Advanced x86:
BIOS and System Management Mode Internals
More Fun with SMM

Xeno Kovah && Corey Kallenberg

LegbaCore, LLC

LEGBACORE

WE DO DIGITAL VOODOO

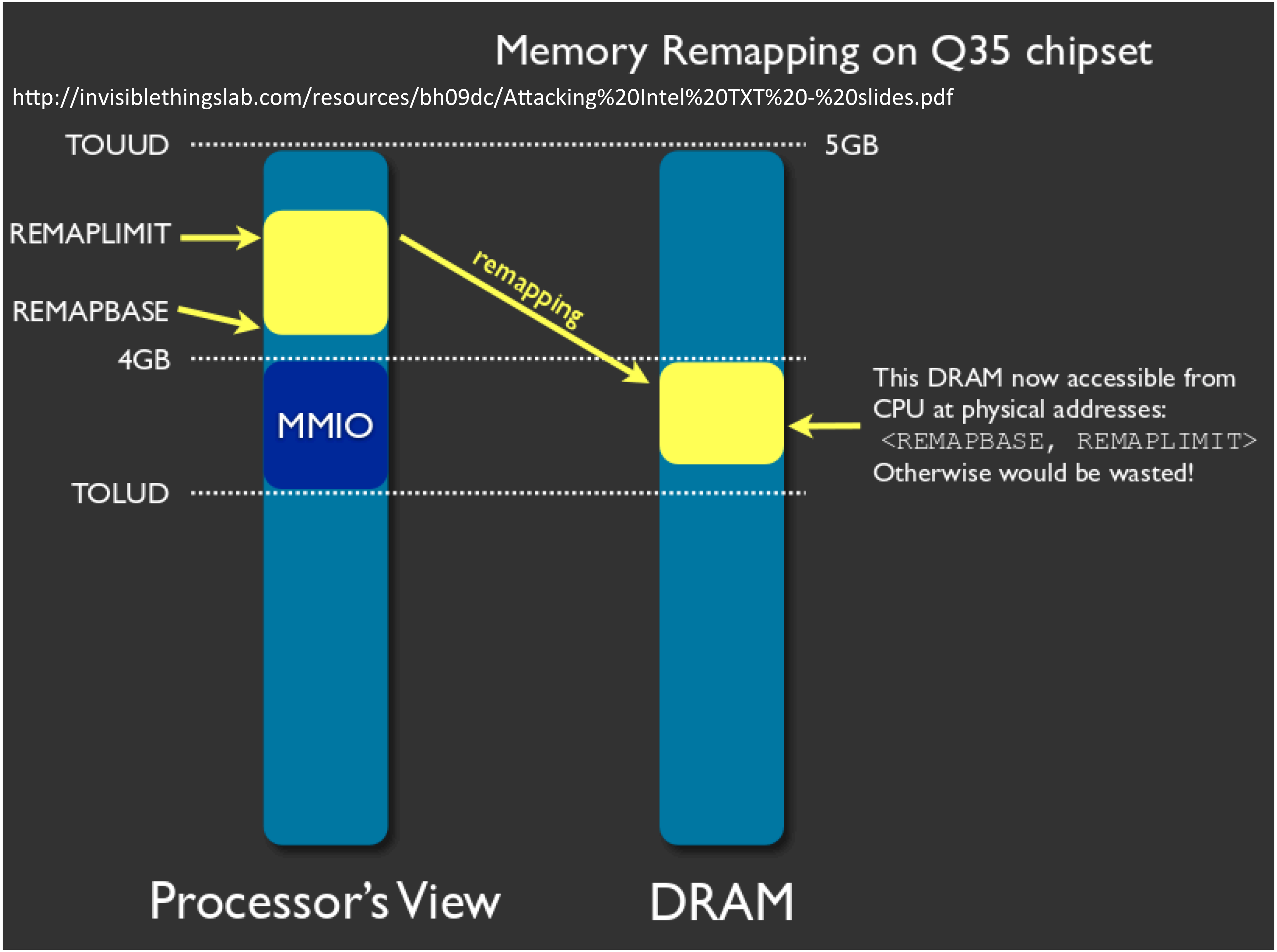
Other ways to break into SMM

Ways to break into SMM so far

- Break into the SPI flash chip, because it sets up the contents of SMRAM
- Get lucky and find out that the vendor didn’t set D_LCK
- Be on a system that’s too old to support SMRR (auto-win)
- Be on a system where the vendor didn’t set the SMRR
- Other?

