this can be ring 0, -1, or -1.5 depending on whether VT-d is being used)
  - Not a generally acknowledged "ring", but the place I think it fits best
- “Ring -2” – System Management Mode (SMM)
- "Ring -2.5" - BIOS (Basic Input Output System), EFI (Extensible Firmware Interface)
  - because they are the first code to execute *on the CPU* and they control what gets loaded into SMM
  - Not a generally acknowledged "ring", but the place I think it fits best
- “Ring -3” – Chipset Based
  - Intel AMT(Active Management Technology)

But BIOS could use VT-d to prevent DMA, and it initializes peripherals, so…?
Yeah, things get squishy at the bottom with non-real-rings.

Stealth Malware Taxonomy

Joanna Rutkowska 2006

- http://invisiblethings.org/papers/malware-taxonomy.pdf
- Type 0: Uses only legitimate system features
- Type 1: Modifies things which should be static
- Type 2: Modifies things which are dynamic
- Type 3: Exists outside the operating system

- Type 4: Exists outside the main CPU/RAM
  - Added by me

Example Type 0 Malware

- Spyware
  - There's nothing illegitimate about a cell phone map application wanting to access your location data to show the local map. It's only when it starts sending that location with your PII to a 3rd party location that it starts to become questionable.
- Trojans
  - There's nothing illegitimate about allowing users to install programs. And there's no realistic way for a user to assess the full extent of all that program's capabilities. When a program contains capabilities which arguably have nothing to do with its advertised purpose, that's when it becomes questionable.
- Bots
  - There's nothing illegitimate about allowing an application to make network connections. It's only when it's making thousands of them as a part of a DDoS that's when it becomes questionable.
- Hide in plain sight
  - Programs can name themselves whatever the developer wants. But when the developer wants it to be named misleadingly similar to a "trusted" software vendor like Microsoft's files, that's when it becomes questionable.

Detecting Type 0

- “Out of scope” for the taxonomy ;)
  - Also mostly out of scope for this class
- Blacklisting
  - Signature-based Anti-Virus
- Behavioral analysis
  - Triumfant, QualysGuard, most AV to some degree
- Filesystem integrity checking
  - Tripwire, Bit9, SolidCore (for HBSS)

Why is Type 0 going undetected?

- Companies are overly invested in blacklisting technology. Explosion in polymorphism undermining signature-based approaches.
- Whitelisting technologies often require dedicated maintainers to understand “expected” or “known good” state. Thus they are typically not targeted at home users.

Stealth Malware Taxonomy

Joanna Rutkowska 2006

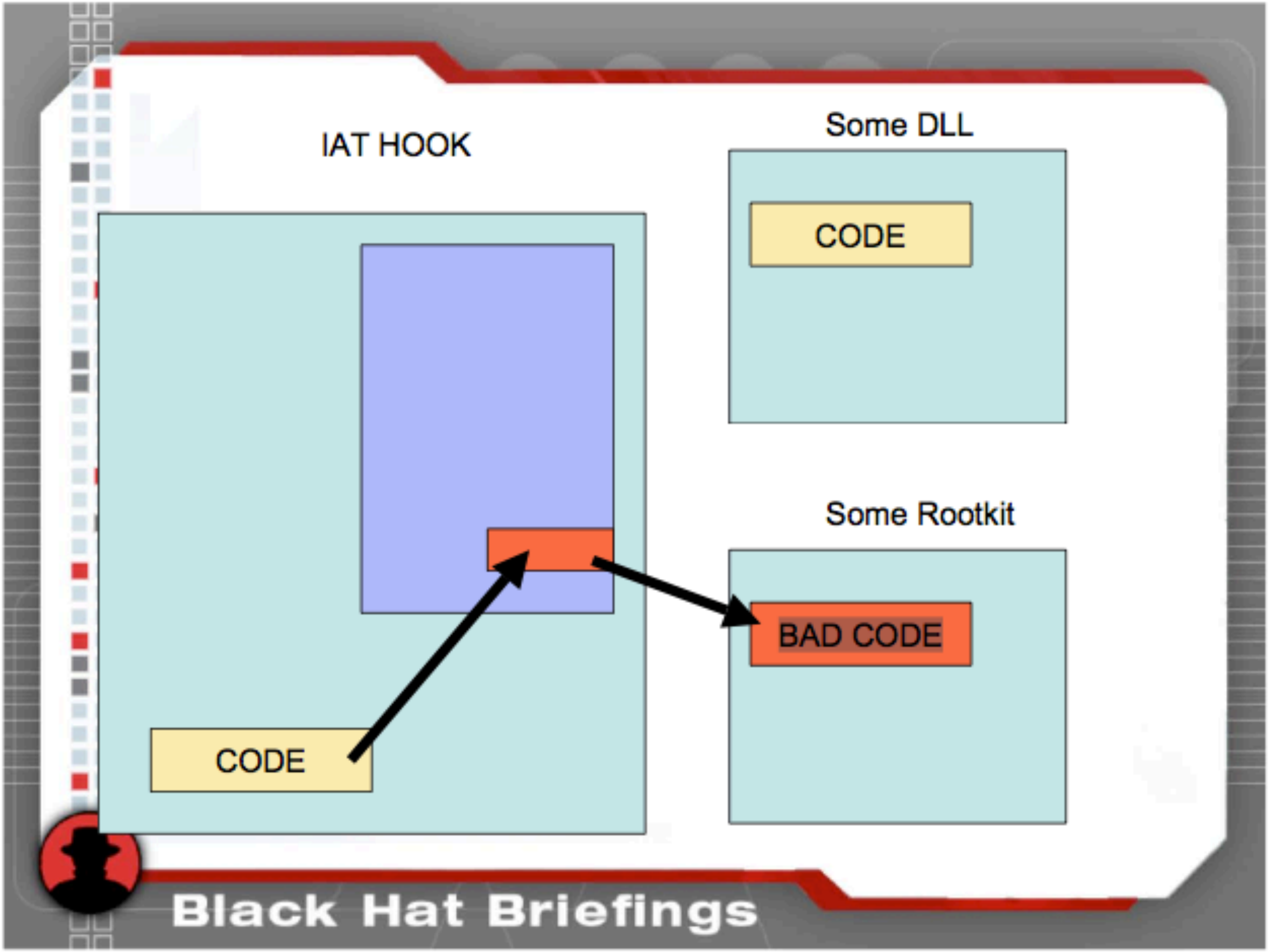
- http://invisiblethings.org/papers/malware-taxonomy.pdf
- Type 0: Uses only legitimate system features
- Type 1: Modifies things which should be static
- Type 2: Modifies things which are dynamic
- Type 3: Exists outside the operating system

- Type 4: Exists outside the main CPU/RAM
  - Added by me

Example Type 1 Malware

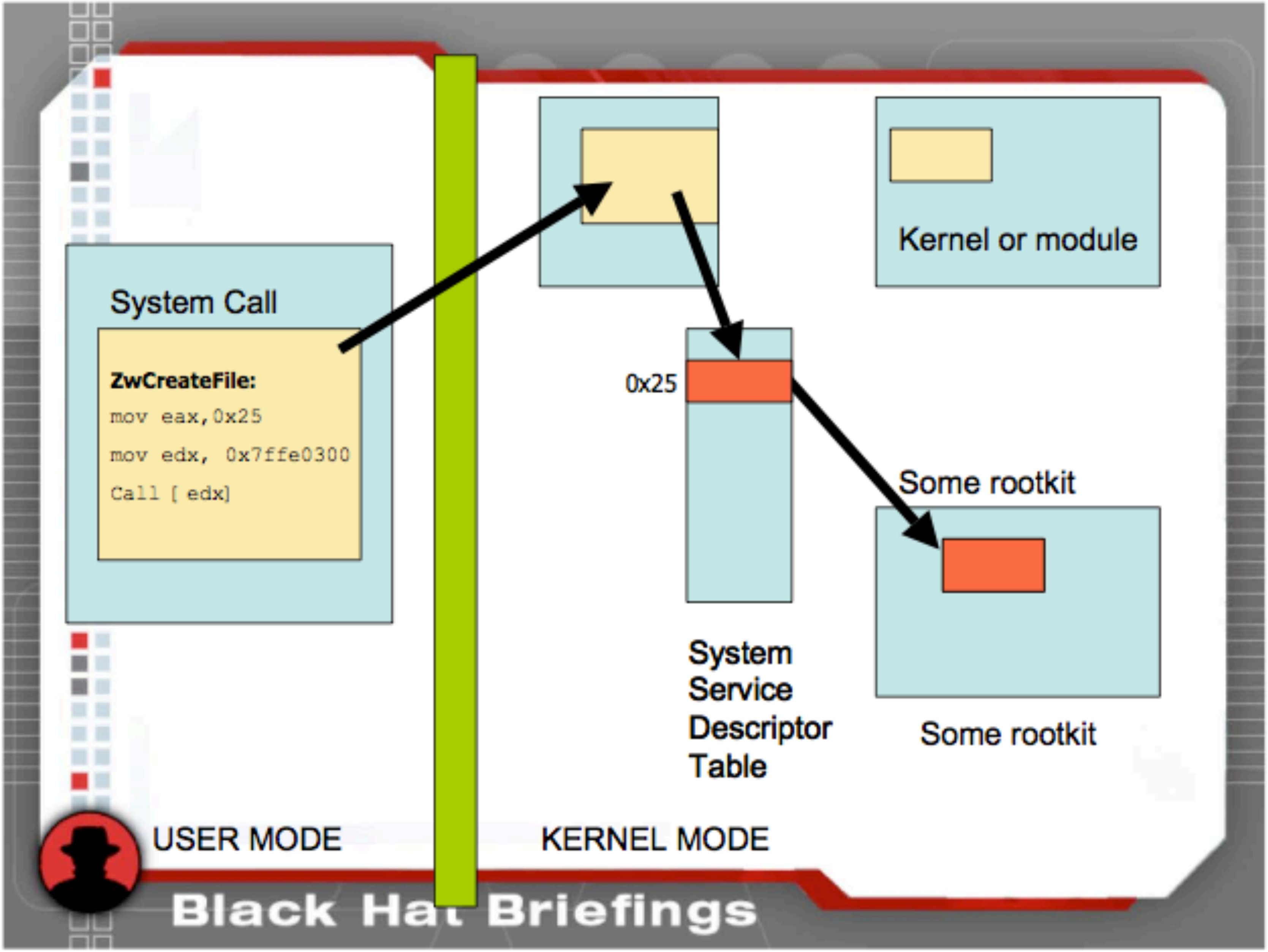
- Most in-the-wild rootkits are a mix of Type 1 and Type 2
- The following are a quick glimpse at some of the techniques we're going to be looking at in this class.

IAT Hook



From: http://www.blackhat.com/presentations/bh-europe-06/bh-eu-06-Silberman-Butler.pdf

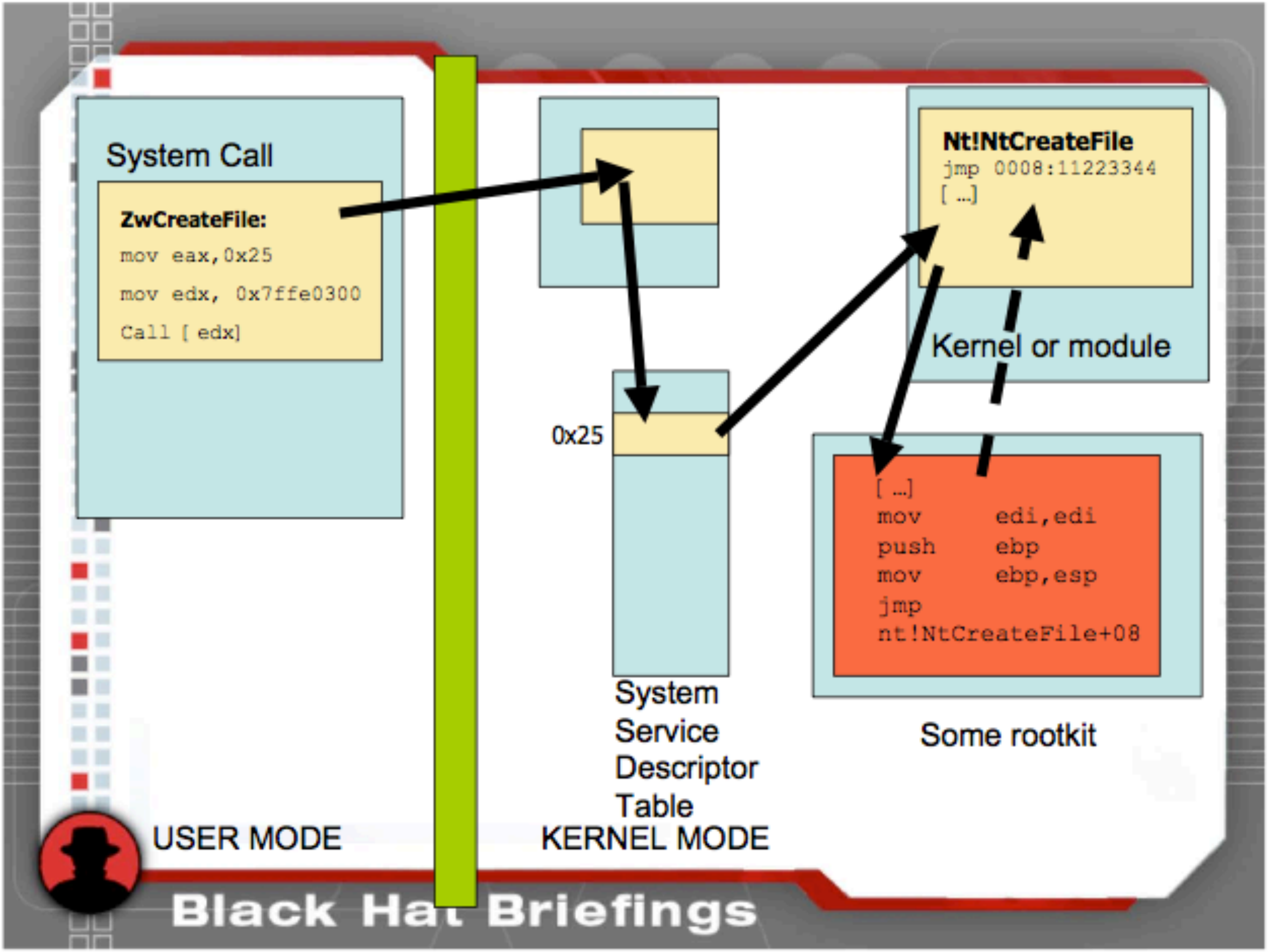
SSDT Hook



```
ZwCreateFile:
mov eax,0x25
mov edx, 0x7ffe0300
Call [ edx]
```

From: http://www.blackhat.com/presentations/bh-europe-06/bh-eu-06-Silberman-Butler.pdf

Inline Hook



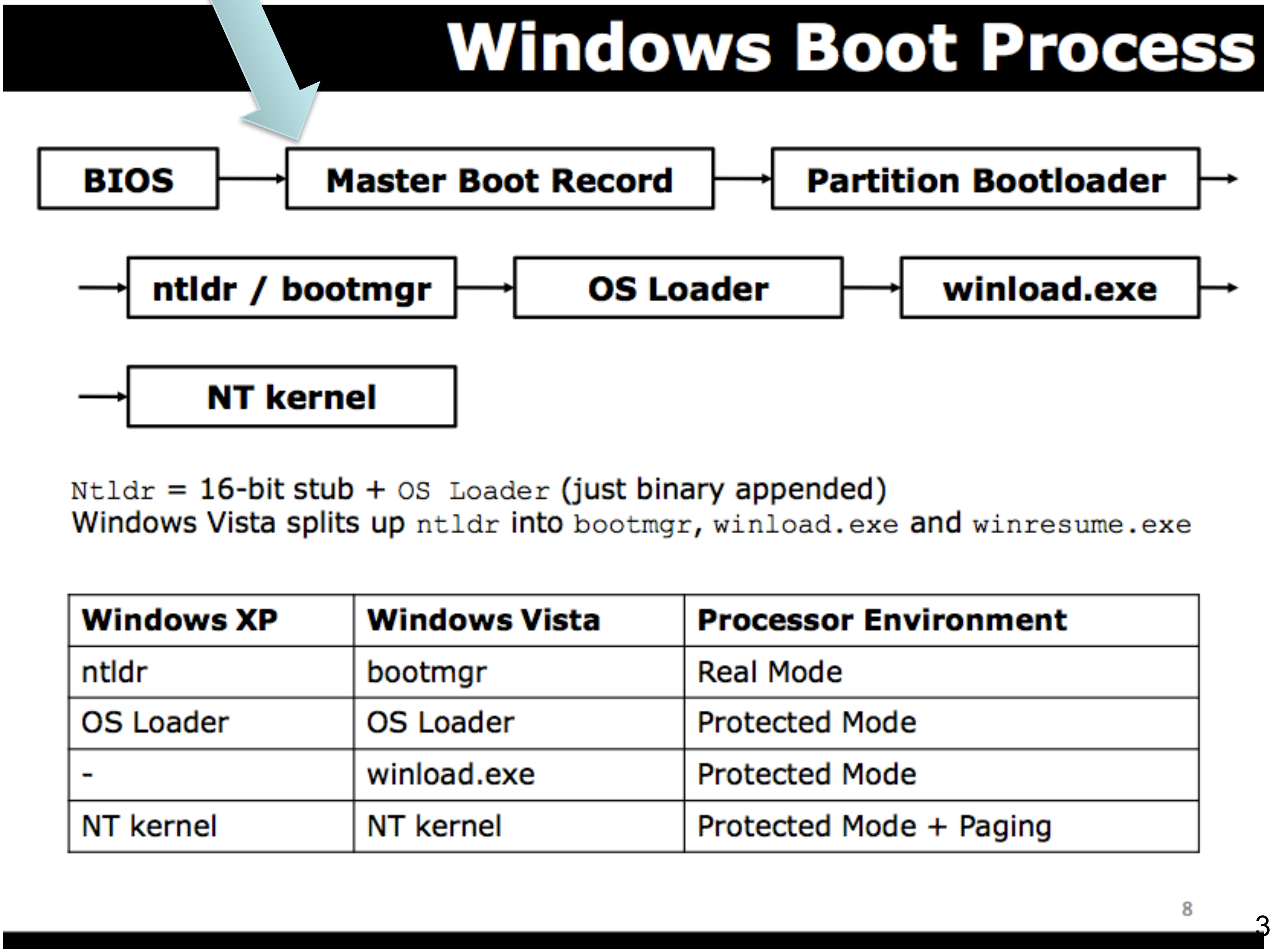
```
ZwCreateFile:
mov eax,0x25
mov edx, 0x7ffe0300
Call [ edx]
```

```
Nt!NtCreateFile
jmp 0008:11223344
[ …]
```

```
[ …]
mov      edi,edi
push     ebp
mov      ebp,esp
jmp
nt!NtCreateFile+08
```

From: http://www.blackhat.com/presentations/bh-europe-06/bh-eu-06-Silberman-Butler.pdf

Bootkit Lives here (from disk), but in order to do anything of consequence it has to keep hooking each subsequent thing to keep control.



Windows Boot Process

BIOS → Master Boot Record → Partition Bootloader →

→ ntldr / bootmgr → OS Loader → winload.exe →

→ NT kernel

`Ntldr` = 16-bit stub + `OS Loader` (just binary appended)
Windows Vista splits up `ntldr` into `bootmgr`, `winload.exe` and `winresume.exe`

Windows XP	Windows Vista	Processor Environment
ntldr	bootmgr	Real Mode
OS Loader	OS Loader	Protected Mode
-	winload.exe	Protected Mode
NT kernel	NT kernel	Protected Mode + Paging

From http://www.stoned-vienna.com/downloads/Presentation.pdf

Detecting Type 1

- ~~GMER - My favorite (www.gmer.net)~~
  - Here comes a new challenger! Virus Blok Ada (the people who found Stuxnet) have been significantly improving their anti-rootkit (Vba32arkit.exe), and since it has extra *removal* capabilities built in, I'm diggin' it. Shoryuken!
- Tuluka, GMER, RootkitUnhooker, IceSword, Helios Lite, RootkitRevealer, System Virginity Verifier(SVV), WinDbg ! chkimg, VICE, RAIDE, chkrootkit, etc,
- See http://www.antirootkit.com/software/index.htm and http://ntinternals.org/anti_rootkits.php
- [VMWatcher] for out of band integrity checks
- Strider [GhostBuster] for cross-view of hiding things on disk (but you can generally detect bootkits with memory integrity checks, and you can’t get GhostBuster anyway)

Preventing Type 1

- PatchGuard. Windows x64
  - Unintended consequences? Pushes Type 1 to Type 0 or Type 2?
  - Still need detection? x64 bootkit in the wild [3]
- [NICKLE]. Assumes virtualized system
  - What about VM escape? Still need detection?
  - [HyperSentry]

Why are Type 1 going undetected?

- None of the previously listed software is meant to be run in an enterprise; they’ re meant to be run manually on single systems.
- The best detectors need deep system knowledge in order to interpret the results. Administrators may not have this knowledge.

Stealth Malware Taxonomy

Joanna Rutkowska 2006

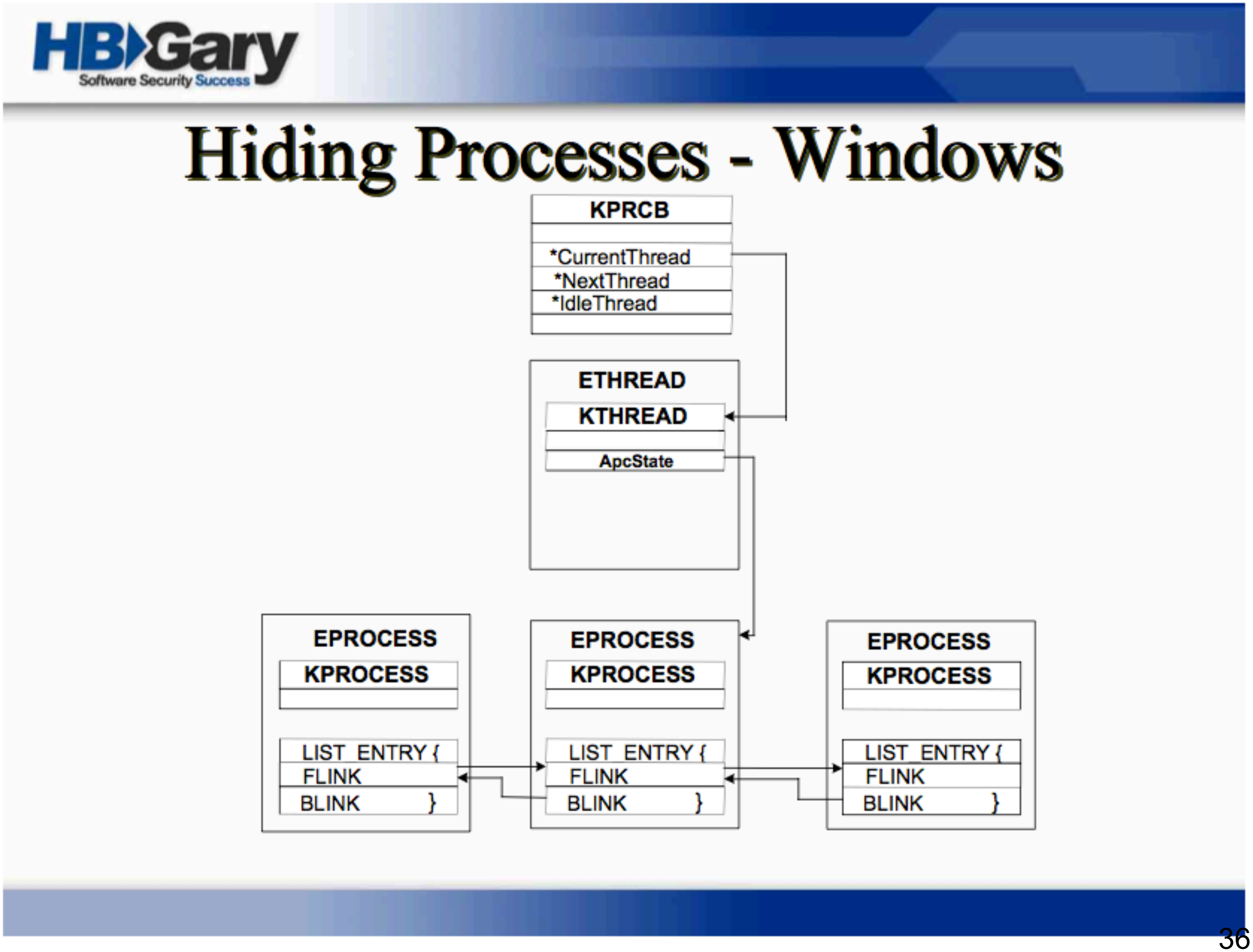
- http://invisiblethings.org/papers/malware-taxonomy.pdf
- Type 0: Uses only legitimate system features
- Type 1: Modifies things which should be static
- Type 2: Modifies things which are dynamic
- Type 3: Exists outside the operating system

- Type 4: Exists outside the main CPU/RAM
  - Added by me

Example Type 2 Malware

- Direct Kernel Object Manipulation [DKOM]
  - Developed specifically to avoid using Type 1 hooking, because it was recognized to be eminently detectable (presented hook detector VICE at same time)
- Kernel Object Hooking [KOH]
  - Generalization of existing techniques, with suggestions of some example Windows objects to hook

Process Linked List Before DKOM



Hiding Processes - Windows

KPRCB

*CurrentThread

*NextThread

*IdleThread

ETHREAD

KTHREAD

ApcState

EPROCESS

KPROCESS

LIST ENTRY {

FLINK

BLINK }

EPROCESS

KPROCESS

LIST ENTRY {

FLINK

BLINK }

EPROCESS

KPROCESS

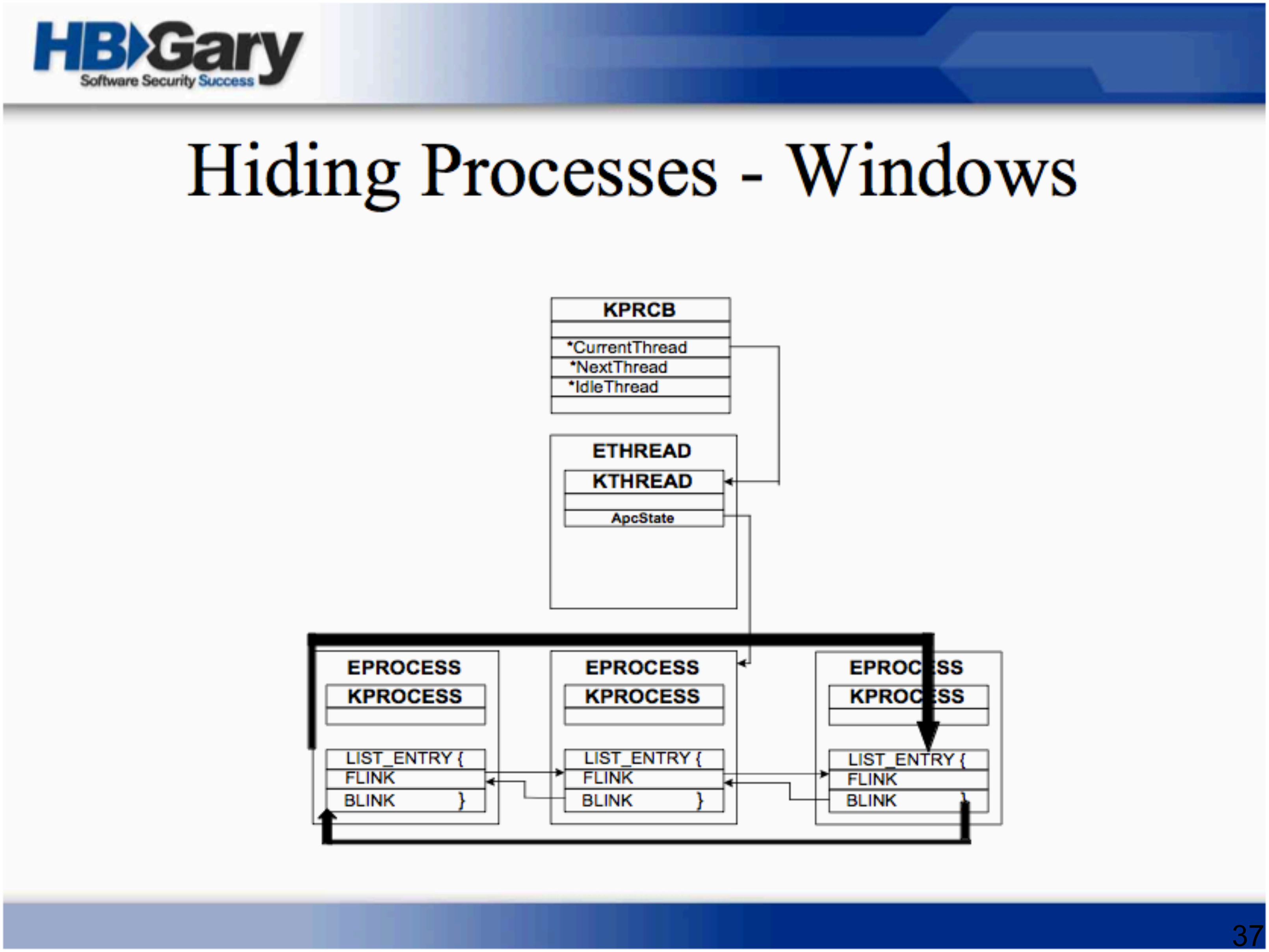
LIST ENTRY {

FLINK

BLINK }

From: http://www.blackhat.com/presentations/win-usa-04/bh-win-04-butler.pdf

Process Linked List After DKOM



Hiding Processes - Windows

From: http://www.blackhat.com/presentations/win-usa-04/bh-win-04-butler.pdf

KOH

- Hook function pointers in dynamically allocated objects in the kernel
- typedef struct {

```
    SHORT                    Type;
    UCHAR                    Number;
    UCHAR                    Importance;
    LIST_ENTRY               DpcListEntry;
    PKDEFERRED_ROUTINE       DeferredRoutine;
    PVOID                    DeferredContext;
    PVOID                    SystemArgument1;
    PVOID                    SystemArgument2;
    PULONG                   Lock;
} KDPC, *PKDPC;
```

Detecting Type 2

- Plenty of things handle canonical DKOM through “cross-view” detection
  - VBA32AR, GMER, IceSword, RootkitRevealer, F-Secure BlackLight, Sophos Anti-Rootkit, etc
- In some cases you may be able to automatically infer semantic constraints on data structures and verify them at runtime [Petroni][LKIM]
- Recent academic interest in KOH
  - [HookMap], [HookSafe], [HookScout]

Why are Type 2 going undetected?

- Same reasons as for Type 1, and…
- No good tools to detect KOH. Detecting KOH system-wide (as opposed to specific things attackers are known to use) looks like it could induce unacceptable performance penalty. Also KOH detection could be more prone to race conditions, and attempts to eliminate these conditions would add more performance overhead. More work needed there.

Stealth Malware Taxonomy

Joanna Rutkowska 2006

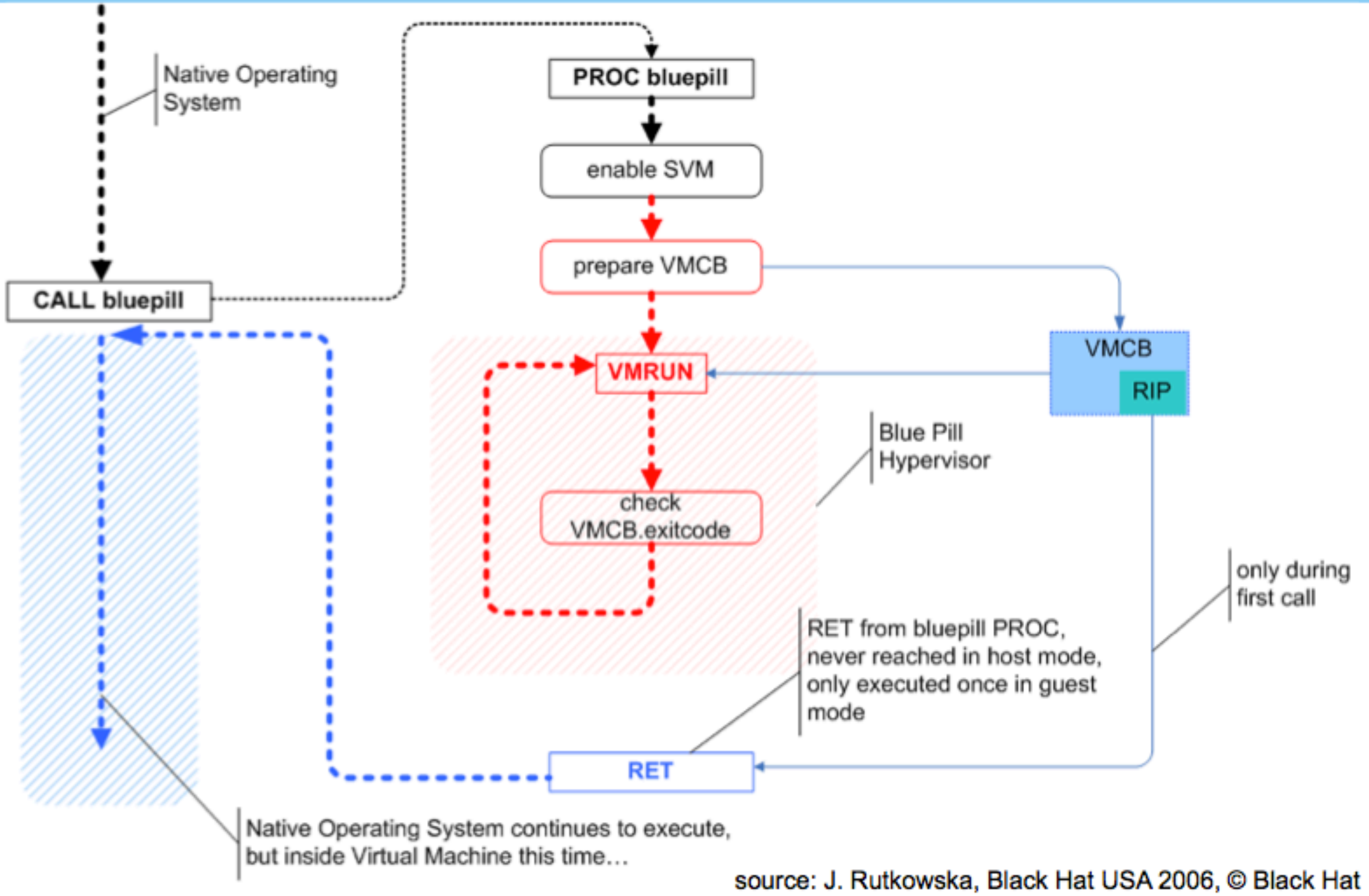
- http://invisiblethings.org/papers/malware-taxonomy.pdf
- Type 0: Uses only legitimate system features
- Type 1: Modifies things which should be static
- Type 2: Modifies things which are dynamic
- Type 3: Exists outside the operating system

- Type 4: Exists outside the main CPU/RAM
  - Added by me

Example Type 3 Malware

- “Ring -1” – Virtualization-Based
  - Intel VT-x(Virtualization Technology for x86), AMD-V (AMD Virtualization), Hypervisor subverted
- "Ring -1.5?" - Post-BIOS, Pre OS/VMM
  - e.g. Master Boot Record (MBR) "bootkit"
  - Peripherals with DMA(Direct Memory Access) (this can be ring 0, -1, or -1.5 depending on whether VT-d is being used)
  - Not a generally acknowledged "ring", but the place I think it fits best
- “Ring -2” – System Management Mode (SMM)
- "Ring -2.5" - BIOS (Basic Input Output System), EFI (Extensible Firmware Interface)
  - because they are the first code to execute *on the CPU* and they control what gets loaded into SMM
  - Not a generally acknowledged "ring", but the place I think it fits best
- “Ring -3” – Chipset Based
  - Intel AMT(Active Management Technology)

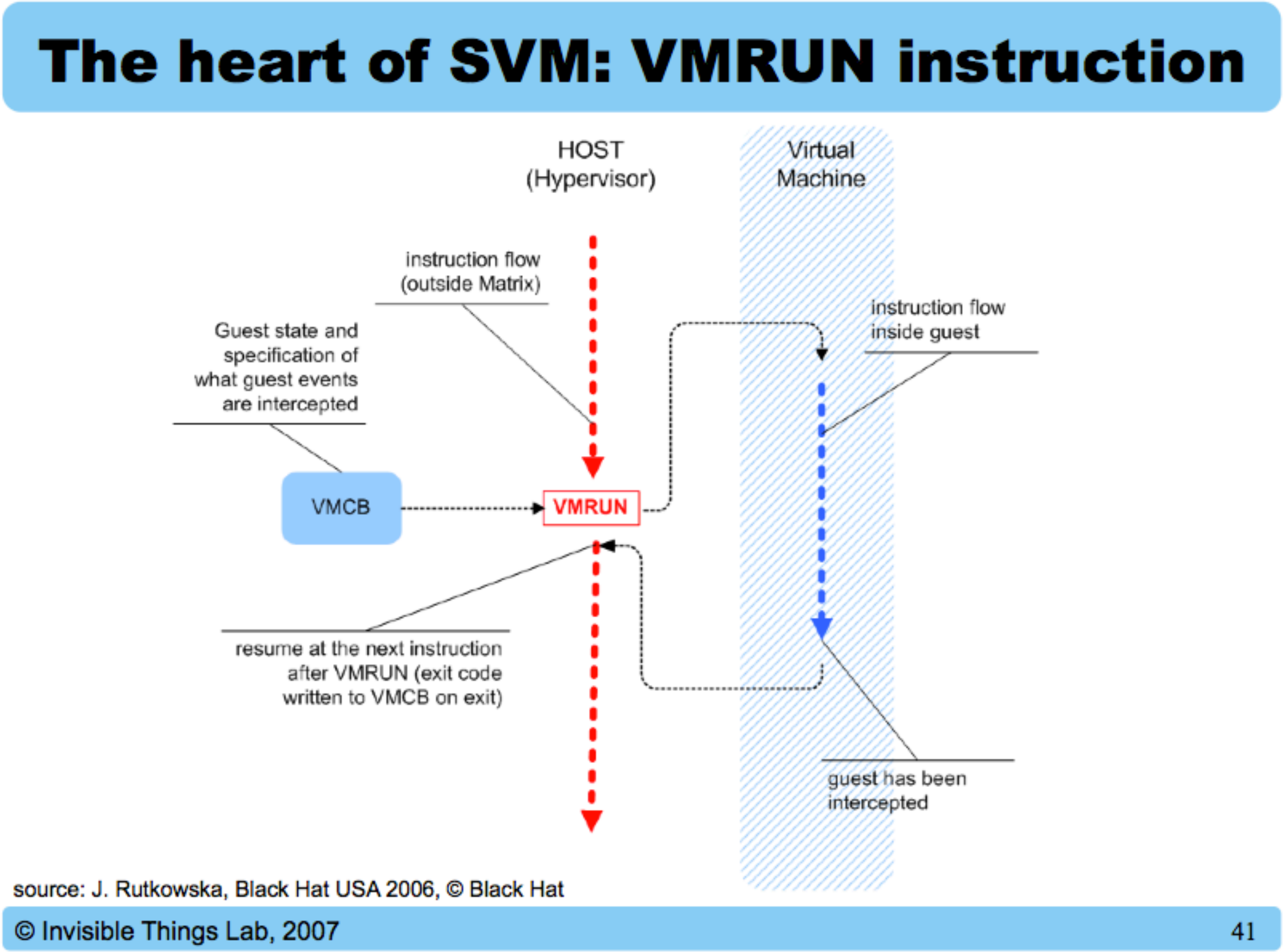
Blue Pill Idea (simplified)



Native Operating System
PROC bluepill
enable SVM
prepare VMCB
CALL bluepill
VMCB
RIP
VMRUN
Blue Pill Hypervisor
check VMCB.exitcode
only during first call
RET from bluepill PROC, never reached in host mode, only executed once in guest mode
RET
Native Operating System continues to execute, but inside Virtual Machine this time…

source: J. Rutkowska, Black Hat USA 2006, © Black Hat

From http://www.invisiblethingslab.com/resources/bh07/IsGameOver.pdf



The heart of SVM: VMRUN instruction

source: J. Rutkowska, Black Hat USA 2006, © Black Hat

© Invisible Things Lab, 2007 41

From http://www.invisiblethingslab.com/resources/bh07/IsGameOver.pdf

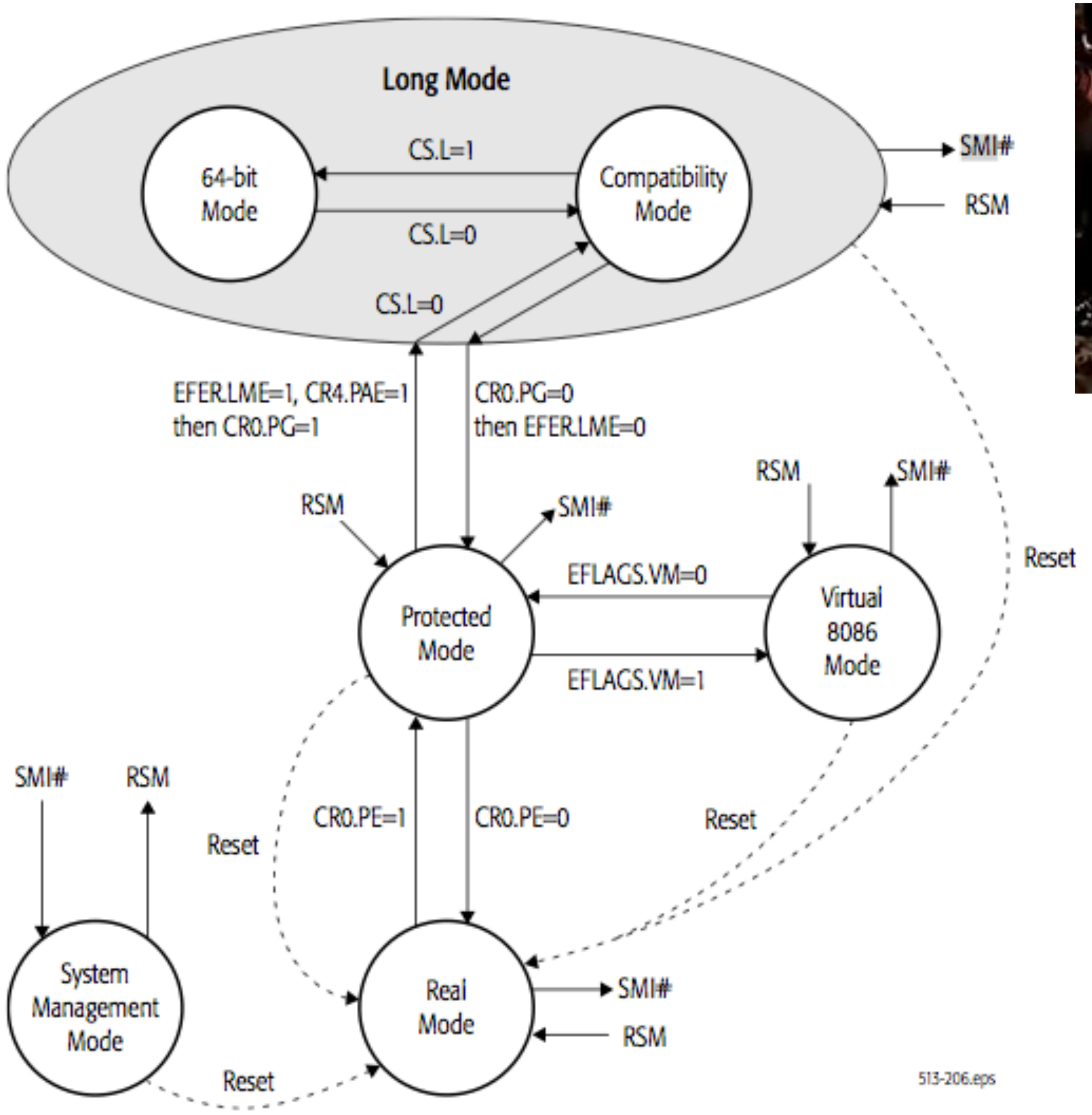


Figure 1-6. Operating Modes of the AMD64 Architecture

Batteries Not Included!

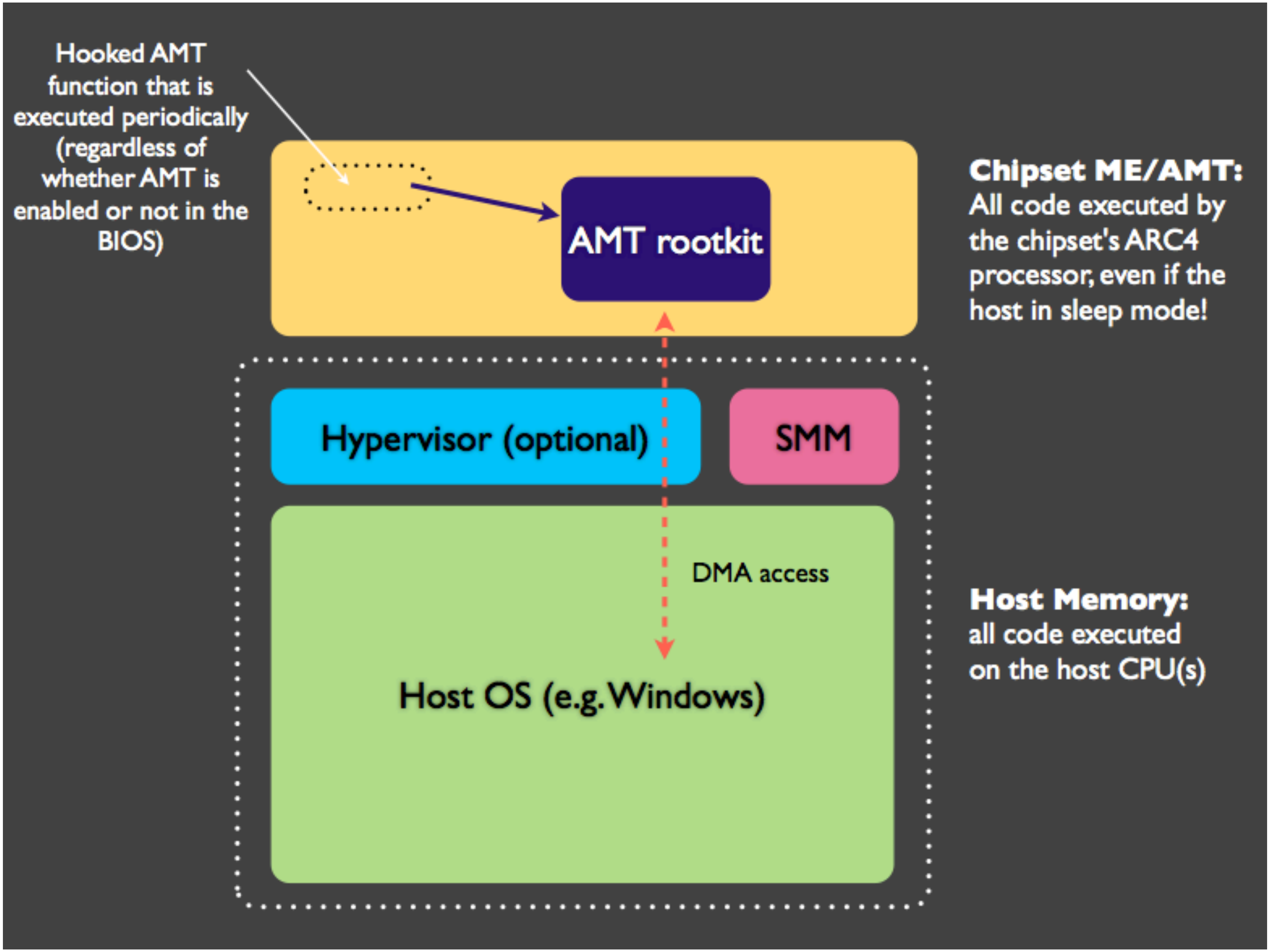
From http://support.amd.com/us/Processor_TechDocs/24593.pdf

Detecting Type 3 – Ring -1

- Due to hype surrounding ring -1 rootkits, people had incentive to find them.
- “Don’t Tell Joanna, The Virtualized Rootkit Is Dead” [8]
  - Exhibits same misunderstanding of “technically detectable” vs “people can actually detect it in practice”
- Timing side-effect detection
- “Compatibility is Not Transparency: VMM Detection Myths and Realities”[9]
- In addition some people have suggested the classic approach of “just go lower”, as in, scan from ring -2 or ring -3 (e.g. [DeepWatch])

Prevent/Detect Type 3 – Ring -2

- There are mechanisms in both Intel and AMD's virtualization extensions to "deprivilege" the code running in SMRAM, by basically virtualizing it, and limiting the code's view of memory so that it can't scribble on your OS/hypervisor.
  - AMD also has an option for the hypervisor to intercept SMIs and fake out a transition directly to SMM without requiring writing the separate minimal hypervisor which lives in SMM – talk on *implementing* this at ShmooCon 2010 [SMMshmoo]
- Not aware of any commercial vendors who do this yet.
- Can theoretically "just" integrity check SMRAM, iff you have access, which requires getting there first, or going through the same hole as an attacker


Hooked AMT function that is executed periodically (regardless of whether AMT is enabled or not in the BIOS)
AMT rootkit
Chipset ME/AMT:
All code executed by the chipset's ARC4 processor, even if the host in sleep mode!
Hypervisor (optional)
SMM
DMA access
Host OS (e.g. Windows)
Host Memory:
all code executed on the host CPU(s)

From http://www.invisiblethingslab.com/resources/bh09usa/Ring%20-3%20Rootkits.pdf

FIXME: add NIC infection

FIXME: add KBC infection

Detecting Type 4 – Ring -3

- Use other ring -3 detectors and get there first? TPM can verify a compatible BIOS, but what about everything else? [DeepWatch] wasn't designed for it, but can it help?
- Self-attestation [SWATT][SBAP] [Pioneer]
- SOL?
- Too soon to say

Why are Type 3 & 4 going undetected?

- Cache 22? Not looking for them in the wild because we' re not hearing about them being found in the wild?
- Even if we want to look for them, there are no tools to help us do so. Have to roll your own.
- Level of development effort and hardware-dependencies probably indicates they will only be used in highly targeted attacks.