Q35 chipset remapping bug

- There is a remapping feature present in chipsets that allows them to remap and reclaim space "lost" to the PCI Memory Mapped IO region of the memory map
- It turned out you could also use that bug to remap the protected SMRAM into non-protected space!
- Then it was a simple matter to read and write it


Memory Remapping on Q35 chipset
http://invisiblethingslab.com/resources/bh09dc/Attacking%20Intel%20TXT%20-%20slides.pdf
TOUUD
5GB
REMAPLIMIT
REMAPBASE
remapping
4GB
MMIO
TOLUD
This DRAM now accessible from CPU at physical addresses:
<REMAPBASE, REMAPLIMIT>
Otherwise would be wasted!
Processor's View
DRAM

http://invisiblethingslab.com/resources/bh09dc/Attacking%20Intel%20TXT%20-%20slides.pdf

Intel patched the bug in August 2008

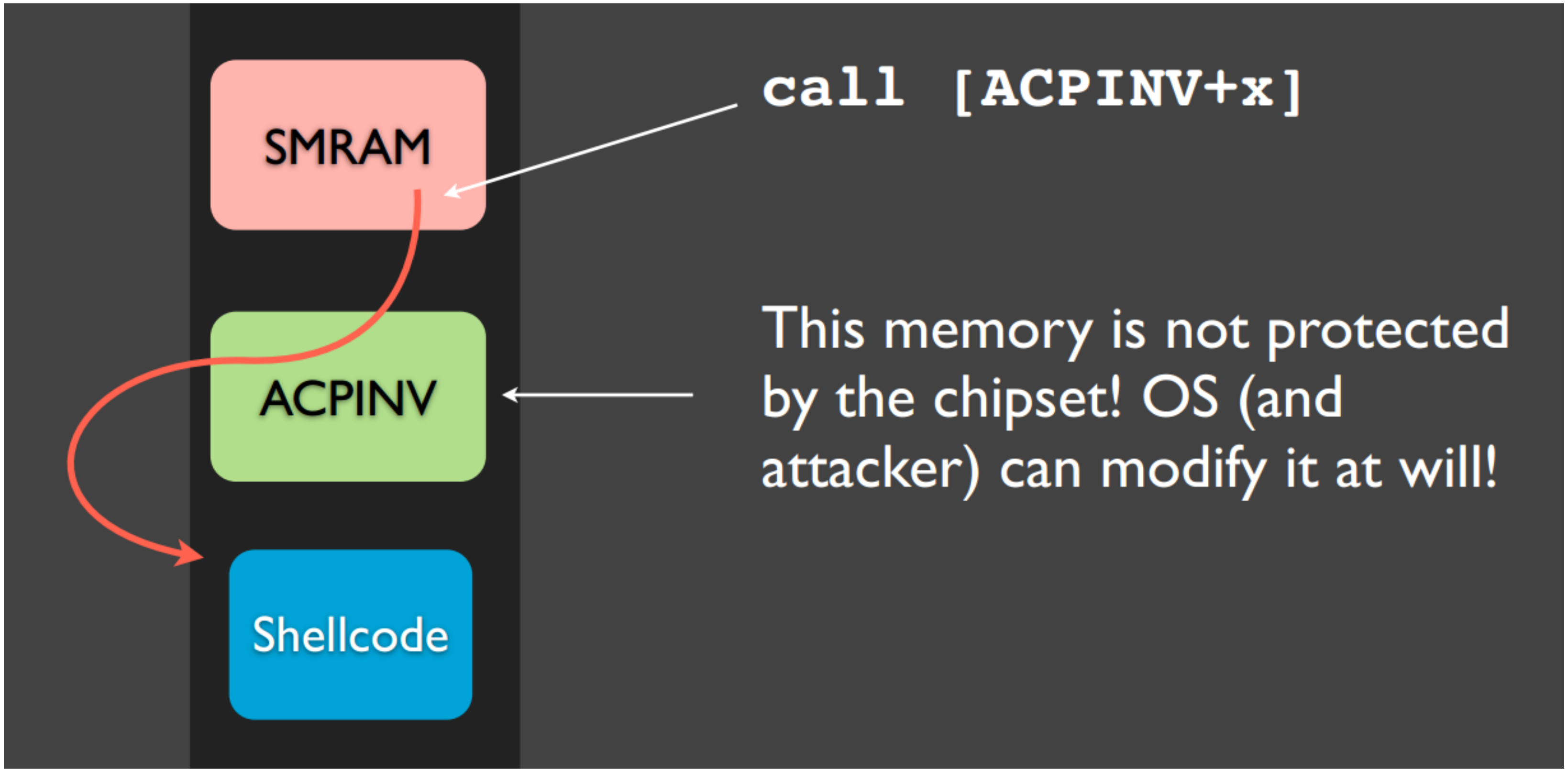
(This was done by patching the BIOS code to properly lock the memory configuration registers)