Stealth Malware Taxonomy

Joanna Rutkowska 2006

- http://invisiblethings.org/papers/malware-taxonomy.pdf
- Type 0: Uses only legitimate system features
- Type 1: Modifies things which should be static
- Type 2: Modifies things which are dynamic
- Type 3: Exists outside the operating system
- Type 4: Exists outside the main CPU/RAM
  - Added by me

They Might Be Giants:
Where your eyes don't go
(rootkit themesong as far as I'm concerned)

- Where your eyes don't go a filthy scarecrow waves its broomstick arms
  And does a parody of each unconscious thing you do
  When you turn around to look it's gone behind you
  On its face it's wearing your confused expression
  Where your eyes don't go

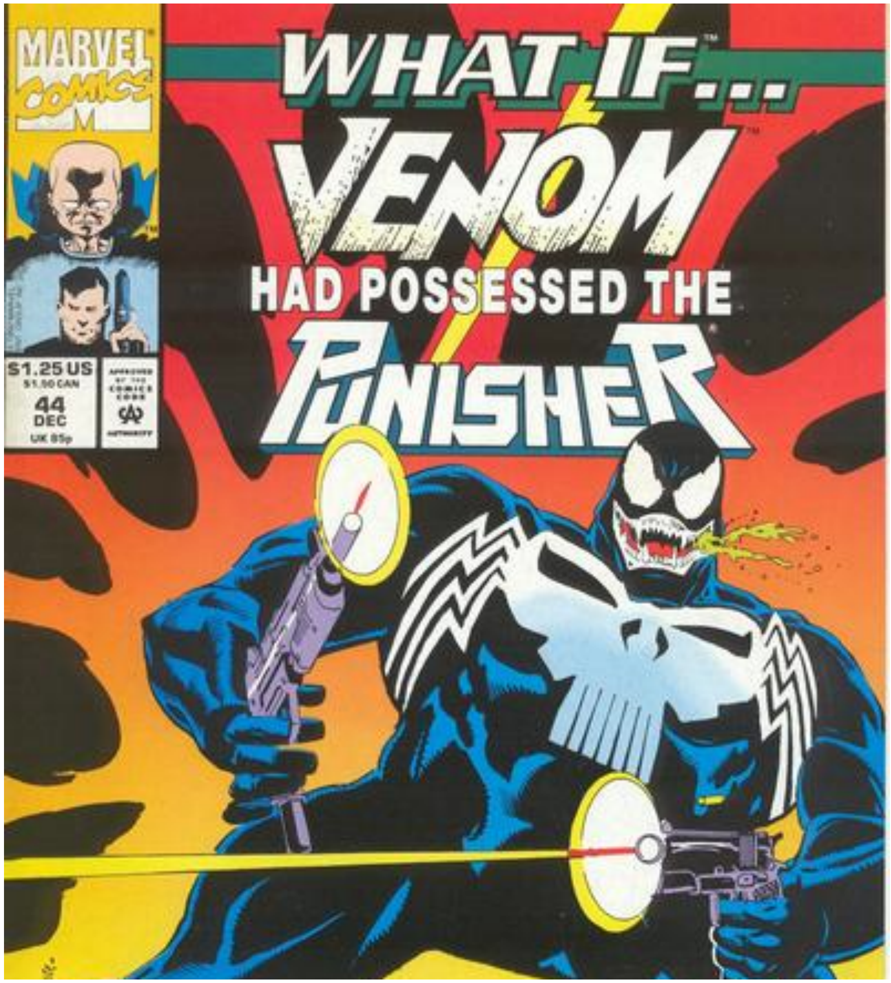
  Where your eyes don't go a part of you is hovering
  It's a nightmare that you'll never be discovering

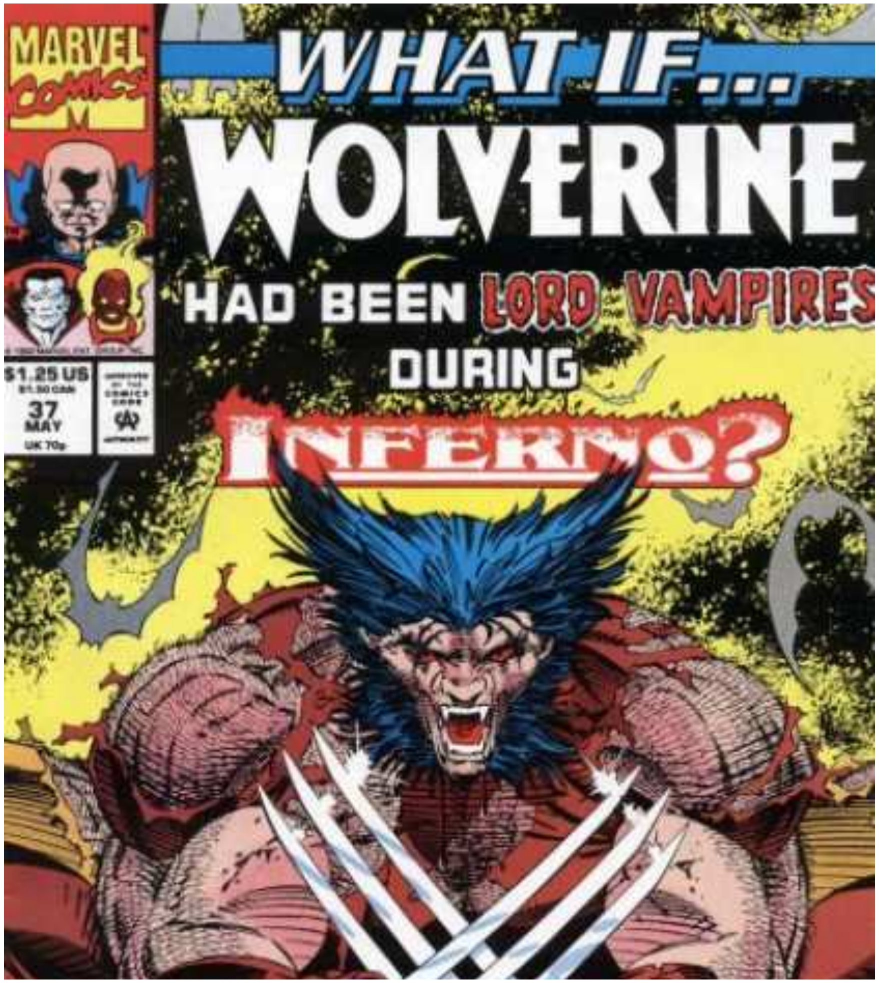
- Should you worry when the skullhead is in front of you
  Or is it worse because it's always waiting where your eyes don't go?

- http://www.youtube.com/watch?v=hqY3kASMFW8

Spoiler Alert

- There are ~8 rootkits leveraging ~10 techniques in the example VM, depending on how you count.





- What If…we ran GMER on our example VM?
- (Note to self, try and crowdsource the interpretation to start with)

Inline Hooks

PE section where the hook resides

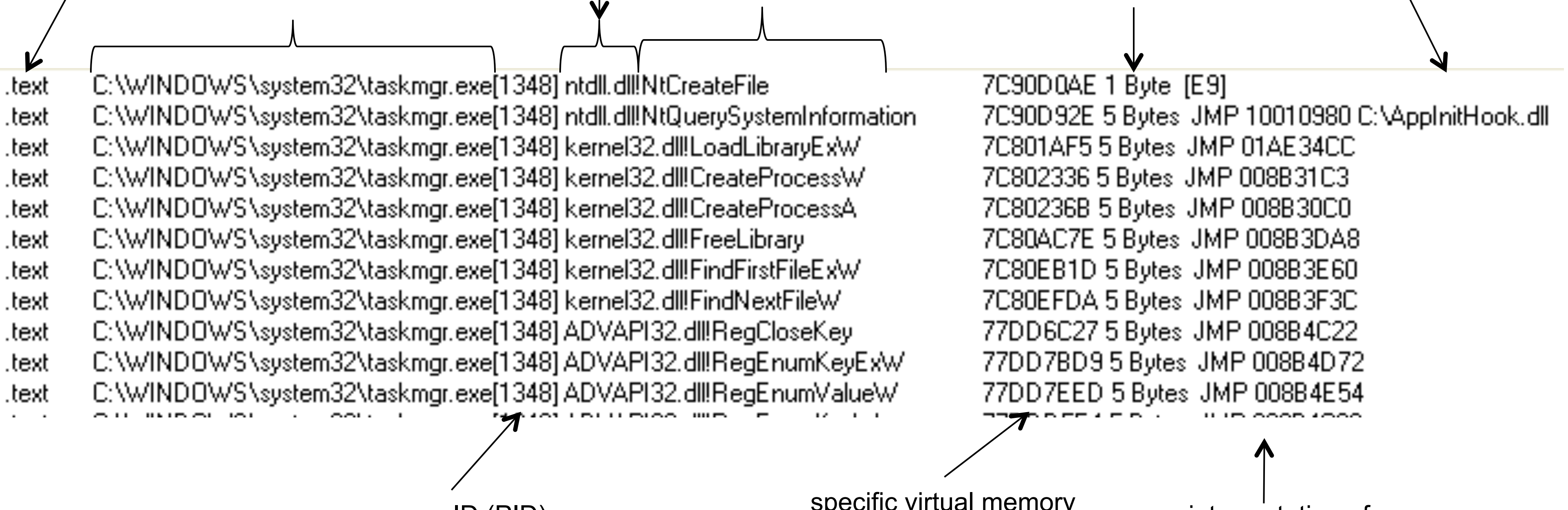
process name

module within process memory

function name within module

number of bytes that changed

if control flow redirect (call, jmp)
module space where it's redirected to
if it is within a module address range



```
.text   C:\WINDOWS\system32\taskmgr.exe[1348] ntdll.dll!NtCreateFile                   7C90D0AE 1 Byte  [E9]
.text   C:\WINDOWS\system32\taskmgr.exe[1348] ntdll.dll!NtQuerySystemInformation       7C90D92E 5 Bytes  JMP 10010980 C:\AppInitHook.dll
.text   C:\WINDOWS\system32\taskmgr.exe[1348] kernel32.dll!LoadLibraryExW              7C801AF5 5 Bytes  JMP 01AE34CC
.text   C:\WINDOWS\system32\taskmgr.exe[1348] kernel32.dll!CreateProcessW              7C802336 5 Bytes  JMP 008B31C3
.text   C:\WINDOWS\system32\taskmgr.exe[1348] kernel32.dll!CreateProcessA              7C80236B 5 Bytes  JMP 008B30C0
.text   C:\WINDOWS\system32\taskmgr.exe[1348] kernel32.dll!FreeLibrary                 7C80AC7E 5 Bytes  JMP 008B3DA8
.text   C:\WINDOWS\system32\taskmgr.exe[1348] kernel32.dll!FindFirstFileExW            7C80EB1D 5 Bytes  JMP 008B3E60
.text   C:\WINDOWS\system32\taskmgr.exe[1348] kernel32.dll!FindNextFileW               7C80EFDA 5 Bytes  JMP 008B3F3C
.text   C:\WINDOWS\system32\taskmgr.exe[1348] ADVAPI32.dll!RegCloseKey                 77DD6C27 5 Bytes  JMP 008B4C22
.text   C:\WINDOWS\system32\taskmgr.exe[1348] ADVAPI32.dll!RegEnumKeyExW               77DD7BD9 5 Bytes  JMP 008B4D72
.text   C:\WINDOWS\system32\taskmgr.exe[1348] ADVAPI32.dll!RegEnumValueW               77DD7EED 5 Bytes  JMP 008B4E54
```

process ID (PID)

specific virtual memory address where the change is found

interpretation of changed bytes (if possible)

Book page 340

!chkimg

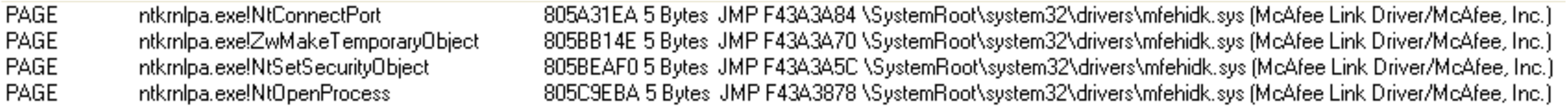
- You can also find modifications to static code/data areas with the !chkimg windbg command. It checks the version in memory against the file on disk

System Virginity Verifier

- http://invisiblethings.org/tools/svv/svv-2.3-src.zip
- http://invisiblethings.org/papers/rutkowska_bhfederal2006.ppt
- Like !chkimg but tries to apply some heuristics to the modifications it found to apply a severity score.

False Positives

McAfee HBSS HIPS



```
PAGE        ntkrnlpa.exe!NtConnectPort               805A31EA 5 Bytes  JMP F43A3A84 \SystemRoot\system32\drivers\mfehidk.sys (McAfee Link Driver/McAfee, Inc.)
PAGE        ntkrnlpa.exe!ZwMakeTemporaryObject       805BB14E 5 Bytes  JMP F43A3A70 \SystemRoot\system32\drivers\mfehidk.sys (McAfee Link Driver/McAfee, Inc.)
PAGE        ntkrnlpa.exe!NtSetSecurityObject         805BEAF0 5 Bytes  JMP F43A3A5C \SystemRoot\system32\drivers\mfehidk.sys (McAfee Link Driver/McAfee, Inc.)
PAGE        ntkrnlpa.exe!NtOpenProcess               805C9EBA 5 Bytes  JMP F43A3878 \SystemRoot\system32\drivers\mfehidk.sys (McAfee Link Driver/McAfee, Inc.)
```

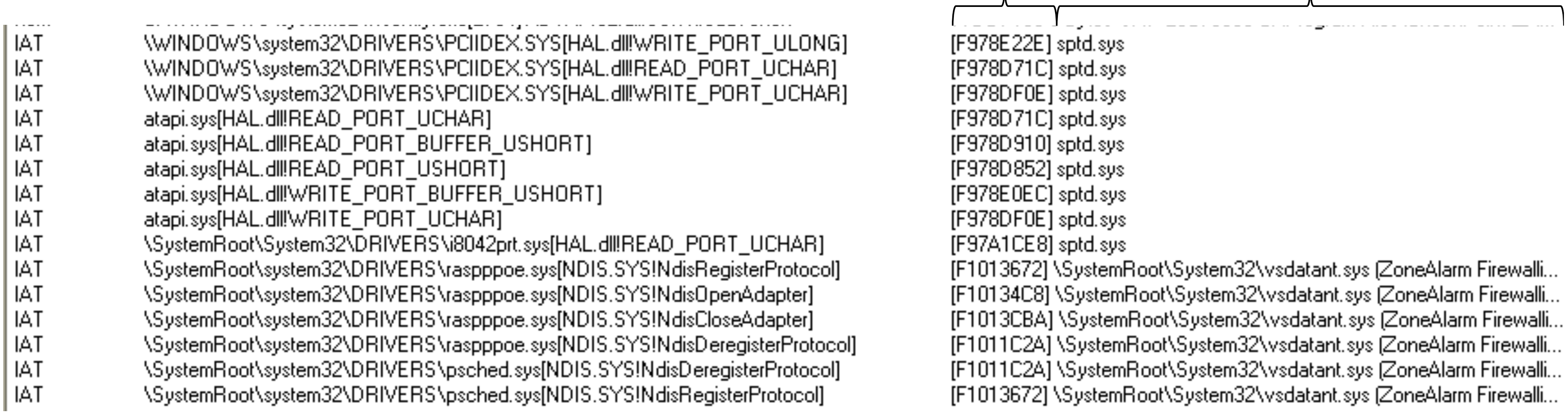
Stuxnet use of inline hooks

- From the Stuxnet Dossier: http://www.symantec.com/content/en/us/enterprise/media/security_response/whitepapers/w32_stuxnet_dossier.pdf
- "~WTR4141.tmp then loads ~WTR4132.tmp, but before doing so, it attempts to hide the files on the removable drive. Hiding the files on the removable drive as early in the infection process as possible is important for the threat since the rootkit functionality is not installed yet, as described in the Windows Rootkit Functionality section. Thus, ~WTR4141.tmp implements its own less-robust technique in the meantime.
- WTR4141.tmp hooks the following APIs from kernel32.dll and Ntdll.dll:
- From Kernel32.dll
  - FindFirstFileW
  - FindNextFileW
  - FindFirstFileExW
- From Ntdll.dll
  - NtQueryDirectoryFile
  - ZwQueryDirectoryFile"

Go with what you know…
Import Address Table (IAT) Hooks

This is the address in the IAT pointing somewhere other than where it should (based on the Exports Address Table (EAT) of the exporting module

If GMER can, it tries to infer which module space the function pointer is pointing into. And if there's version information in that module, it pulls that out too



```
IAT    \WINDOWS\system32\DRIVERS\PCIIDEX.SYS[HAL.dll!WRITE_PORT_ULONG]            [F978E22E] sptd.sys
IAT    \WINDOWS\system32\DRIVERS\PCIIDEX.SYS[HAL.dll!READ_PORT_UCHAR]             [F978D71C] sptd.sys
IAT    \WINDOWS\system32\DRIVERS\PCIIDEX.SYS[HAL.dll!WRITE_PORT_UCHAR]            [F978DF0E] sptd.sys
IAT    atapi.sys[HAL.dll!READ_PORT_UCHAR]                                         [F978D71C] sptd.sys
IAT    atapi.sys[HAL.dll!READ_PORT_BUFFER_USHORT]                                 [F978D910] sptd.sys
IAT    atapi.sys[HAL.dll!READ_PORT_USHORT]                                        [F978D852] sptd.sys
IAT    atapi.sys[HAL.dll!WRITE_PORT_BUFFER_USHORT]                                [F978E0EC] sptd.sys
IAT    atapi.sys[HAL.dll!WRITE_PORT_UCHAR]                                        [F978DF0E] sptd.sys
IAT    \SystemRoot\System32\DRIVERS\i8042prt.sys[HAL.dll!READ_PORT_UCHAR]         [F97A1CE8] sptd.sys
IAT    \SystemRoot\system32\DRIVERS\raspppoe.sys[NDIS.SYS!NdisRegisterProtocol]   [F1013672] \SystemRoot\System32\vsdatant.sys [ZoneAlarm Firewalli...
IAT    \SystemRoot\system32\DRIVERS\raspppoe.sys[NDIS.SYS!NdisOpenAdapter]        [F10134C8] \SystemRoot\System32\vsdatant.sys [ZoneAlarm Firewalli...
IAT    \SystemRoot\system32\DRIVERS\raspppoe.sys[NDIS.SYS!NdisCloseAdapter]       [F1013CBA] \SystemRoot\System32\vsdatant.sys [ZoneAlarm Firewalli...
IAT    \SystemRoot\system32\DRIVERS\raspppoe.sys[NDIS.SYS!NdisDeregisterProtocol] [F1011C2A] \SystemRoot\System32\vsdatant.sys [ZoneAlarm Firewalli...
IAT    \SystemRoot\system32\DRIVERS\psched.sys[NDIS.SYS!NdisDeregisterProtocol]   [F1011C2A] \SystemRoot\System32\vsdatant.sys [ZoneAlarm Firewalli...
IAT    \SystemRoot\system32\DRIVERS\psched.sys[NDIS.SYS!NdisRegisterProtocol]     [F1013672] \SystemRoot\System32\vsdatant.sys [ZoneAlarm Firewalli...
```

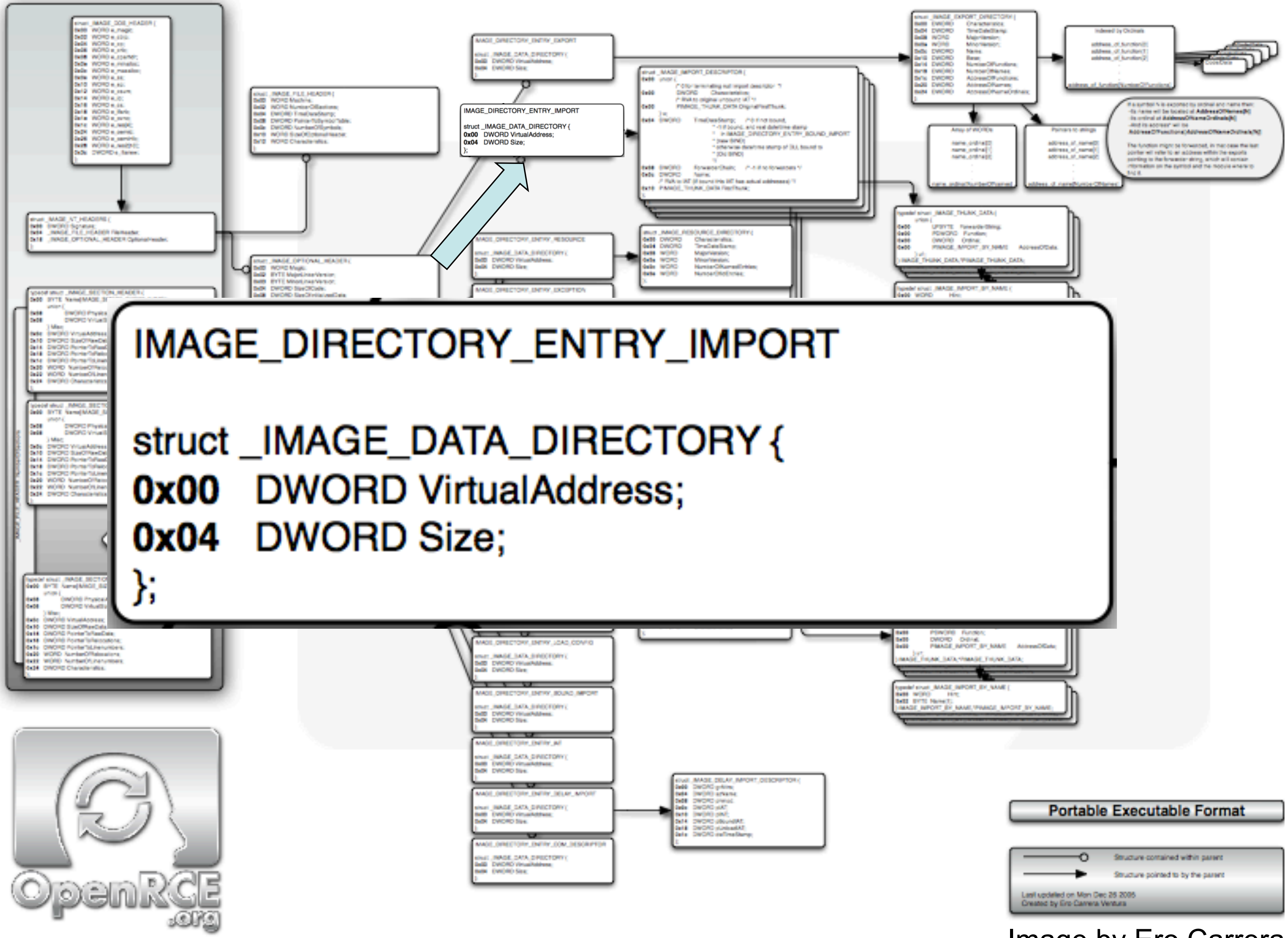
This is the module doing the importing

This is the function being imported by the first module and exported by the second

Telling you that this is an IAT hook

This is the module doing the exporting

Book page 265

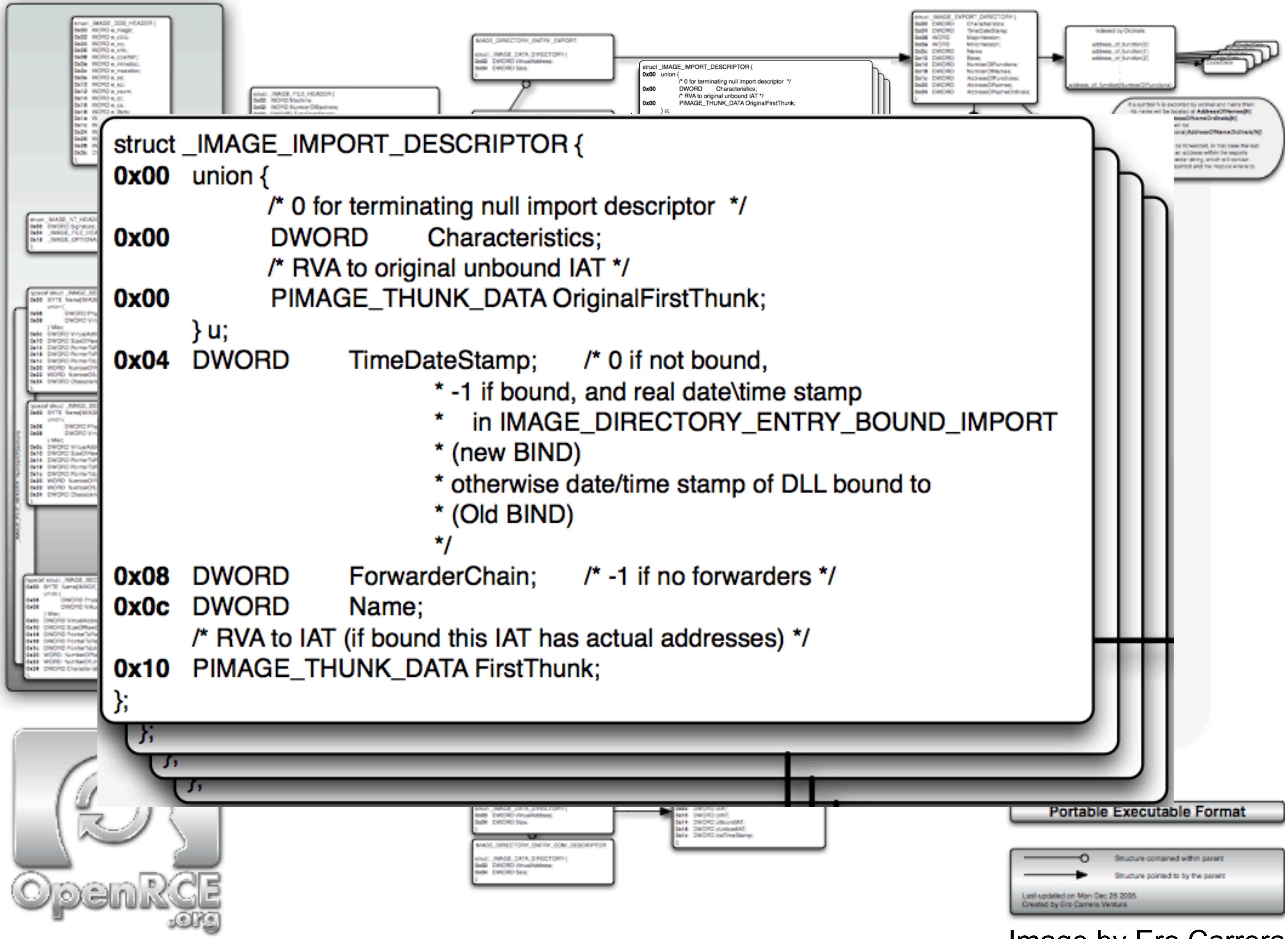


IMAGE_DIRECTORY_ENTRY_EXPORT
IMAGE_DIRECTORY_ENTRY_IMPORT
struct _IMAGE_DATA_DIRECTORY {
0x00 DWORD VirtualAddress;
0x04 DWORD Size;
};
OpenRCE.org
Portable Executable Format
Structure contained within parent
Structure pointed to by the parent
Last updated on Mon Dec 26 2005
Created by Ero Carrera Ventura

```
IMAGE_DIRECTORY_ENTRY_IMPORT

struct _IMAGE_DATA_DIRECTORY {
0x00   DWORD VirtualAddress;
0x04   DWORD Size;
};
```

Image by Ero Carrera



```
struct _IMAGE_IMPORT_DESCRIPTOR {
0x00   union {
                 /* 0 for terminating null import descriptor  */
0x00             DWORD       Characteristics;
                 /* RVA to original unbound IAT */
0x00             PIMAGE_THUNK_DATA OriginalFirstThunk;
       } u;
0x04   DWORD        TimeDateStamp;      /* 0 if not bound,
                                  * -1 if bound, and real date\time stamp
                                  *    in IMAGE_DIRECTORY_ENTRY_BOUND_IMPORT
                                  * (new BIND)
                                  * otherwise date/time stamp of DLL bound to
                                  * (Old BIND)
                                  */
0x08   DWORD        ForwarderChain;     /* -1 if no forwarders */
0x0c   DWORD        Name;
       /* RVA to IAT (if bound this IAT has actual addresses) */
0x10   PIMAGE_THUNK_DATA FirstThunk;
};
```

Portable Executable Format
Structure contained within parent
Structure pointed to by the parent
Last updated on Mon Dec 26 2005
Created by Ero Carrera Ventura
OpenRCE.org

Image by Ero Carrera

Review: Import Descriptor

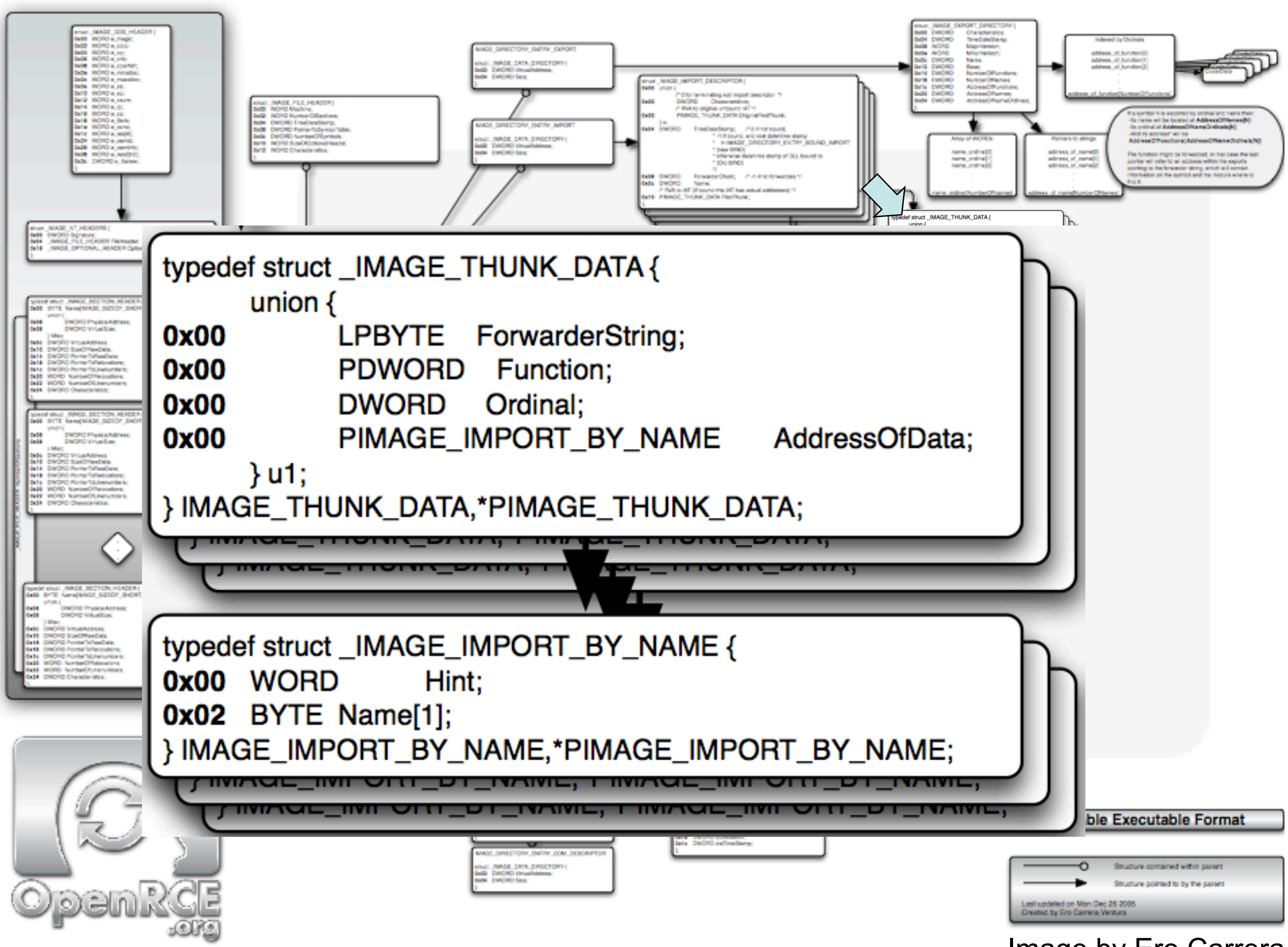
(from winnt.h)

I think they meant “INT”

```
typedef struct _IMAGE_IMPORT_DESCRIPTOR {
    union {
        DWORD   Characteristics;            // 0 for terminating null import descriptor
        DWORD   OriginalFirstThunk;         // RVA to original unbound IAT (PIMAGE_THUNK_DATA)
                                            //Xeno Comment: In reality a PIMAGE_THUNK_DATA
    };
    DWORD   TimeDateStamp;                  // 0 if not bound,
                                            // -1 if bound, and real date\time stamp
                                            //    in IMAGE_DIRECTORY_ENTRY_BOUND_IMPORT (new BIND)
                                            // O.W. date/time stamp of DLL bound to (Old BIND)

    DWORD   ForwarderChain;                 // -1 if no forwarders
    DWORD   Name;
    DWORD   FirstThunk;                     // RVA to IAT (if bound this IAT has actual addresses)
                                            //Xeno Comment: In reality a PIMAGE_THUNK_DATA
} IMAGE_IMPORT_DESCRIPTOR;
```

- While the things in blue are the fields filled in for the most common case, we will actually have to understand everything for this structure, because you could run into all the variations.



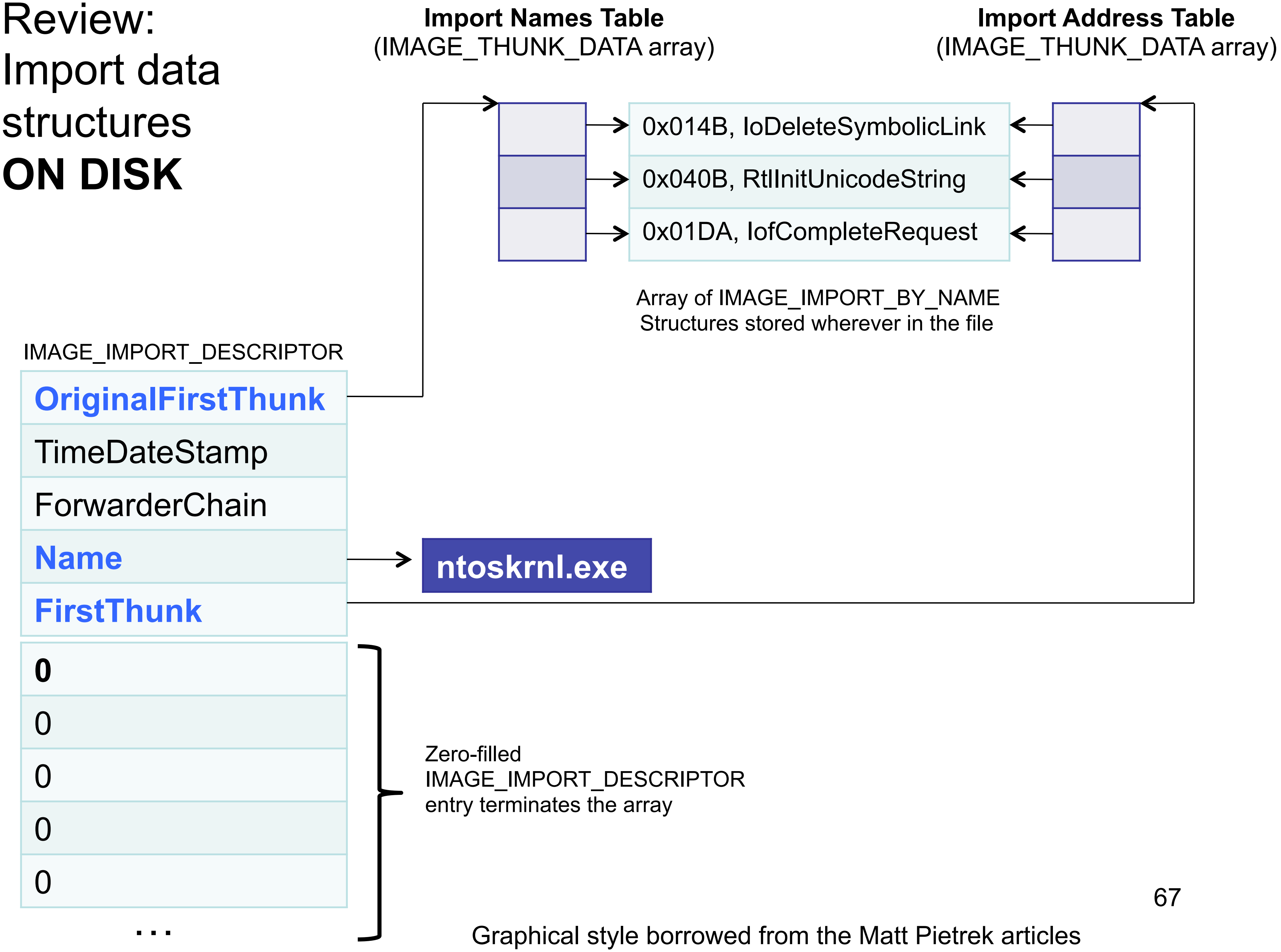
ble Executable Format
Structure contained within parent
Structure pointed to by the parent
Last updated on Mon Dec 26 2005
Created by Ero Carrera Ventura
OpenRCE.org

```
typedef struct _IMAGE_THUNK_DATA {
          union {
0x00              LPBYTE    ForwarderString;
0x00              PDWORD    Function;
0x00              DWORD     Ordinal;
0x00              PIMAGE_IMPORT_BY_NAME       AddressOfData;
          } u1;
} IMAGE_THUNK_DATA,*PIMAGE_THUNK_DATA;
```

```
typedef struct _IMAGE_IMPORT_BY_NAME {
0x00   WORD         Hint;
0x02   BYTE  Name[1];
} IMAGE_IMPORT_BY_NAME,*PIMAGE_IMPORT_BY_NAME;
```

Image by Ero Carrera

Review: Import data structures **ON DISK**



Import Names Table
(IMAGE_THUNK_DATA array)

Import Address Table
(IMAGE_THUNK_DATA array)

0x014B, IoDeleteSymbolicLink

0x040B, RtlInitUnicodeString

0x01DA, IofCompleteRequest

Array of IMAGE_IMPORT_BY_NAME
Structures stored wherever in the file

IMAGE_IMPORT_DESCRIPTOR

OriginalFirstThunk

TimeDateStamp

ForwarderChain

Name

FirstThunk

ntoskrnl.exe

0

0

0

0

0

. . .

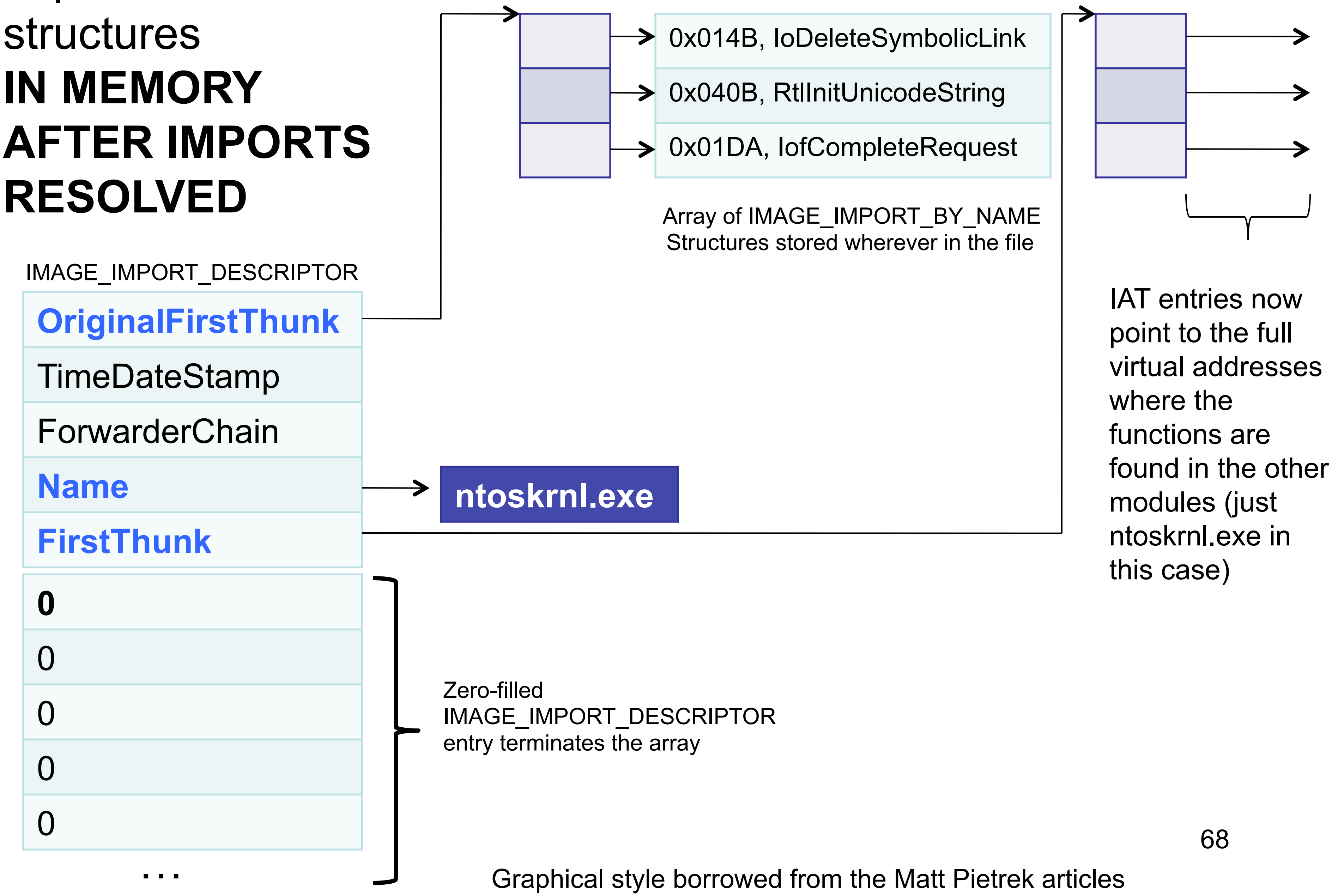
Zero-filled
IMAGE_IMPORT_DESCRIPTOR
entry terminates the array

Graphical style borrowed from the Matt Pietrek articles

Review: Import data structures **IN MEMORY AFTER IMPORTS RESOLVED**

Import Names Table
(IMAGE_THUNK_DATA array)

Import Address Table
(IMAGE_THUNK_DATA array)



0x014B, IoDeleteSymbolicLink

0x040B, RtlInitUnicodeString

0x01DA, IofCompleteRequest

Array of IMAGE_IMPORT_BY_NAME
Structures stored wherever in the file

IMAGE_IMPORT_DESCRIPTOR

OriginalFirstThunk
TimeDateStamp
ForwarderChain
Name
FirstThunk
0
0
0
0
0

…

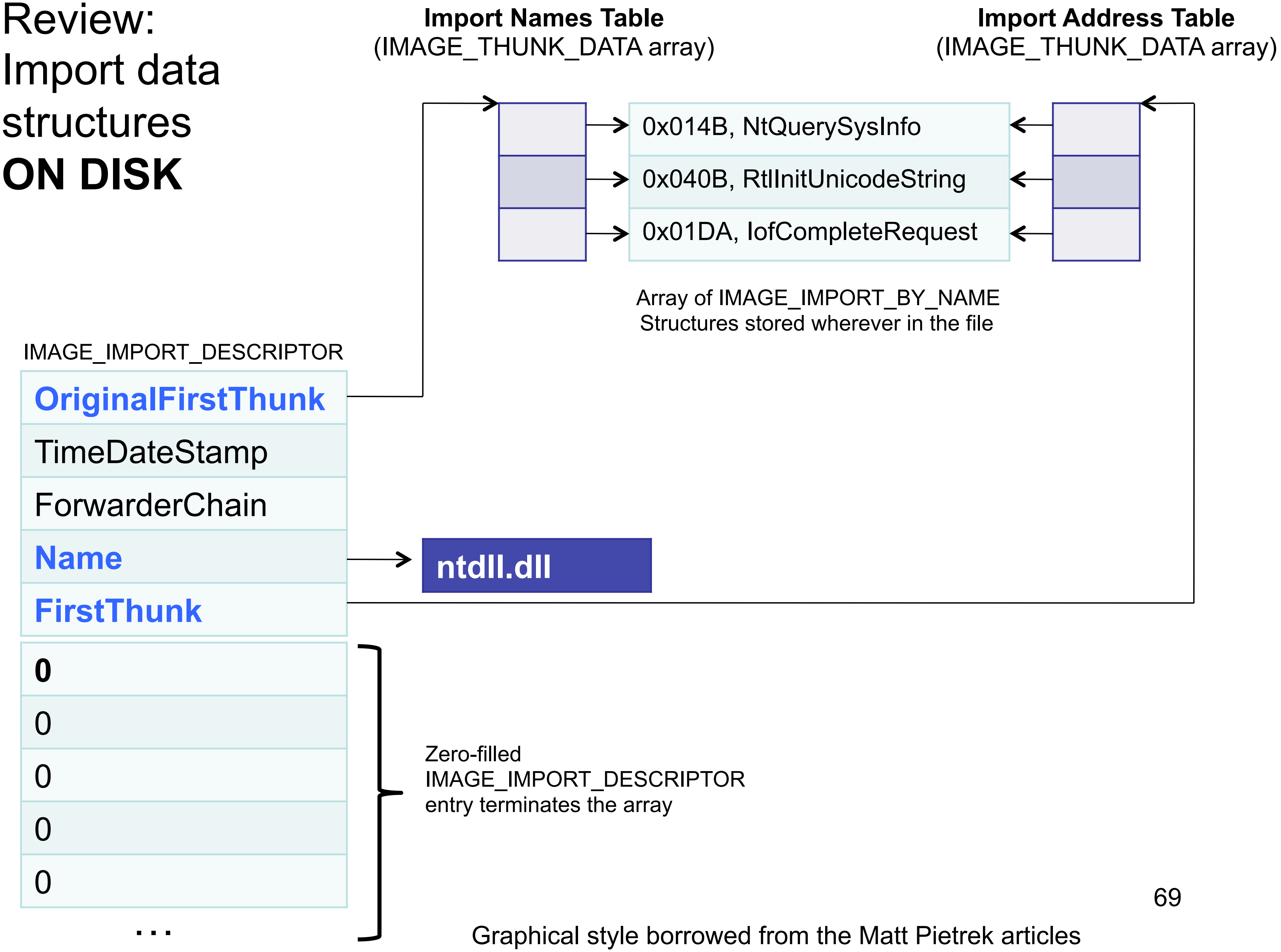
ntoskrnl.exe

IAT entries now point to the full virtual addresses where the functions are found in the other modules (just ntoskrnl.exe in this case)

Zero-filled
IMAGE_IMPORT_DESCRIPTOR
entry terminates the array

Graphical style borrowed from the Matt Pietrek articles

Review: Import data structures **ON DISK**



Import Names Table
(IMAGE_THUNK_DATA array)

Import Address Table
(IMAGE_THUNK_DATA array)

0x014B, NtQuerySysInfo

0x040B, RtlInitUnicodeString

0x01DA, IofCompleteRequest

Array of IMAGE_IMPORT_BY_NAME
Structures stored wherever in the file

IMAGE_IMPORT_DESCRIPTOR

OriginalFirstThunk
TimeDateStamp
ForwarderChain
Name
FirstThunk

ntdll.dll

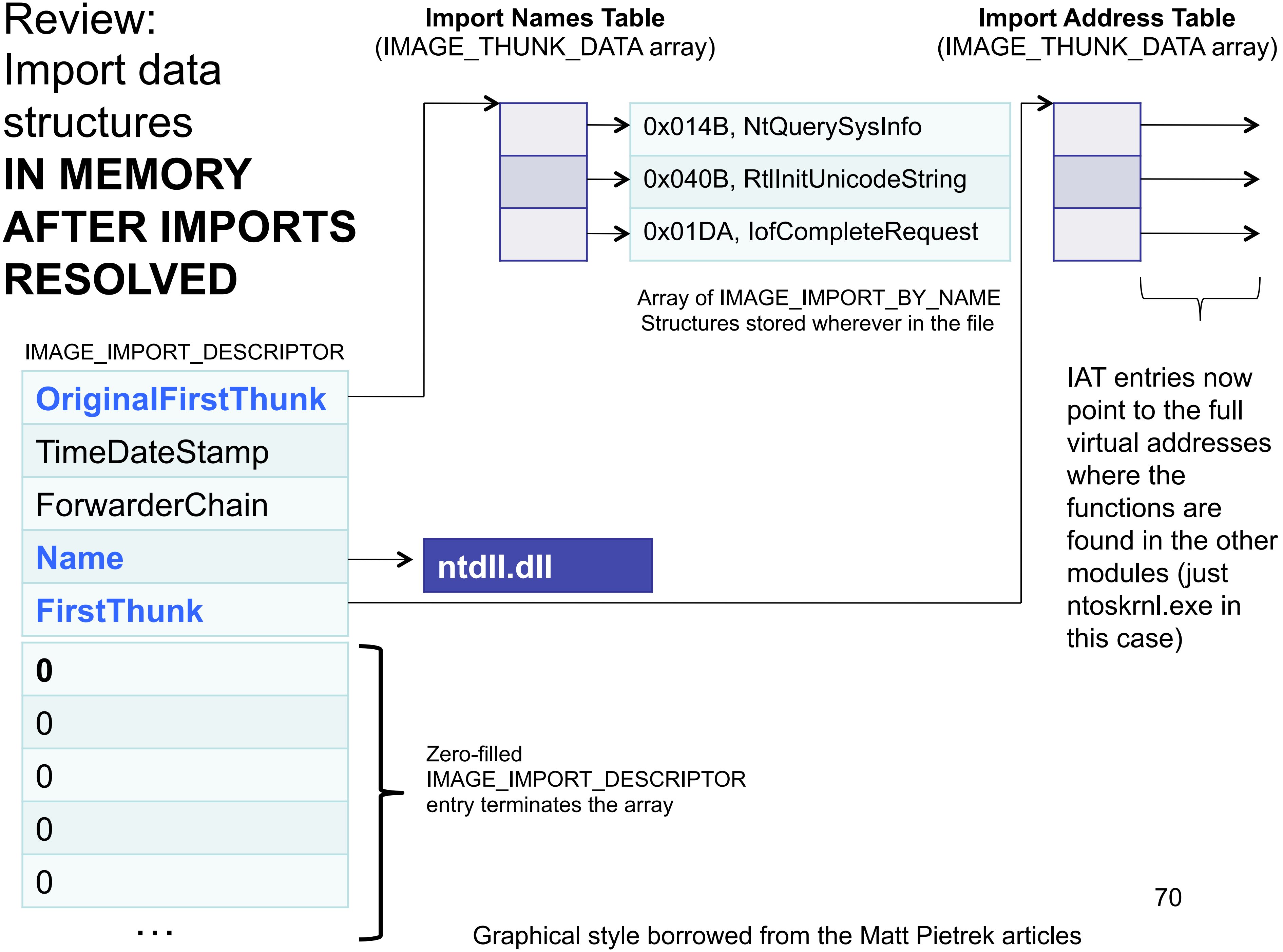
0
0
0
0
0

. . .

Zero-filled
IMAGE_IMPORT_DESCRIPTOR
entry terminates the array

Graphical style borrowed from the Matt Pietrek articles

Review: Import data structures **IN MEMORY AFTER IMPORTS RESOLVED**



Import Names Table
(IMAGE_THUNK_DATA array)

Import Address Table
(IMAGE_THUNK_DATA array)

0x014B, NtQuerySysInfo

0x040B, RtlInitUnicodeString

0x01DA, IofCompleteRequest

Array of IMAGE_IMPORT_BY_NAME
Structures stored wherever in the file

IAT entries now point to the full virtual addresses where the functions are found in the other modules (just ntoskrnl.exe in this case)

IMAGE_IMPORT_DESCRIPTOR

OriginalFirstThunk
TimeDateStamp
ForwarderChain
Name
FirstThunk
0
0
0
0
0

. . .

ntdll.dll

Zero-filled
IMAGE_IMPORT_DESCRIPTOR
entry terminates the array

Graphical style borrowed from the Matt Pietrek articles

Review: IAT Hooking

- When the IAT is fully resolved, it is basically an array of function pointers. Somewhere, in some code path, there’s something which is going to take an IAT address, and use whatever’s in that memory location as the destination of the code it should call.
- What if the “whatever’s in that memory location” gets changed after the OS loader is done? What if it points at attacker code?

Review: IAT Hooking 2

- Well, that would mean the attacker's code would functionally be "man-in-the-middle"ing the call to the function. He can then change parameters before forwarding the call on to the original function, and filter results that come back from the function, or simply never call the original function, and send back whatever status he pleases.
  - Think rootkits. Say you're calling OpenFile. It looks at the file name and if you're asking for a file it wants to hide, it simply returns "no file found."
- But how does the attacker change the IAT entries? This is a question of assumptions about where the attacker is.