Xeno note:
This implies that other BIOSes could be vulnerable, if they're not setting the configuration correctly.
We never got around to re-investigating this, and therefore it's not a Copernicus built in check.
It could be that similar issues are lurking in deployed BIOSes.

What if…

- What if the SMI handler code was poorly written, and it basically reached out and grabbed resources outside of its protected SMRAM area?
- What if it executed code completely outside of its protected area?!

ITL Attack



- Untrusted ACPI function pointer called by SMM led to easily exploitable vulnerability

https://www.blackhat.com/presentations/bh-usa-09/WOJTCZUK/BHUSA09-Wojtczuk-AtkIntelBios-SLIDES.pdf

Code segment 0F000 is translated to physical RAM addresses F0000h - 100000h. This region contains system BIOS code such as POST and BIOS interrupts. This segment is not protected by SMM memory protections like SMI code. Any process with sufficient privileges to access physical memory can replace contents of this region with own code.

So, for instance, linear address 0F000:08070 in the above SMI handler is translated to physical address F8070h. During the boot this address gets loaded with BIOS code that reads registers in power management I/O space using ports 800h+offset:

```
 00008387: BA0008 mov dx,00800
 0000838A: 02D4 add dl,ah
 0000838C: 80D600 adc dh,000
 0000838F: C3 retn
 00008390: 52 push dx
 00008391: E8F3FF call 000008387
 00008394: EC in al,dx
 00008395: 5A pop dx
 00008396: C3 retn

; These instructions are loaded to 0F000:08070 address
; (F8070h in physical memory) by the BIOS from ROM chip
 00008397: E8F6FF call 000008390
 0000839A: CB retf
```

These BIOS instructions can be replaced with a jump to malicious code, so that this code will get executed by SMI handler with SMM privileges