Review: IAT Hooking 3

- In a traditional memory-corrupting exploit, the attacker is, by definition, in the memory space of the attacked process, upon successfully gaining arbitrary code execution. The attacker can now change memory such as the IAT for this process only, because remember (from OS class or Intermediate x86) each process has a separate memory space.
- If the attacker wants to change the IAT on other processes, he must be in their memory spaces as well. Typically the attacker will format some of his code as a DLL and then perform “DLL Injection” in order to get his code in other process’ memory space.
- The ability to do something like DLL injection is generally a prerequisite in order to leverage IAT hooking across many userspace processes. In the kernel, kernel modules are generally all sharing the same memory space with the kernel, and therefore one subverted kernel module can hook the IAT of any other modules that it wants.

Review: DLL Injection

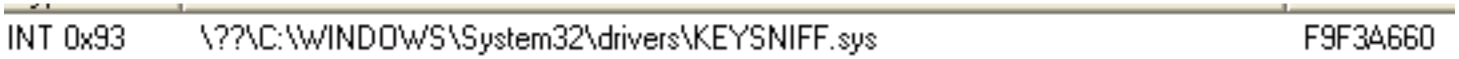
- See http://en.wikipedia.org/wiki/DLL_injection for more ways that this can be achieved on Windows/*nix
- We’ re going to use the AppInit_DLLs way of doing this, out of laziness
- (Note: AppInit_DLLs' behavior has changed in releases > XP, it now has to be enabled with Administrator level permissions.)

Review: Lab: IAT hooking

- http://www.codeproject.com/KB/vista/api-hooks.aspx
  - This will hook NtQuerySystemInformation(), which is what taskmgr.exe uses in order to list the currently running processes. It will replace this with HookedNtQuerySystemInformation(), which will hide calc.exe
  - I modified that code to use IAT hooking rather than inline (which is much simpler actually)
- Steps:
  - Compile AppInitHookIAT.dll
  - Place at C:\AppInitHookIAT.dll for simplicity
  - Use regedit.exe to add C:\AppInitHookIAT.dll as the value for the key **HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT \CurrentVersion\Windows\AppInit_DLLs** (if there is already something there, separate the entries with a comma)
  - Start calc.exe, start taskmgr.exe, confirm that calc.exe doesn't show up in the list of running processes.
  - Remove C:\AppInitHookIAT.dll from AppInit_DLLs and restart taskmgr.exe.
  - Confirm calc.exe shows up in the list of running processes.
  - (This is a basic "userspace rootkit" technique. Because of this, all entries in this registry key should always be looked upon with suspicion.)

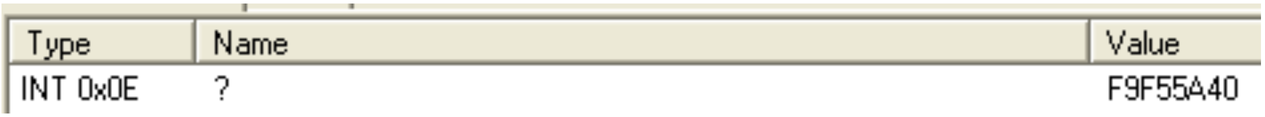
Go with what you know: IDT

If we had run the bhwin_keysniff from IntermediateX86 we would have seen the following:



INT 0x93	\??\C:\WINDOWS\System32\drivers\KEYSNIFF.sys	F9F3A660

As it is, we see something like:



Type	Name	Value
INT 0x0E	?	F9F55A40

This indicates that interrupt index 0xE in the Interrupt Descriptor Table (IDT) does not point as its normal location, it points at memory address 0xF9F55A40, and GMER has not been able to determine which driver, if any, is associated with that memory range (thanks to another rootkit we'll learn about later.)

Let's do a quick review of what we learned about segmentation and the IDT.

Book page 270

Review: Surprise! No one uses segmentation directly for memory protection! :D

- On most systems, segmentation is not providing the primary RWX type permissions, they instead rely on paging protections.

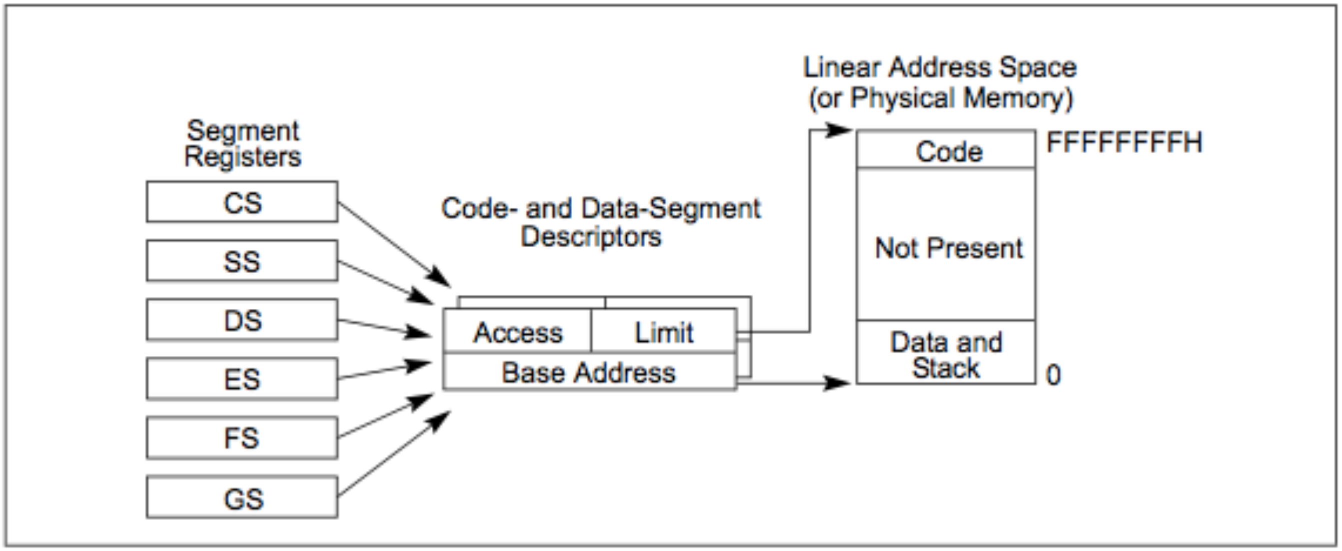


Figure 3-2. Flat Model

Vol.3a, Sect. 3.2.1

Review: One more time

One of the segment registers
(SS/CS/DS/ES/FS/GS)

The address you see in
assembly instructions
(implicitly with a CS or SS selector)

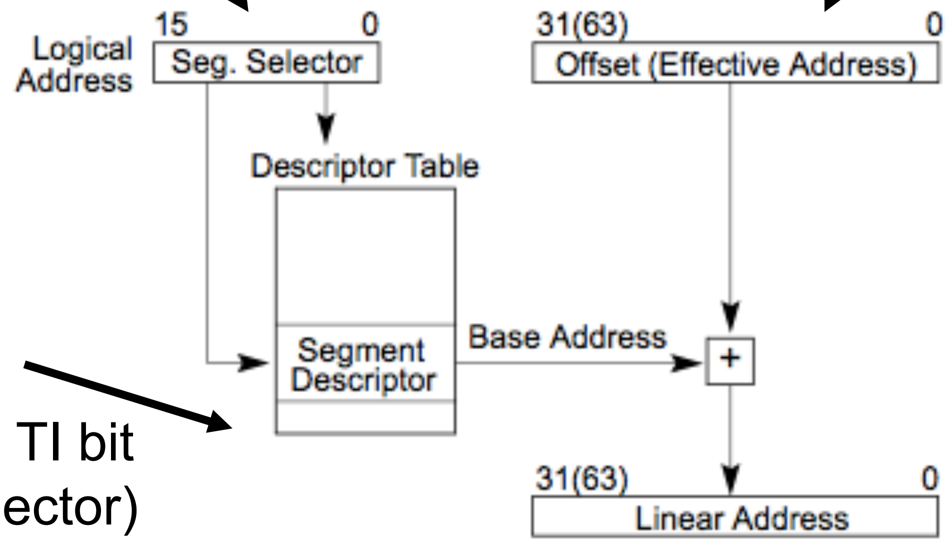
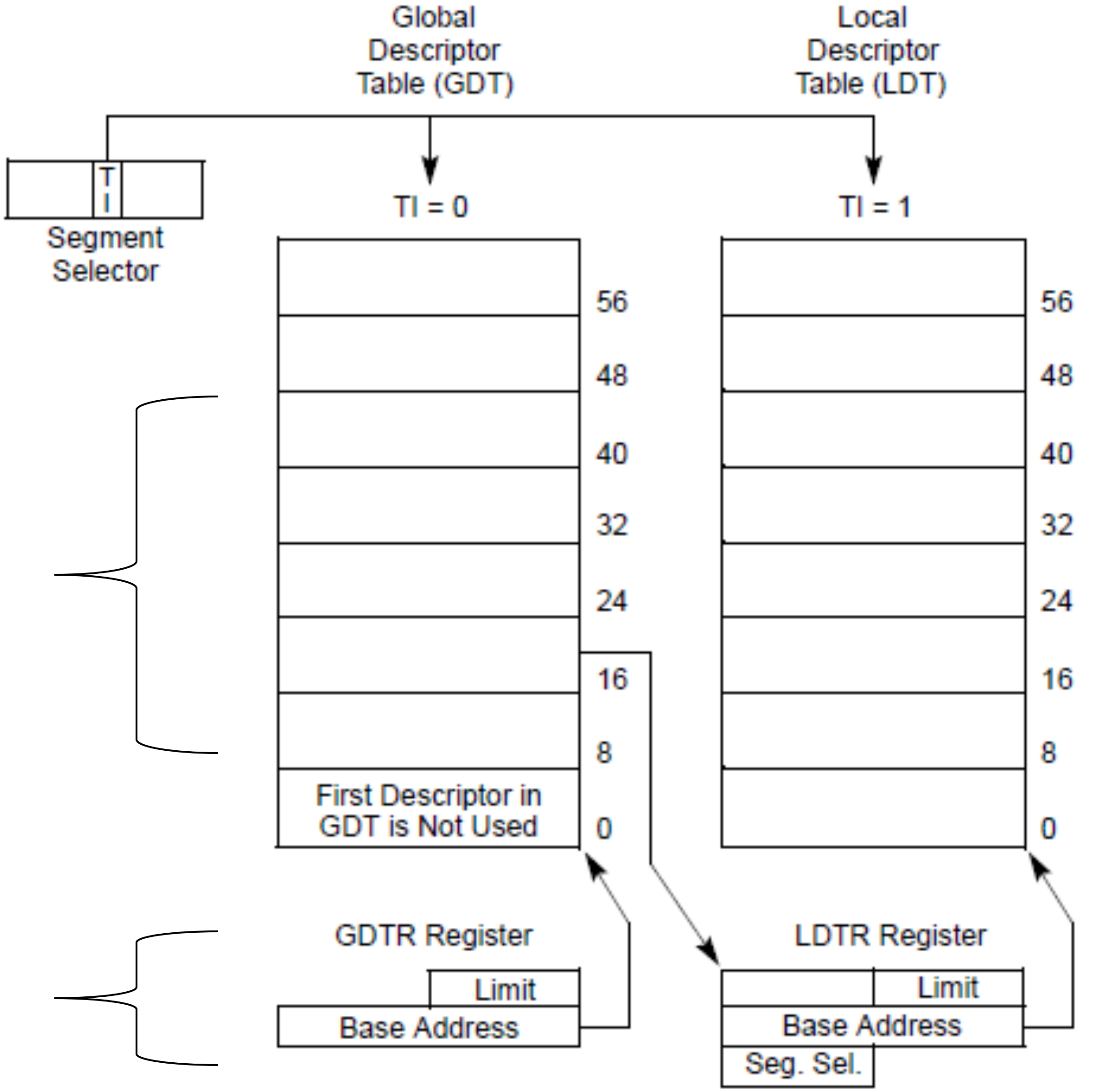


Figure 3-5. Logical Address to Linear Address Translation

GDT or LDT
(depending on the TI bit
of the segment selector)

Review: GDT & LDT



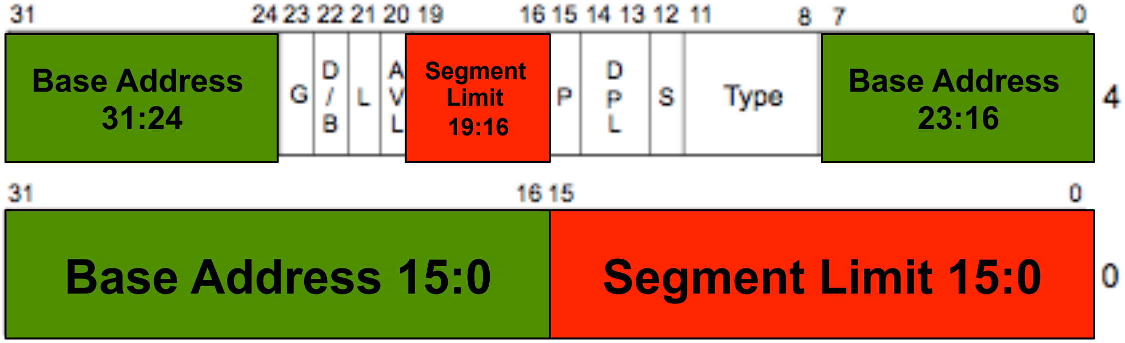
All entries in these tables are "Segment Descriptor" structures

Special registers point to the base of the tables & specify their size

Figure 3-10. Global and Local Descriptor Tables

Review: Segment Descriptors

- “Each segment has a segment descriptor, which specifies the size of the segment, the access rights and privilege level for the, the segment type, and the location of the first byte of the segment in the linear address space (called the base address of the segment).”



L — 64-bit code segment (IA-32e mode only)
AVL — Available for use by system software
BASE — Segment base address
D/B — Default operation size (0 = 16-bit segment; 1 = 32-bit segment)
DPL — Descriptor privilege level
G — Granularity
LIMIT — Segment Limit
P — Segment present
S — Descriptor type (0 = system; 1 = code or data)
TYPE — Segment type

Figure 3-8. Segment Descriptor

I approve of this summary

Review: IDTR Usage

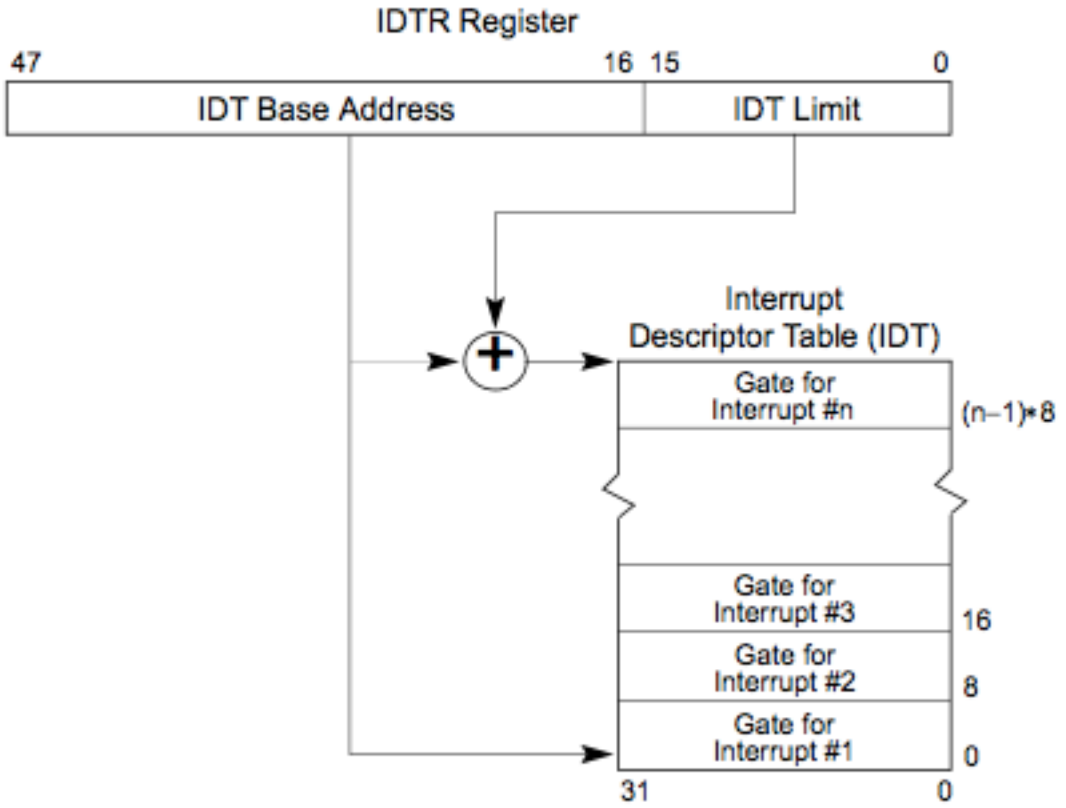
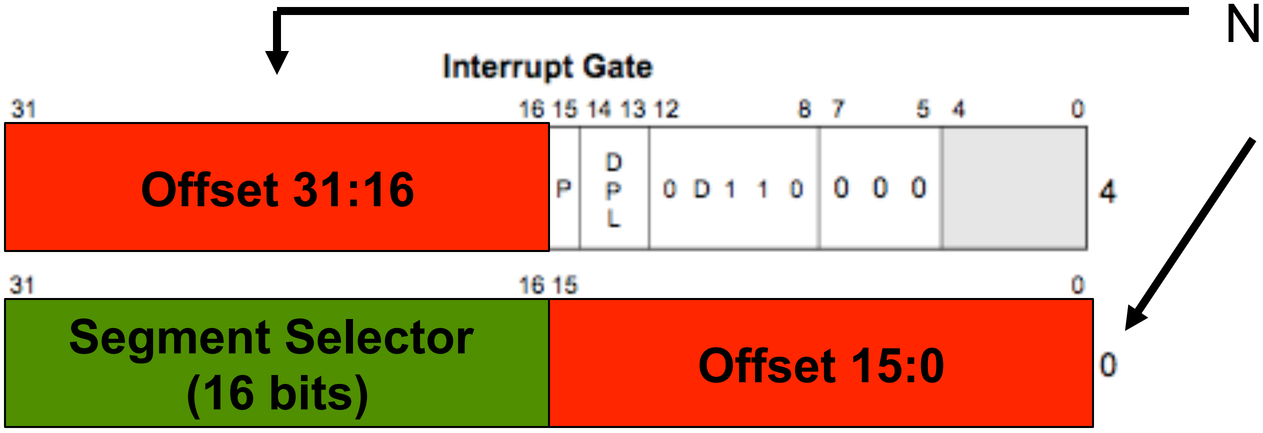


Figure 5-1. Relationship of the IDTR and IDT

Review: Interrupt Gate Descriptor



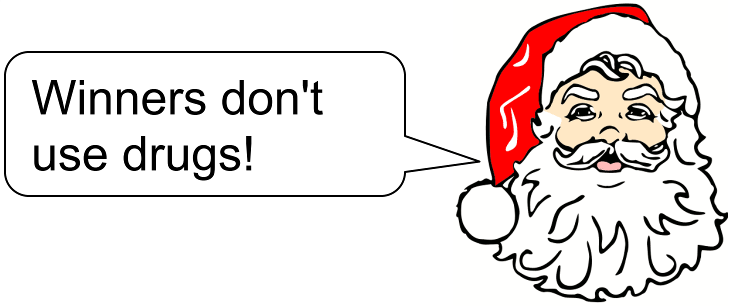
Note that the two halves of the offset form a 32 bit address.

DPL Descriptor Privilege Level
Offset Offset to procedure entry point
P Segment Present flag
Selector Segment Selector for destination code segment
D Size of gate: 1 = 32 bits; 0 = 16 bits

▨ Reserved

Descriptors not in use should have P = 0

Figure 5-2. IDT Gate Descriptors



From IDT to Interrupt Handler

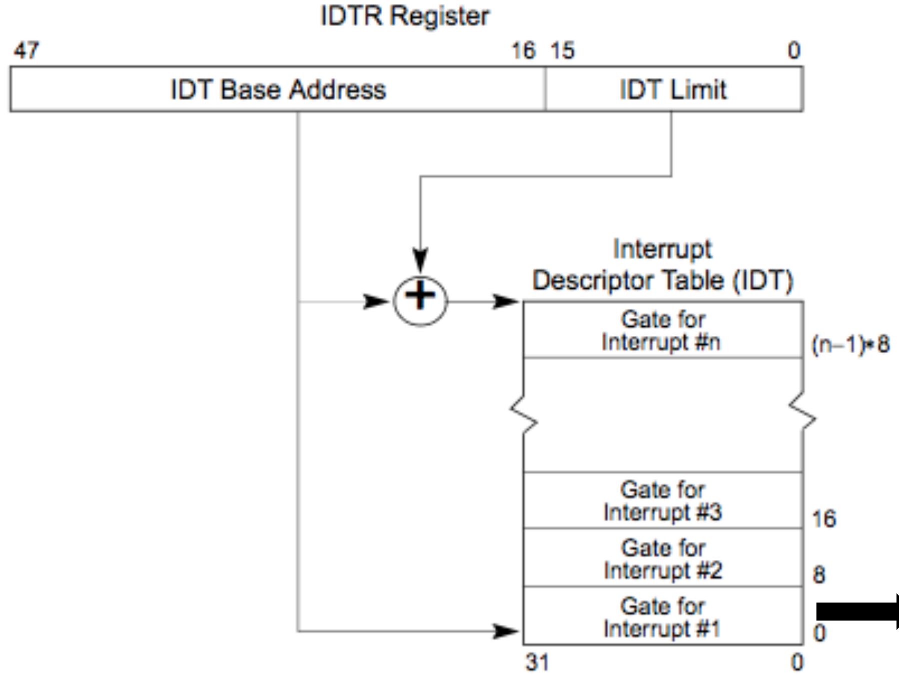


Figure 5-1. Relationship of the IDTR and IDT

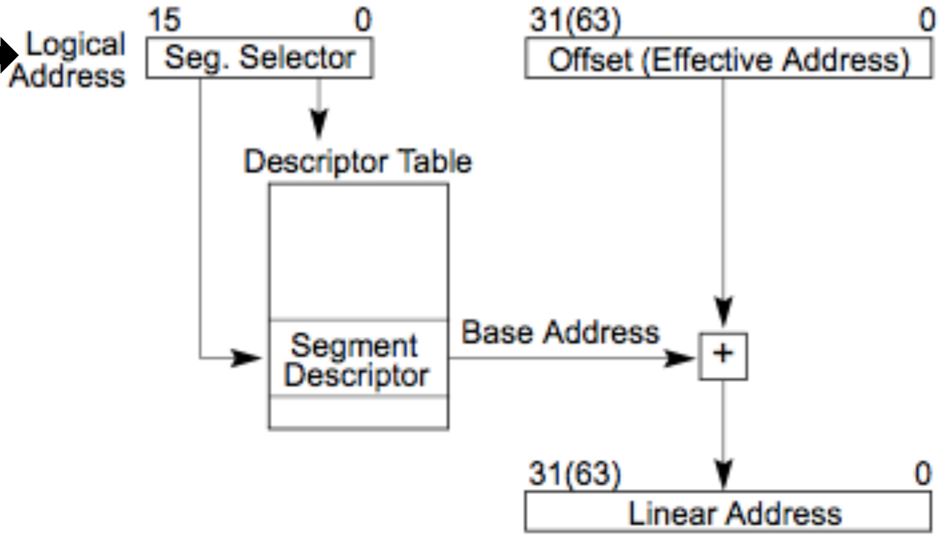


Figure 3-5. Logical Address to Linear Address Translation

Review: IDT Relation to Segments

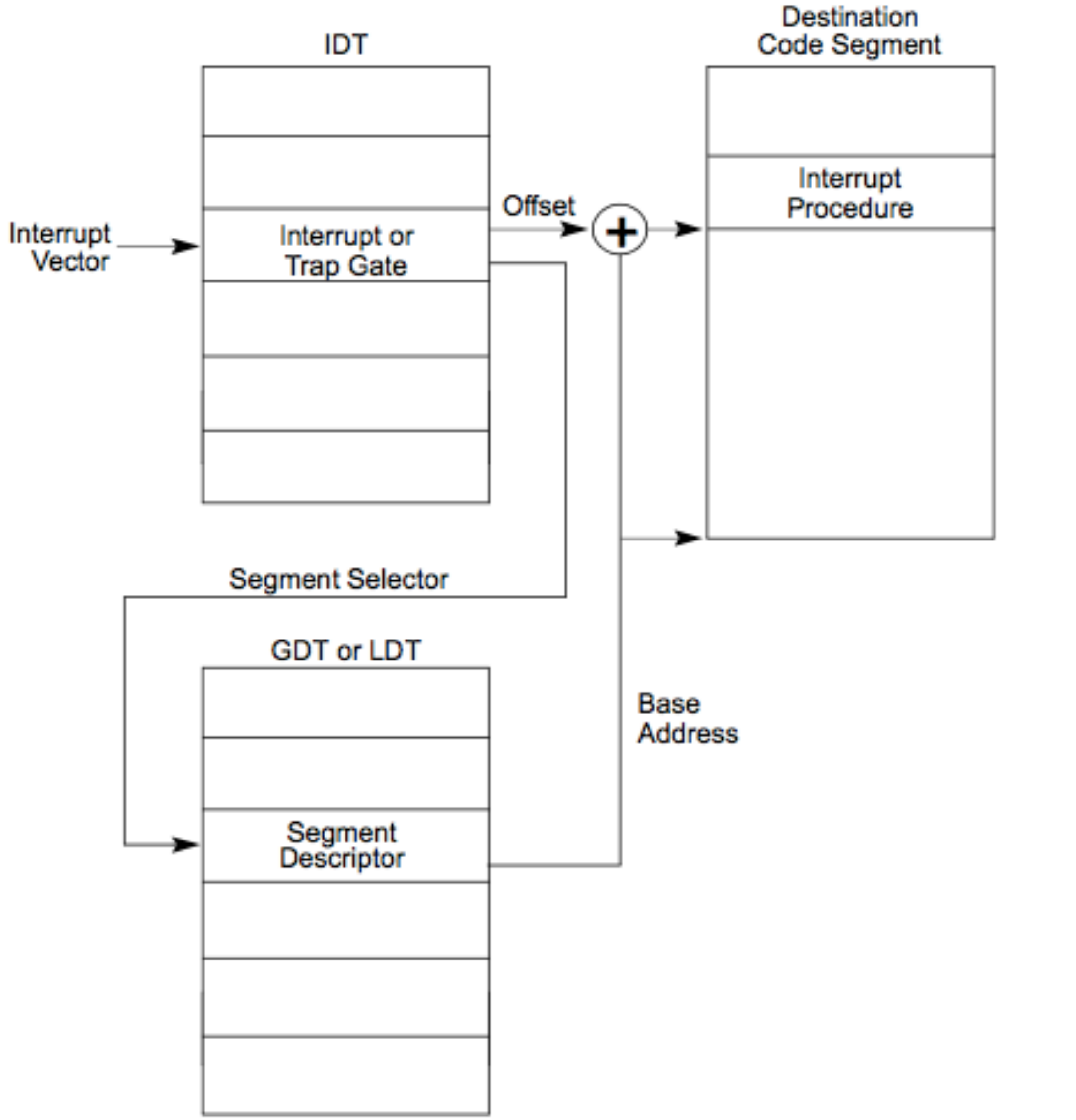
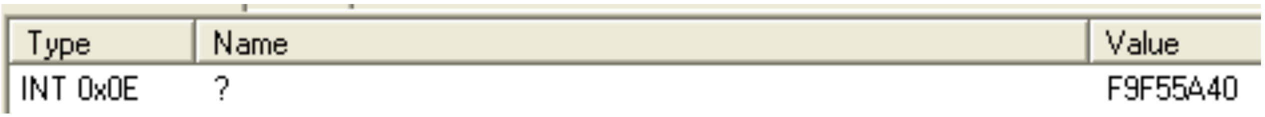