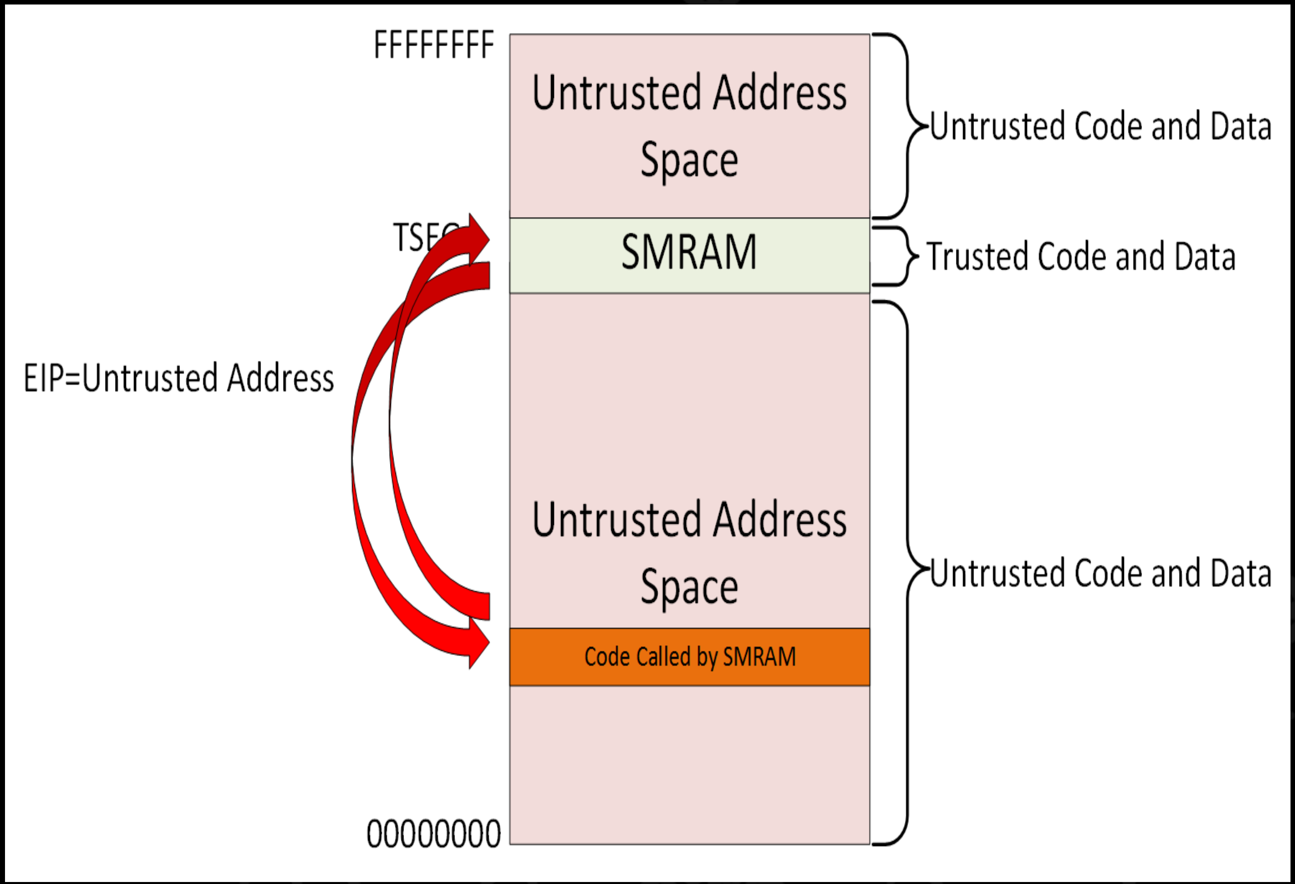
- “ASUS Eee PC and other series: BIOS SMM privilege escalation vulnerabilities” by core collapse
- Numerous instances of untrusted code execution by SMM in OEM firmware
- Probably lots more of these in proprietary SMM modules