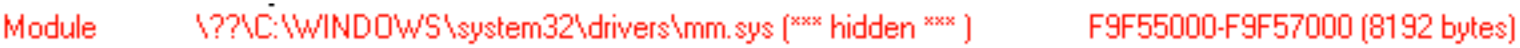
Figure 5-3. Interrupt Procedure Call

A hint



Type	Name	Value
INT 0x0E	?	F9F55A40

+



Module \??\C:\WINDOWS\system32\drivers\mm.sys (*** hidden ***) F9F55000-F9F57000 (8192 bytes)

+

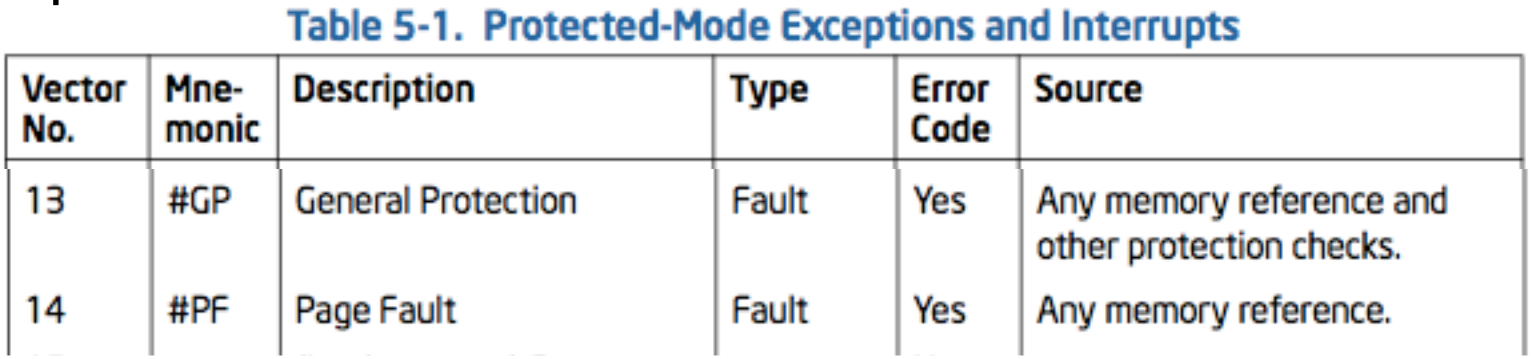


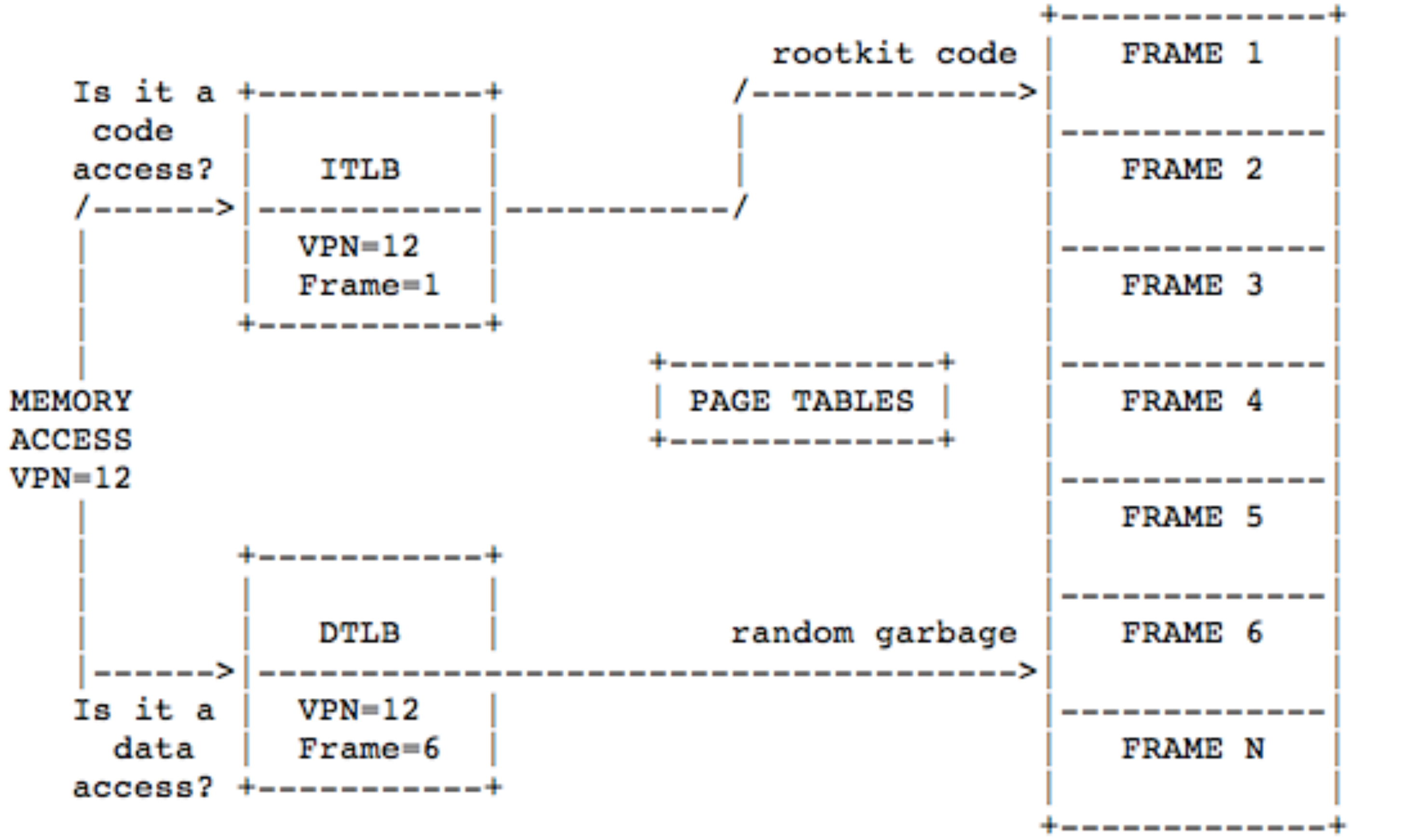
Table 5-1. Protected-Mode Exceptions and Interrupts

Vector No.	Mne-monic	Description	Type	Error Code	Source
13	#GP	General Protection	Fault	Yes	Any memory reference and other protection checks.
14	#PF	Page Fault	Fault	Yes	Any memory reference.

=

The IDT change seems to be due to a module called mm.sys which hooks the Page Fault handler… Hmm…who do we know that might want to do that…

Review: ASCII Art of Doooom!



```
                                                  +-------------+
                                     rootkit code |   FRAME 1   |
   Is it a +-----------+           /------------->|             |
    code   |           |           |              |-------------|
   access? |   ITLB    |           |              |   FRAME 2   |
   /------>|-----------|-----------/              |             |
   |       |   VPN=12  |                          |-------------|
   |       |   Frame=1 |                          |   FRAME 3   |
   |       +-----------+                          |             |
   |                           +-------------+    |-------------|
MEMORY                         | PAGE TABLES |    |   FRAME 4   |
ACCESS                         +-------------+    |             |
VPN=12                                            |-------------|
   |                                              |   FRAME 5   |
   |       +-----------+                          |             |
   |       |           |                          |-------------|
   |       |   DTLB    |           random garbage |   FRAME 6   |
   |------>|------------------------------------->|             |
   Is it a |   VPN=12  |                          |-------------|
     data  |   Frame=6 |                          |   FRAME N   |
   access? +-----------+                          |             |
                                                  +-------------+
```

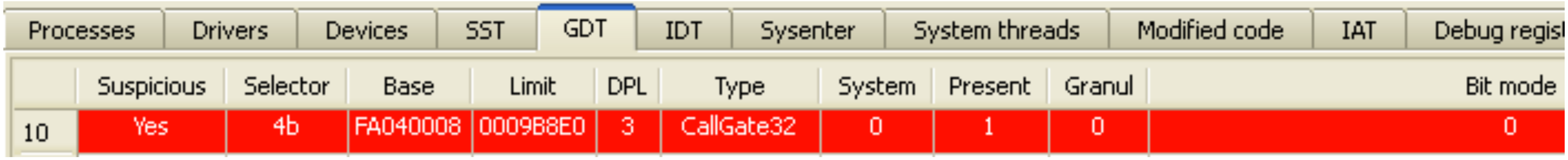
[Figure 5 - Faking Read / Writes by Desynchronizing the Split TLB]

Book page 516

http://www.phrack.com/issues.html?issue=63&id=8

Missed one!

- Turns out the GDT is modified to have a call gate. While you could see this with manual windbg inspection using the !descriptor plugin from the Intermediate x86 class, Tuluka also detects it:



	Suspicious	Selector	Base	Limit	DPL	Type	System	Present	Granul	Bit mode
10	Yes	4b	FA040008	0009B8E0	3	CallGate32	0	1	0	0

- Let's go review call gates quick shall we?

Book page 308

Review: Call Gates

("I'm down with Bill Gates, I call him Money for short. I phone him up at home, and I make him do my tech support!"
- Weird Al, "It's All About the Pentiums")

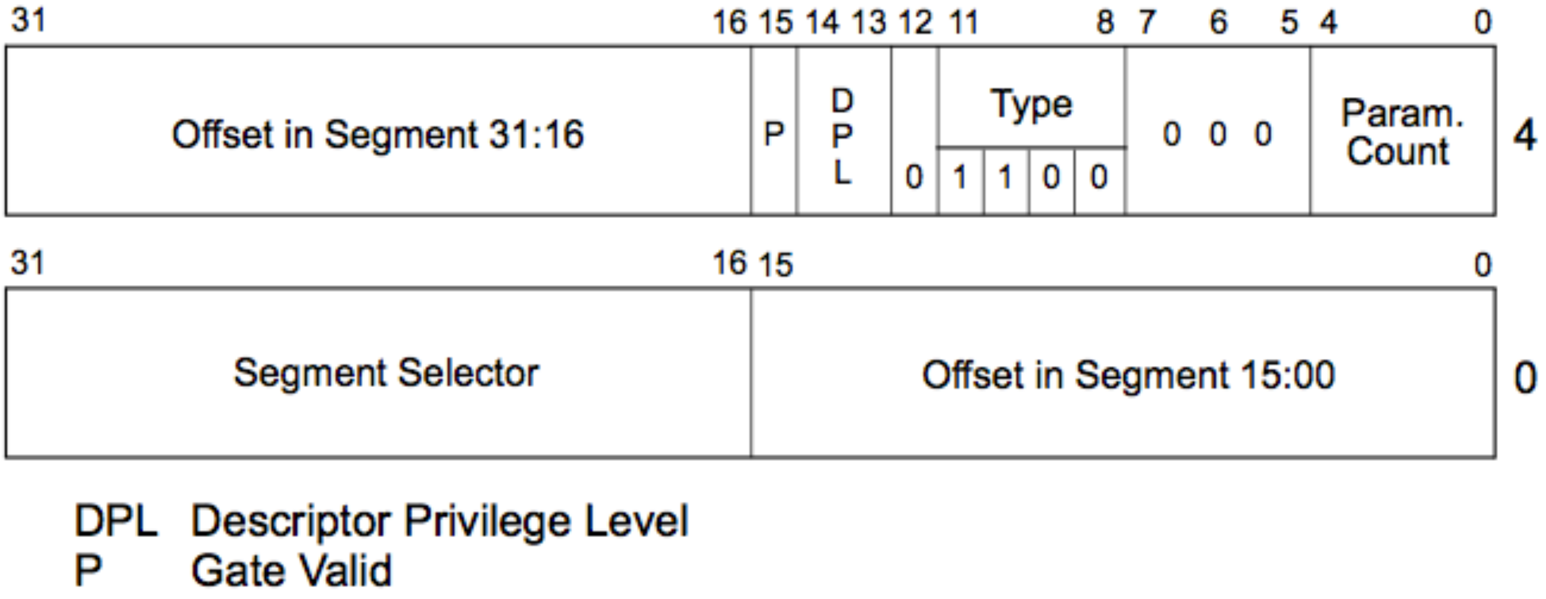


Figure 4-8. Call-Gate Descriptor

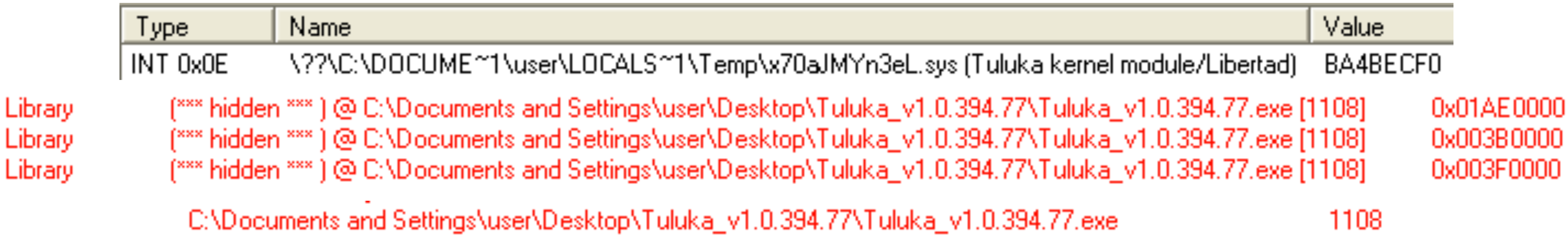
- Call gates are basically a way to transfer control from one segment to another segment (possibly at a different privilege ring, possible at a different size in terms of whether it's 16/32 bits.)
- But the key point is you don't want people to be able to call to anywhere in the other segment, you want the interface to be controlled and well-understood. So calling to a call gate brings code to a specific place which the kernel has set up.

Review: Call Gates 2

- The CALL, RET, and JMP x86 instructions have a special form for when they are doing inter-segment control flow transfer (normal call, ret, jmps are intra-segment for reasons which will become clear shortly.)
- Each of them takes a single far pointer as an argument (though in ret's case, it's popping it off the stack).
- A call gate expects as many parameters as specified by the "Param Count" field on the previous slide (max of 32 due to 5 bit field). Parameters are just pushed onto the stack right to left like a normal cdecl/stdcall calling convention.
- Return value from the far call is returned in eax.
- __asm{call fword ptr 0x48:0x12345678};

Funny thing that…

- Run GMER while Tuluka is loaded, get:



Type	Name	Value
INT 0x0E	\??\C:\DOCUME~1\user\LOCALS~1\Temp\x70aJMYn3eL.sys (Tuluka kernel module/Libertad)	BA4BECF0

```
Library        (*** hidden *** ) @ C:\Documents and Settings\user\Desktop\Tuluka_v1.0.394.77\Tuluka_v1.0.394.77.exe [1108]      0x01AE0000
Library        (*** hidden *** ) @ C:\Documents and Settings\user\Desktop\Tuluka_v1.0.394.77\Tuluka_v1.0.394.77.exe [1108]      0x003B0000
Library        (*** hidden *** ) @ C:\Documents and Settings\user\Desktop\Tuluka_v1.0.394.77\Tuluka_v1.0.394.77.exe [1108]      0x003F0000

                 C:\Documents and Settings\user\Desktop\Tuluka_v1.0.394.77\Tuluka_v1.0.394.77.exe                  1108
```

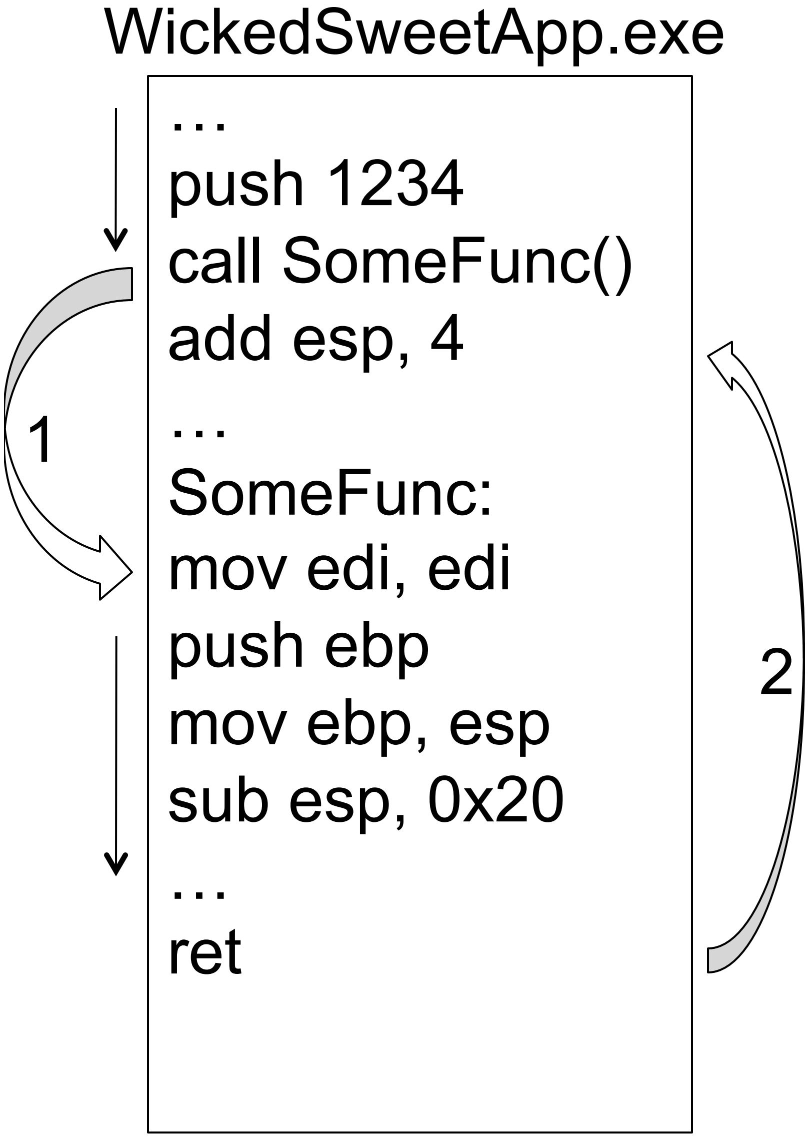
YO DAWG I HEARD YOU LIKE ROOTKITS
SO I PUT A ROOTKIT IN YOUR ROOTKIT DETECTOR SO YOU CAN ROOTKIT WHILE YOU DETECT ROOTKITS

(With thanks to http://memegenerator.net/yo-dawg/ for making that easy!)

A portrait of the rootkit as a young man in the middle

(CC BY-NC-SA 2.0) image by thrill kills sunday pills
http://www.flickr.com/photos/27086700@N03/2994587384/in/photostream/

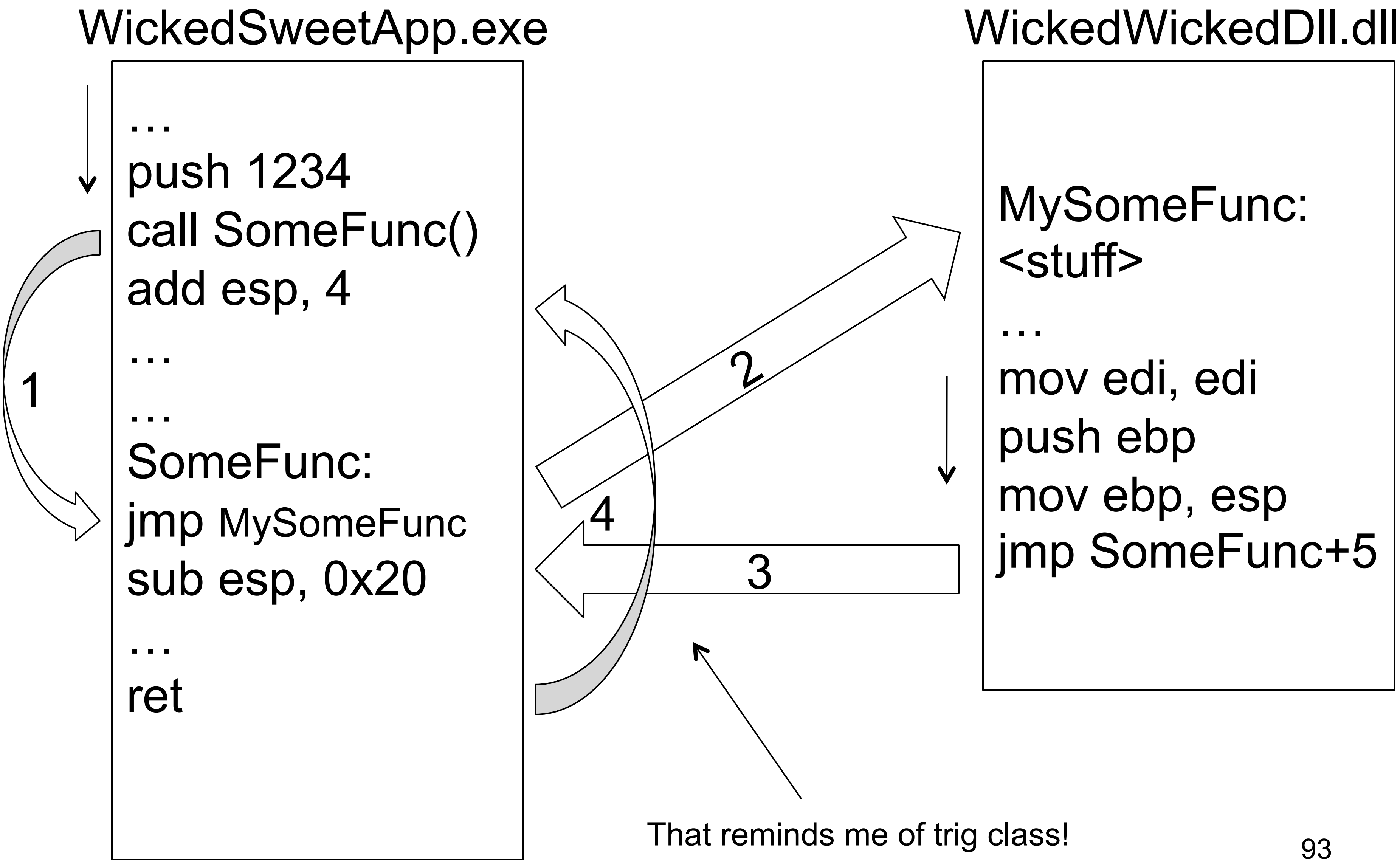
Normal Intra-Module Function Call



WickedSweetApp.exe

```
…
push 1234
call SomeFunc()
add esp, 4
…
SomeFunc:
mov edi, edi
push ebp
mov ebp, esp
sub esp, 0x20
…
ret
```

Inline Hooked Intra-Module Function Call



WickedSweetApp.exe

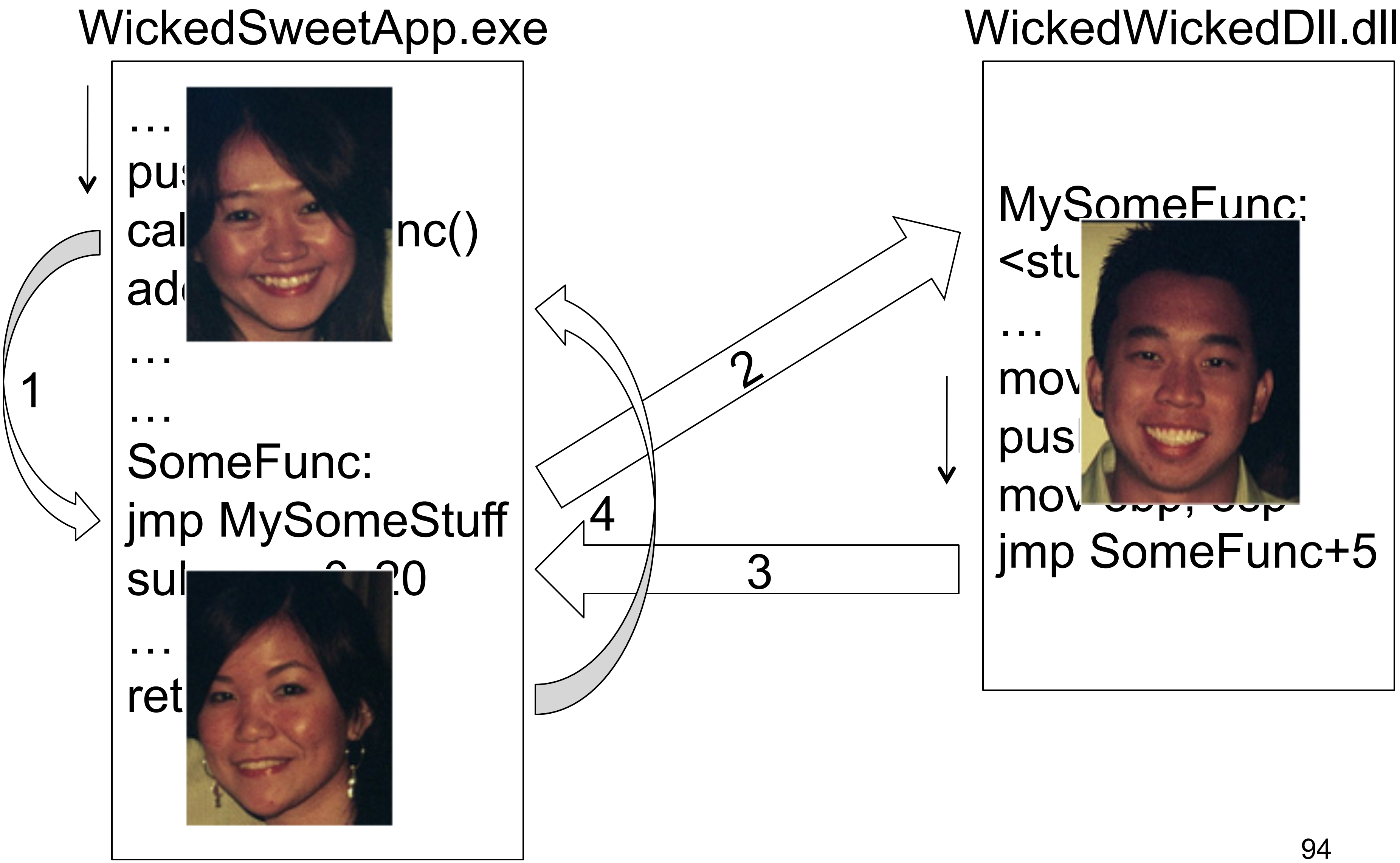
```
…
push 1234
call SomeFunc()
add esp, 4
…
…
SomeFunc:
jmp MySomeFunc
sub esp, 0x20
…
ret
```

WickedWickedDll.dll

```
MySomeFunc:
<stuff>
…
mov edi, edi
push ebp
mov ebp, esp
jmp SomeFunc+5
```

That reminds me of trig class!