http://archives.neohapsis.com/archives/bugtraq/2009-08/0059.html


NOT-SMM
THE INCURSION WALL IS HERE.
SMM

```
0000A1E0  81 31 4E 35 00 85 49 16 F9 51 77 95 1E 55 46 7E  .1N5.…I.ùQw•.UF~
0000A1F0  B7 73 53 94 8C D5 0F 3E 26 7D BF 5B 38 B2 D5 8D  ·sS”ŒÕ.>&}¿[8²Õ.
0000A200  00 01 00 00 00 00 00 00 00 FC 8F CB 25 28 4C 4C  .........ü.Ë%(LL
0000A210  44 42 52 43 00 00 00 00 18 39 06 DB 00 00 00 00  DBRC.....9.Û....
0000A220  70 AF 05 DB 00 00 00 00 00 00 00 00 00 00 00 00  p¯.Û............
0000A230  00 00 00 00 00 00 00 00 30 00 00 00 00 00 00 00  ........0.......
0000A240  04 05 00 00 00 00 00 00 FF FF FF FF 00 00 00 00  ........ÿÿÿÿ....
0000A250  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0000A260  00 00 00 00 00 00 00 00 34 00 00 00 00 00 00 00  ........4.......
0000A270  04 05 00 00 00 00 00 00 CC EB 05 DB 00 00 00 00  ........Ìë.Û....
0000A280  61 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  a...............
0000A290  49 ED 05 DB 00 00 00 00 01 96 23 D0 00 00 00 00  Ií.Û....–#Ð....
0000A2A0  00 00 00 00 00 00 00 00 60 89 05 DB 00 00 00 00  ........`‰.Û....
```

- We did a little RE work to determine which SMM code we could invoke from the OS by writing to port 0xB2
- In this case, function 0xDB05EDCC within SMM can be reached by writing 0x61 to port 0xB2
- Almost every UEFI system we surveyed used this format to record reachable SMM code



- We found a lot of these vulnerabilities
- They were so easy to find, we could write a ~300 line IDAPython script that found so many I stopped counting and (some) vendors stopped emailing me back

```
int smi_handler_9d37fe78()
{
  __int64 v0; // rax@1

  LODWORD(v0) = v9CEBED38(v9CEBECC8);
  v9CEBEE6C = v0;
  return v0;
}
```

```
int smi_handler_9d37fe78()
{
  __int64 v0; // rax@1

  LODWORD(v0) = v9CEBED38(v9CEBECC8);
  v9CEBEE6C = v0;
  return v0;
}
```

```
int smi_handler_9d37fc18()
{
  __int64 v0; // rax@1
  __int64 v1; // rcx@1
  char v3; // [sp+40h] [bp+18h]@1

  LODWORD(v0) = (*(int (__fastcall **)(char *))(v9CEBED58 + 24i64))(&v3);
  v9CEBEE74 = v0;
  if ( v0 >= 0 )
  {
    LOBYTE(v1) = v3;
    LODWORD(v0) = (*(int (__fastcall **)(__int64))(v9CEBED58 + 64i64))(v1);
    v9CEBEE74 = v0;
  }
  return v0;
```

```
int smi_handler_9d37fc18()
{
  __int64 v0; // rax@1
  __int64 v1; // rcx@1
  char v3; // [sp+40h] [bp+18h]@1

  LODWORD(v0) = (*(int (__fastcall **)(char *))(v9CEBED58 + 24i64))(&v3);
  v9CEBEE74 = v0;
  if ( v0 >= 0 )
  {
    LOBYTE(v1) = v3;
    LODWORD(v0) = (*(int (__fastcall **)(__int64))(v9CEBED58 + 64i64))(v1);
    v9CEBEE74 = v0;
  }
  return v0;
```

Seriously HP?

```
char __fastcall smi_handler_bbb8c660(__int64 a1, __int64 a2)
{
  char v2; // bl@1
  signed __int64 v3; // rcx@1
  unsigned __int8 v4; // dl@10
  __int64 v5; // r8@20
  char result; // al@21
  __int16 v7; // [sp+30h] [bp-28h]@20
  __int16 v8; // [sp+32h] [bp-26h]@20

  v2 = vEFF01040;
  vEFF01040 |= 0x30u;
  v3 = 3149860880i64;
  qword_BBB8DCF8 = 3149860880i64;
  if ( v1D2 == -5200 || v1D2 == -5549 )
  {
    LOBYTE(a2) = vF803A;
    vBB29C788(&qword_BBB8DCF0, a2);
    if ( vF803A )
    {
      vBB24A1C0();
      vBB2893C0();
      vBB27B380();
      if ( v1C5 )
        vBB2893C8(432i64);
```