Inline Hooked Intra-Module Function Call



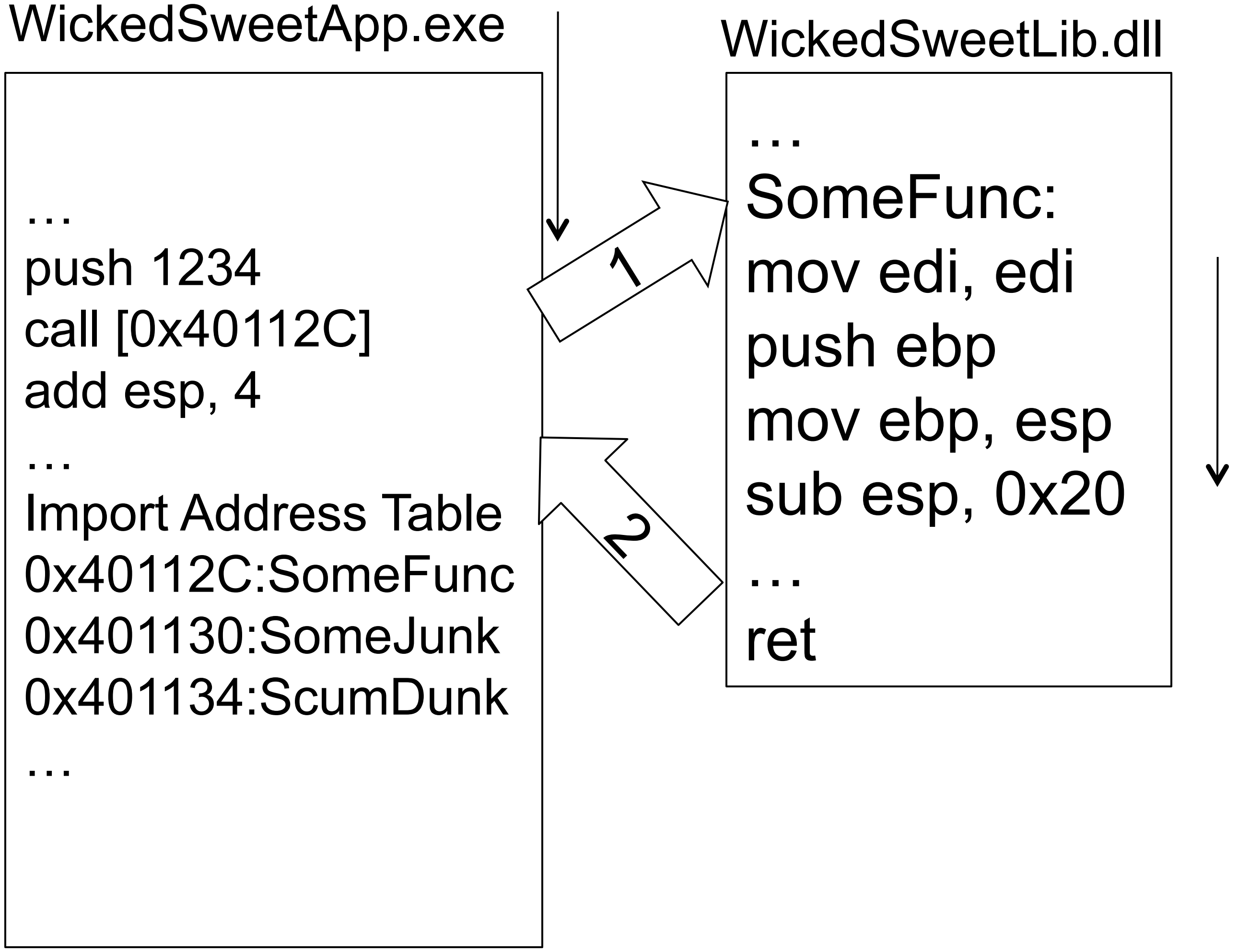
WickedSweetApp.exe

```
…
pu
cal          nc()
ad
…
…
SomeFunc:
jmp MySomeStuff
sul     0 20
…
ret
```

WickedWickedDll.dll

```
MySomeFunc:
<stu
…
mov
pusl
mov
jmp SomeFunc+5
```

Normal Inter-Module Function Call



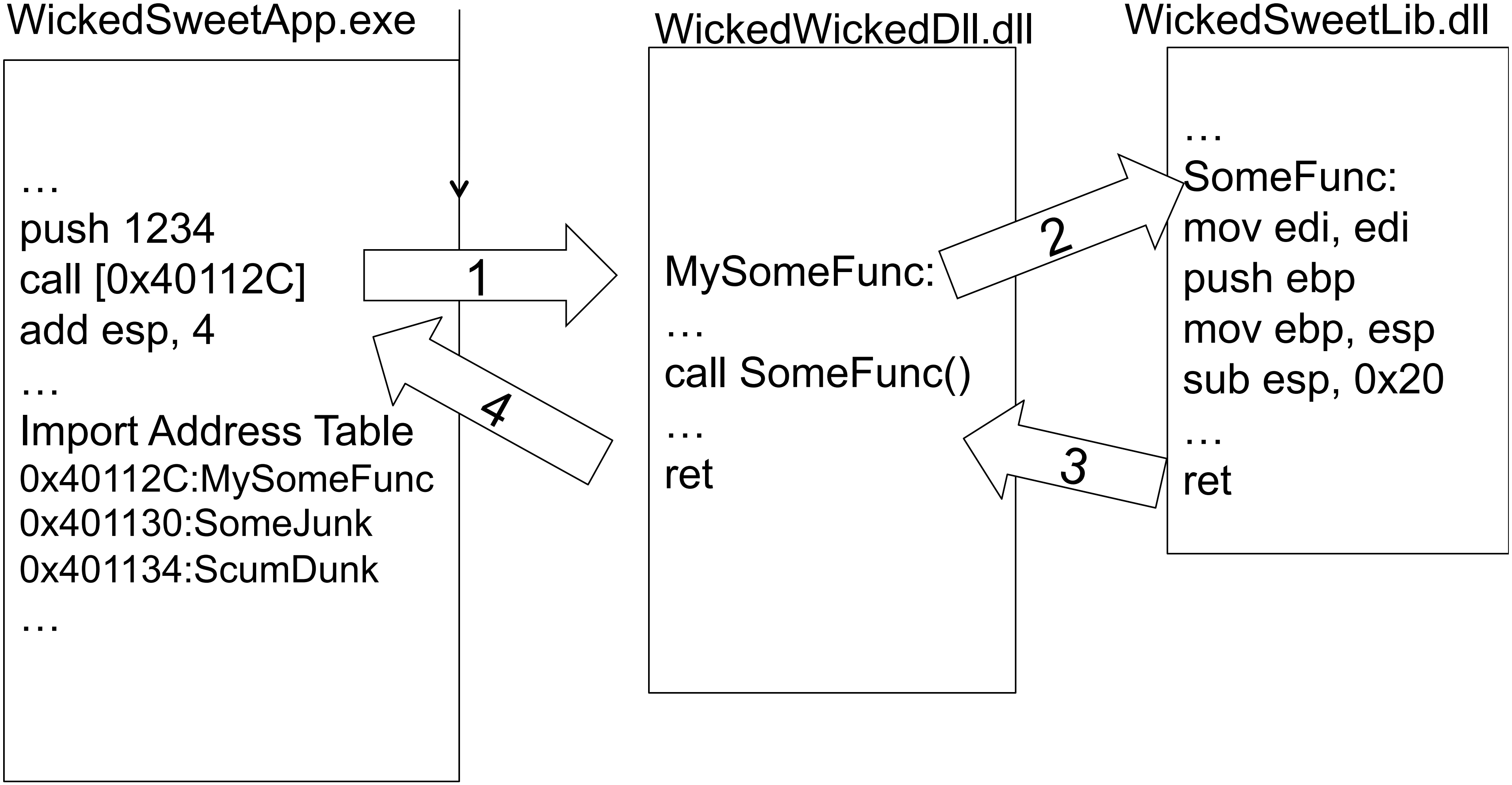
WickedSweetApp.exe

```
…
push 1234
call [0x40112C]
add esp, 4
…
Import Address Table
0x40112C:SomeFunc
0x401130:SomeJunk
0x401134:ScumDunk
…
```

WickedSweetLib.dll

```
…
SomeFunc:
mov edi, edi
push ebp
mov ebp, esp
sub esp, 0x20
…
ret
```

Normal Inter-Module Function Call



WickedSweetApp.exe

```
…
push 1234
call [0x40112C]
add esp, 4
…
Import Address Table
0x40112C:MySomeFunc
0x401130:SomeJunk
0x401134:ScumDunk
…
```

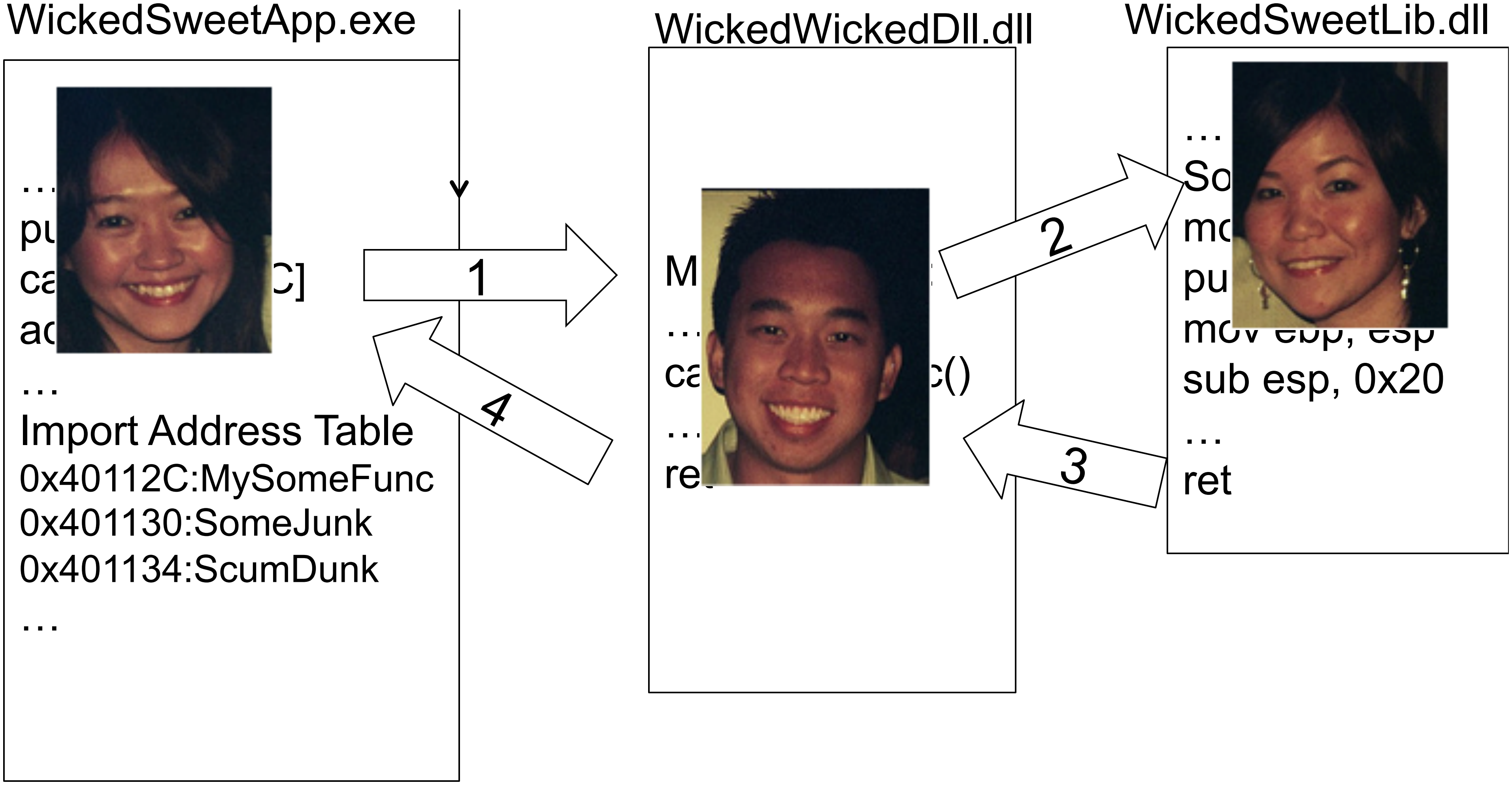
WickedWickedDll.dll

```
MySomeFunc:
…
call SomeFunc()
…
ret
```

WickedSweetLib.dll

```
…
SomeFunc:
mov edi, edi
push ebp
mov ebp, esp
sub esp, 0x20
…
ret
```

Normal Inter-Module Function Call



WickedSweetApp.exe

…
pu
ca C]
ac
…
Import Address Table
0x40112C:MySomeFunc
0x401130:SomeJunk
0x401134:ScumDunk
…

WickedWickedDll.dll

M :
…
ca c()
…
ret

WickedSweetLib.dll

…
So
mo
pu
mov ebp, esp
sub esp, 0x20
…
ret

Normal Interrupt Event

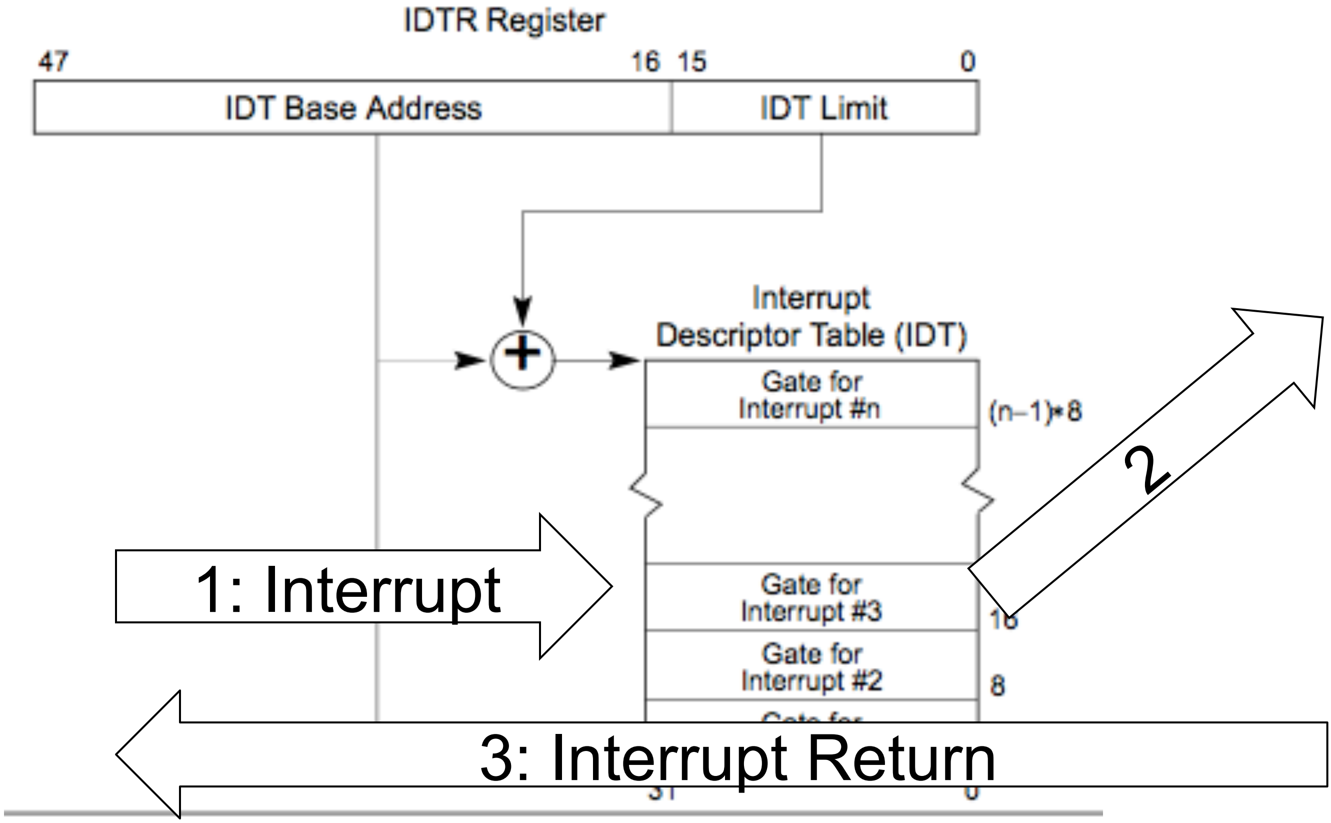


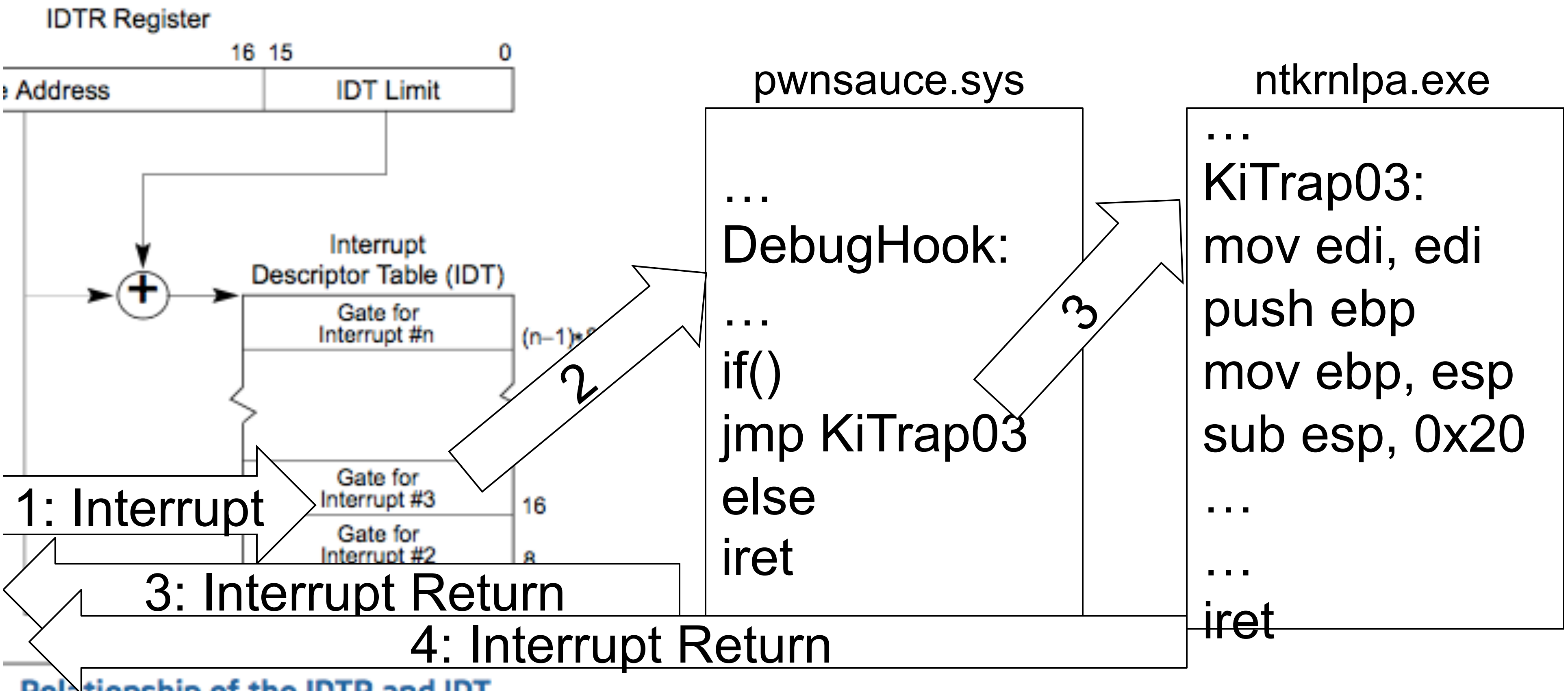
Figure 5-1. Rel

ntkrnlpa.exe

```
…
KiTrap03:
mov edi, edi
push ebp
mov ebp, esp
sub esp, 0x20
…
iret
```

Pop quiz, hot shot. What's the difference between ntoskrnl.exe and ntkrnlpa.exe?

Hooked Interrupt Event



pwnsauce.sys

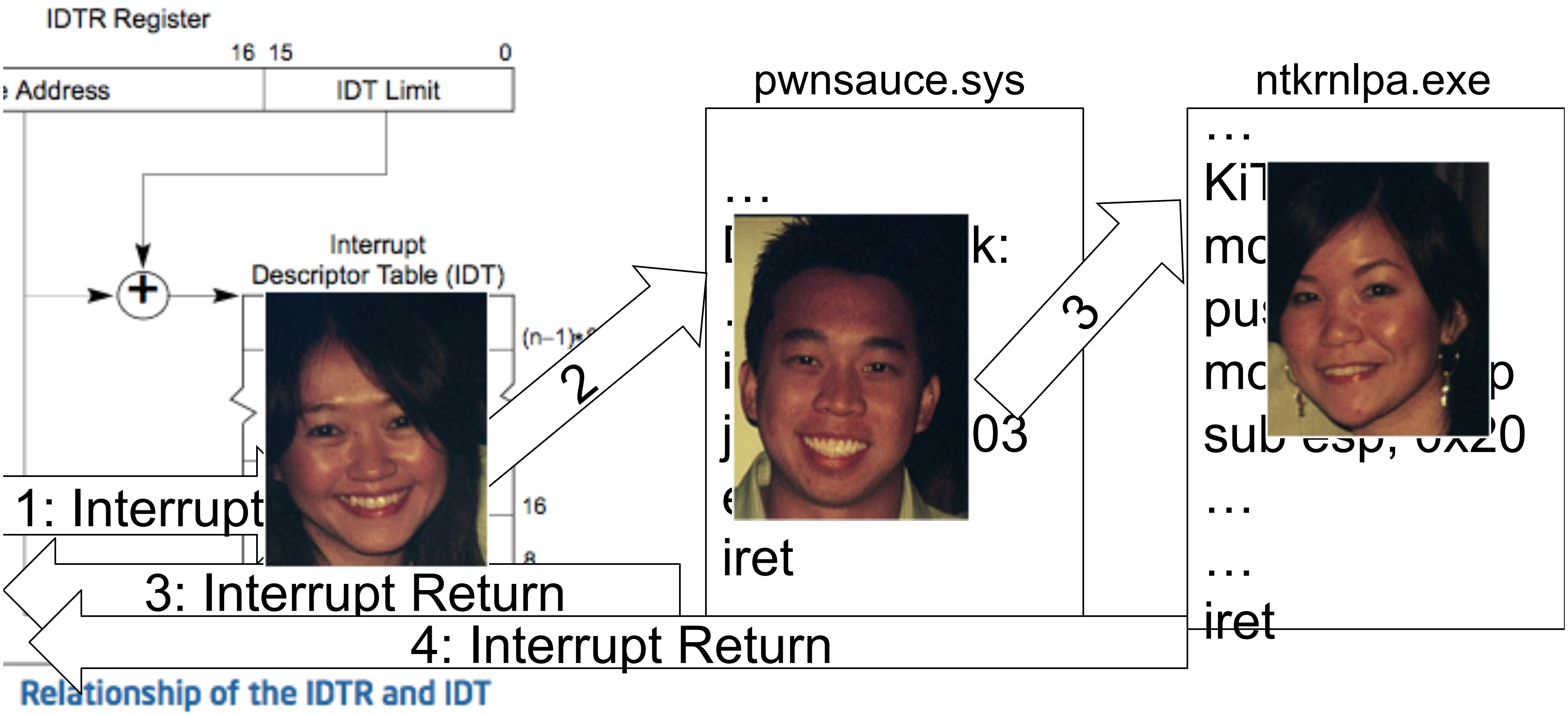
```
…
DebugHook:
…
if()
jmp KiTrap03
else
iret
```

ntkrnlpa.exe

```
…
KiTrap03:
mov edi, edi
push ebp
mov ebp, esp
sub esp, 0x20
…
…
iret
```

Relationship of the IDTR and IDT

Hooked Interrupt Event



1: Interrupt

3: Interrupt Return

4: Interrupt Return

pwnsauce.sys

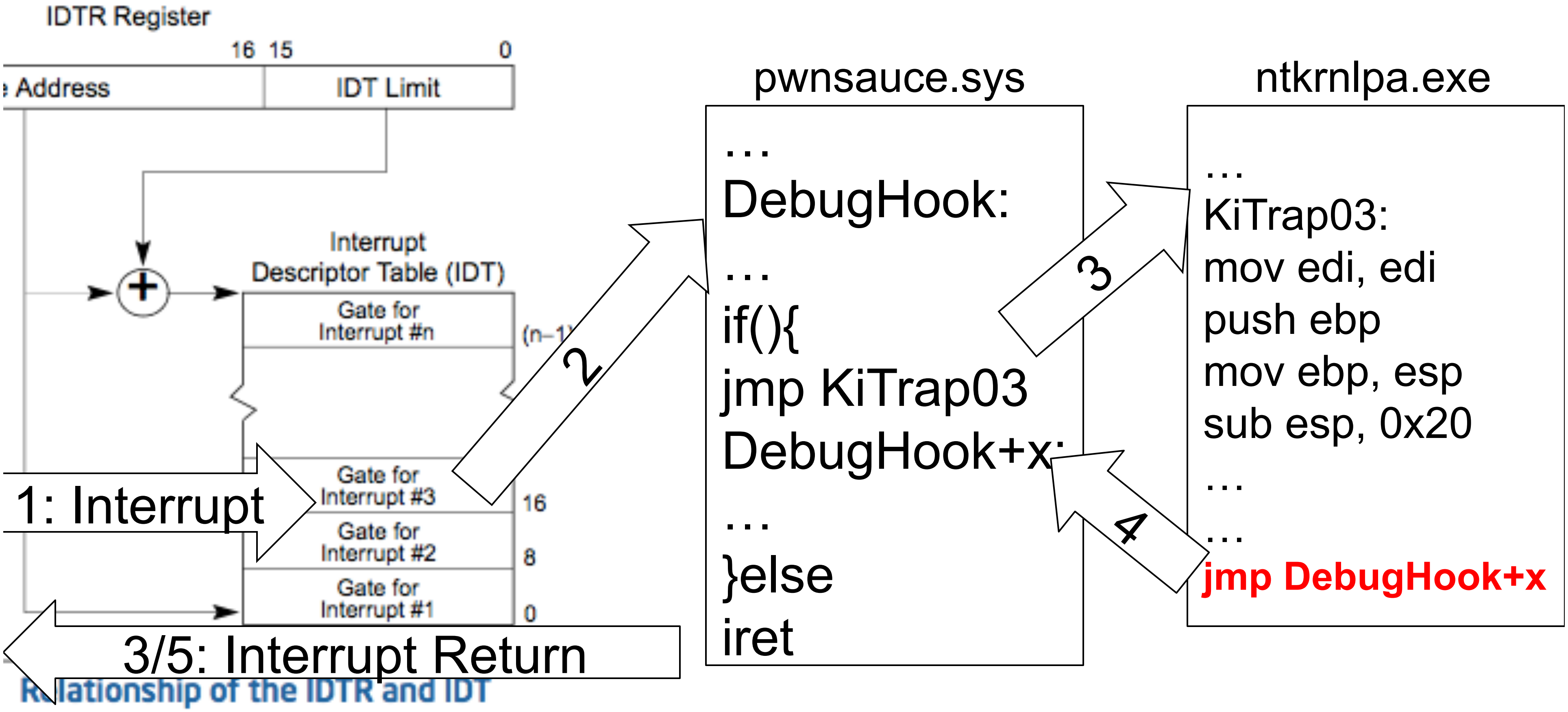
```
…
D          k:
.
i
j          03
e
iret
```

ntkrnlpa.exe

```
…
KiT
mo
pu
mo          p
sub esp, 0x20
…
…
iret
```

Hooked IDT + inline hook

(not common, just saying. be aware of potential to mix and match techniques)



pwnsauce.sys

```
…
DebugHook:
…
if(){
jmp KiTrap03
DebugHook+x:
…
}else
iret
```

ntkrnlpa.exe

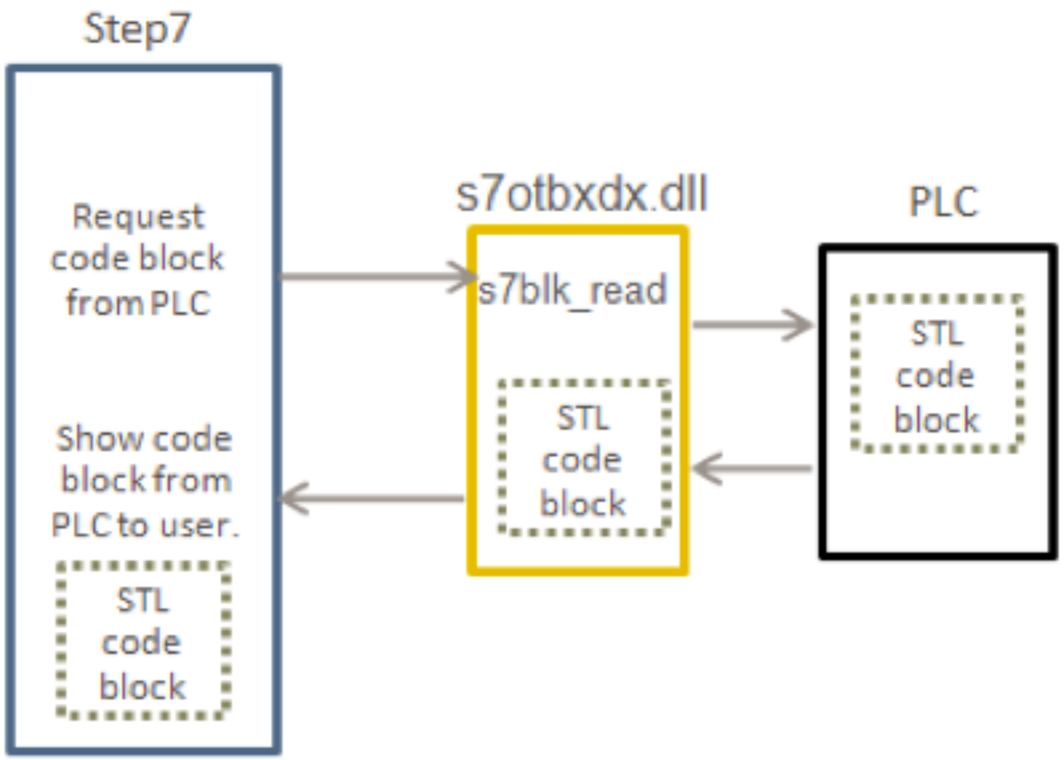
```
…
KiTrap03:
mov edi, edi
push ebp
mov ebp, esp
sub esp, 0x20
…
…
jmp DebugHook+x
```

Stuxnet trojaned DLL

- Stuxnet used forwarded exports for the 93 of 109 exports in s7otbxdx.dll which it didn’t need to intercept.

Figure 18

Step7 and PCL communicating via s7otbxdx.dll

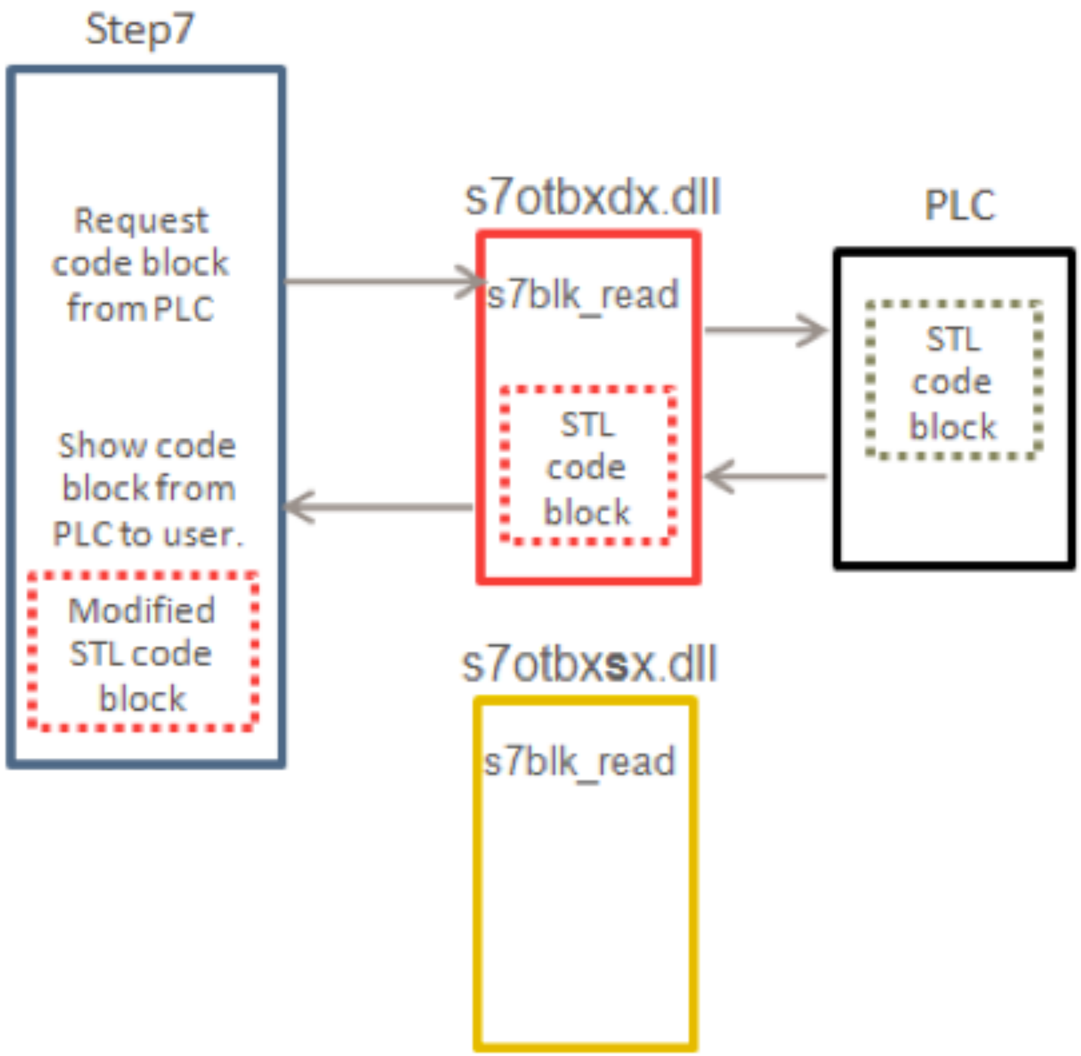


From http://www.symantec.com/content/en/us/enterprise/media/security_response/whitepapers/w32_stuxnet_dossier.pdf

Stuxnet trojaned DLL 2

Figure 19

Communication with malicious version of s7otbxdx.dll

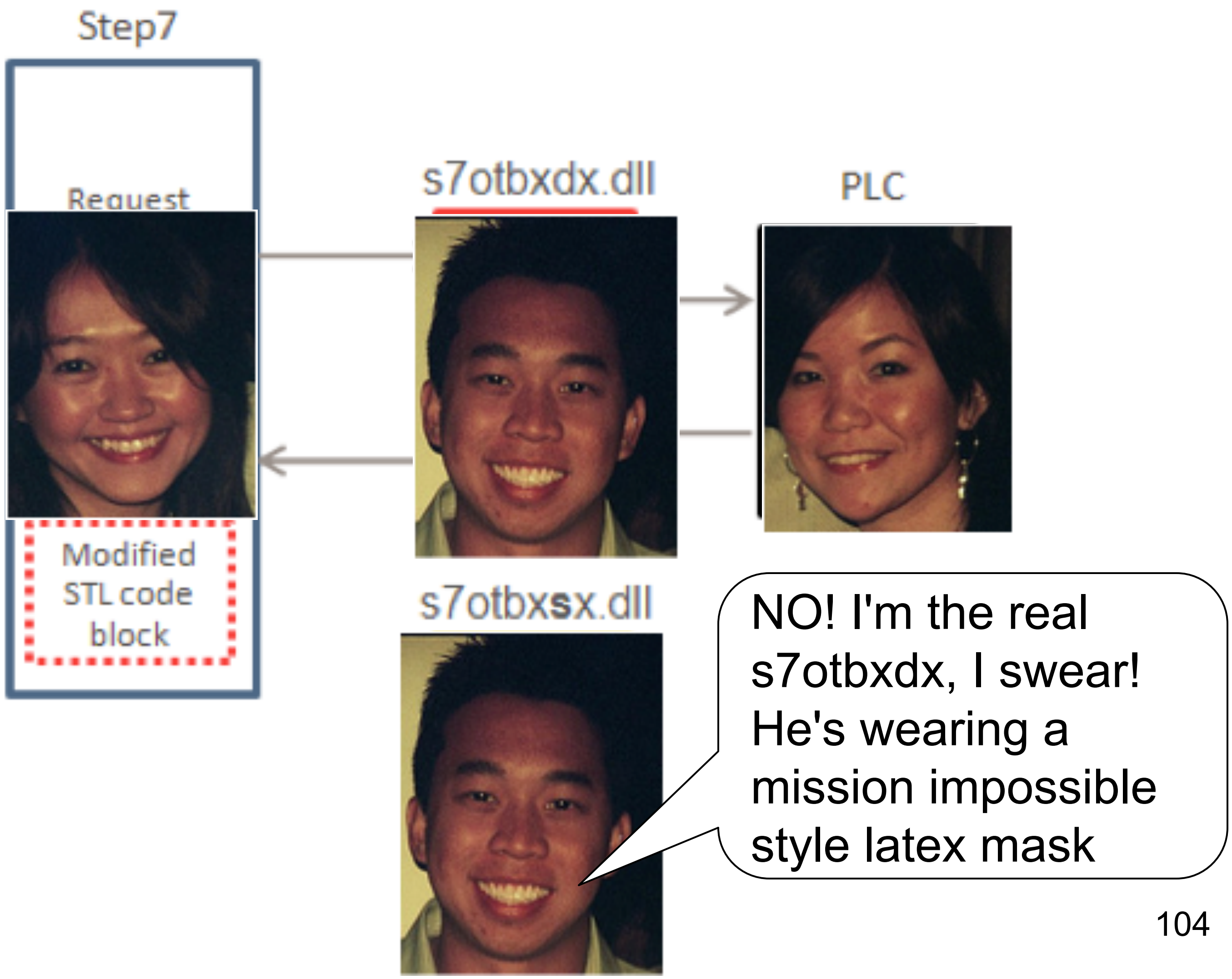


From http://www.symantec.com/content/en/us/enterprise/media/security_response/whitepapers/w32_stuxnet_dossier.pdf

Stuxnet trojaned DLL 2

Figure 19

Communication with malicious version of s7otbxdx.dll



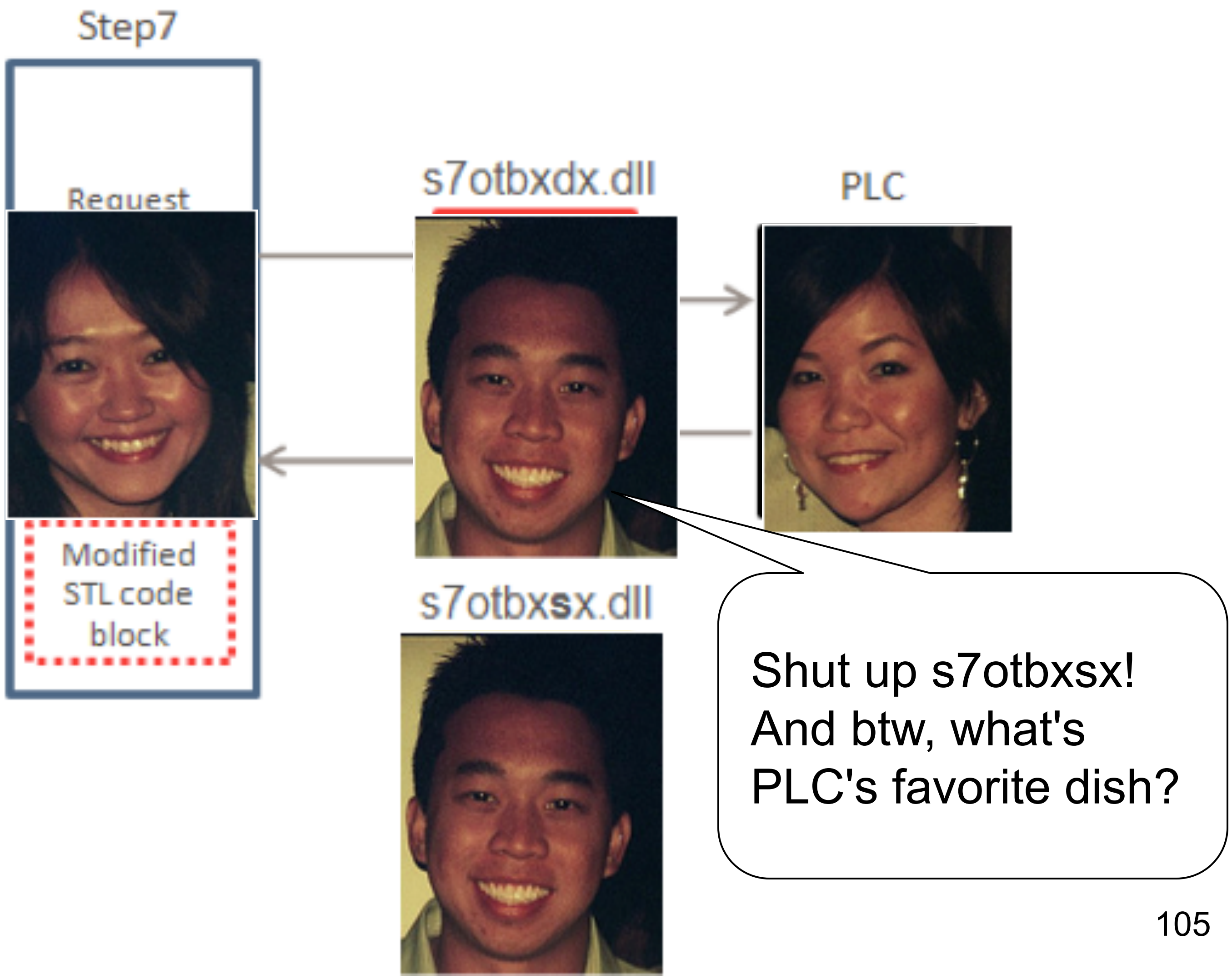
NO! I'm the real s7otbxdx, I swear! He's wearing a mission impossible style latex mask

From http://www.symantec.com/content/en/us/enterprise/media/security_response/whitepapers/w32_stuxnet_dossier.pdf

Stuxnet trojaned DLL 2

Figure 19

Communication with malicious version of s7otbxdx.dll



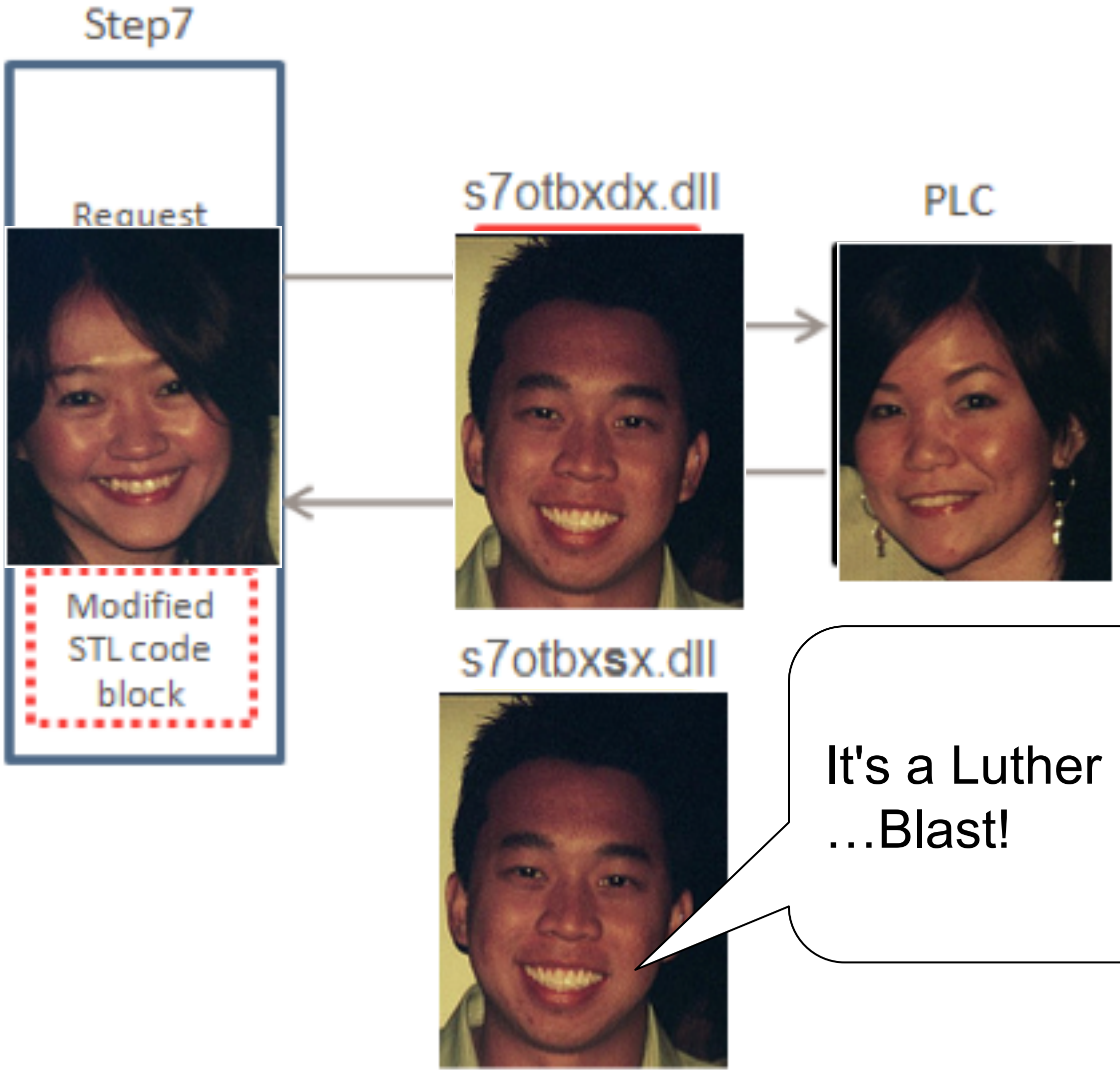
Shut up s7otbxsx! And btw, what's PLC's favorite dish?

From http://www.symantec.com/content/en/us/enterprise/media/security_response/whitepapers/w32_stuxnet_dossier.pdf

Stuxnet trojaned DLL 2

Figure 19

Communication with malicious version of s7otbxdx.dll



It's a Luther Burger.
…Blast!

From http://www.symantec.com/content/en/us/enterprise/media/security_response/whitepapers/w32_stuxnet_dossier.pdf

Further Reading

- Hacker Defender Readme: http://www.megasecurity.org/trojans/h/hackerdefender/Hackerdefender1.00r.html

References
(from the early "Rootkits are lame" talk slides)

- [VMWatcher] http://www.csc.ncsu.edu/faculty/jiang/pubs/CCS07.pdf
- [NICKLE]: http://friends.cs.purdue.edu/dokuwiki/doku.php?id=nickle
- [3] “TDL rootkit x64 goes wild” http://www.prevx.com/blog/154/TDL-rootkit-x-goes-in-the-wild.html
- [HyperSentry] http://discovery.csc.ncsu.edu/pubs/ccs10.pdf
- [HookMap] http://www4.ncsu.edu/~zwang15/files/raid08.pdf
- [HookSafe] http://www4.ncsu.edu/~zwang15/files/ccs09.pdf
- [HookScout] http://www.ecs.syr.edu/faculty/yin/pubs/hookscout-dimva10.pdf

References 2
(from the early "Rootkits are lame" talk slides)

- [8] “Don’ t Tell Joanna, The Virtualized Rootkit Is Dead” https://www.blackhat.com/presentations/bh-usa-07/Ptacek_Goldsmith_and_Lawson/Presentation/bh-usa-07-ptacek_goldsmith_and_lawson.pdf
- [9] “Compatibility is Not Transparency: VMM Detection Myths and Realities”

  http://www.usenix.org/event/hotos07/tech/full_papers/garfinkel/garfinkel_html/
- [DKOM] “VICE – Catch the hookers”- http://www.blackhat.com/presentations/bh-usa-04/bh-us-04-butler/bh-us-04-butler.pdf
- [KOH] “Kernel Object Hooking (KOH) Rootkits” - http://www.rootkit.com/newsread.php?newsid=501
- [DeepWatch] “Chipset Based Approach to Detect Virtualization Malware” http://www.blackhat.com/presentations/bh-usa-08/Bulygin/bulygin_Chip_Based_Approach_to_Detect_Rootkits.pdf

References 3
(from the early "Rootkits are lame" talk slides)

- [SWATT] SWATT: SOFTWARE-BASED ATTESTATION FOR EMBEDDED SYSTEMS, http://sparrow.ece.cmu.edu/~adrian/projects/swatt.pdf
- [SBAP] SBAP: SOFTWARE-BASED ATTESTATION FOR PERIPHERALS, http://sparrow.ece.cmu.edu/group/pub/li_mccune_perrig_SBAP_trust10.pdf
- [SMMshmoo] Ring -1 vs. Ring -2: Containerizing Malicious SMM Interrupt Handlers on AMD-V, http://www.shmoocon.org/2010/slides/containerizing.zip
- [GhostBuster] The Strider GhostBuster Project, http://research.microsoft.com/en-us/um/redmond/projects/strider/rootkit/
- [LKIM] Linux kernel integrity measurement using contextual inspection, portal.acm.org/citation.cfm?id=1314354.1314362
- [Petroni] An Architecture for Specification-Based Detection of Semantic Integrity Violations in Kernel Dynamic Data

  http://www.usenix.org/event/sec06/tech/full_papers/petroni/petroni_html/