ACPI remapping attack

- "Memory sinkhole attack" by Domas at BlackHat 2015
- Fixed in Sandy Bridge (2nd generation Core I series) & Atom 2013 processors
  - Vulnerable on older

TODO: Intel SMRAM overlap bugs

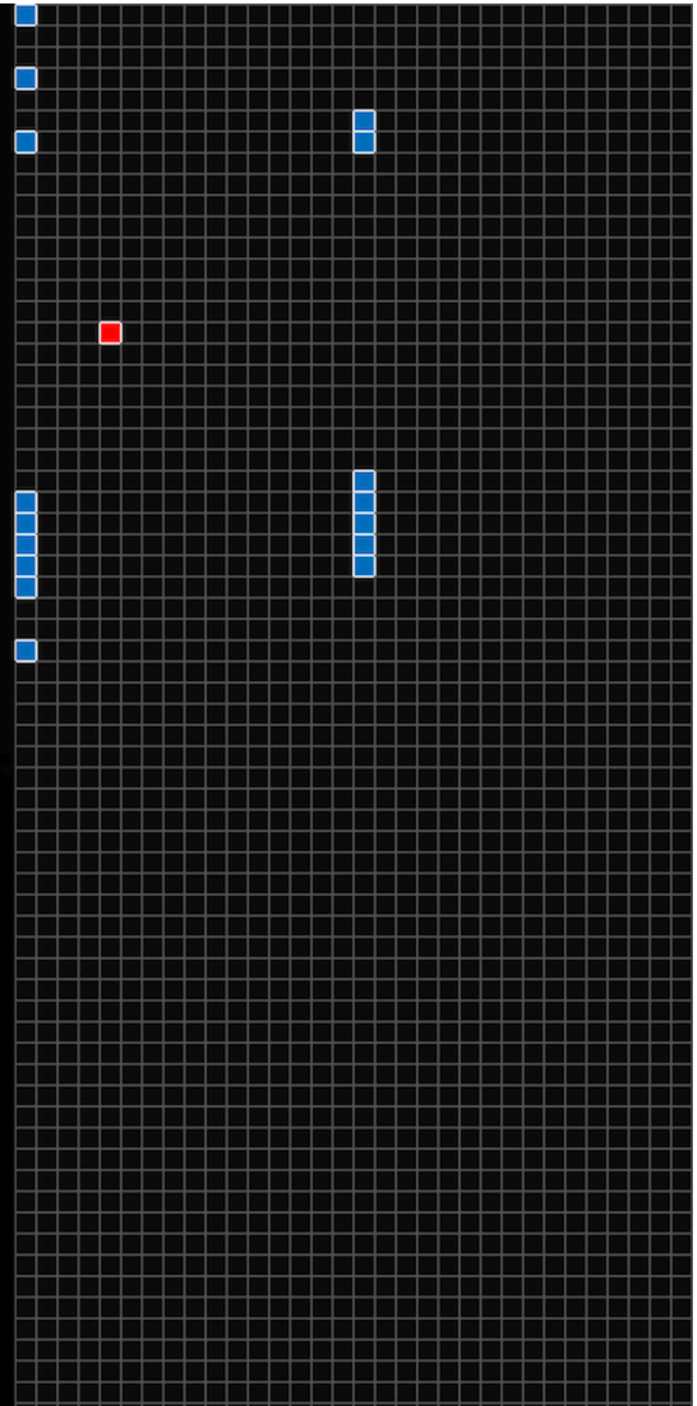
; from SMM
; smbase: 0x1ff80000
mov eax, [0x1ff80000]
; reads 0x00000000

The MCH never receives the memory request: the primary enforcer of SMM security is removed from the picture.

The APIC Remap Attack

- The Challenge:
  - Must be 4K aligned
    - Begin @ exactly SMI entry
  - 4096 bytes available
  - These are writeable
  - (And only a few bits each)
  - And this is an invalid instruction

The APIC Payload



“Unpatchable”?

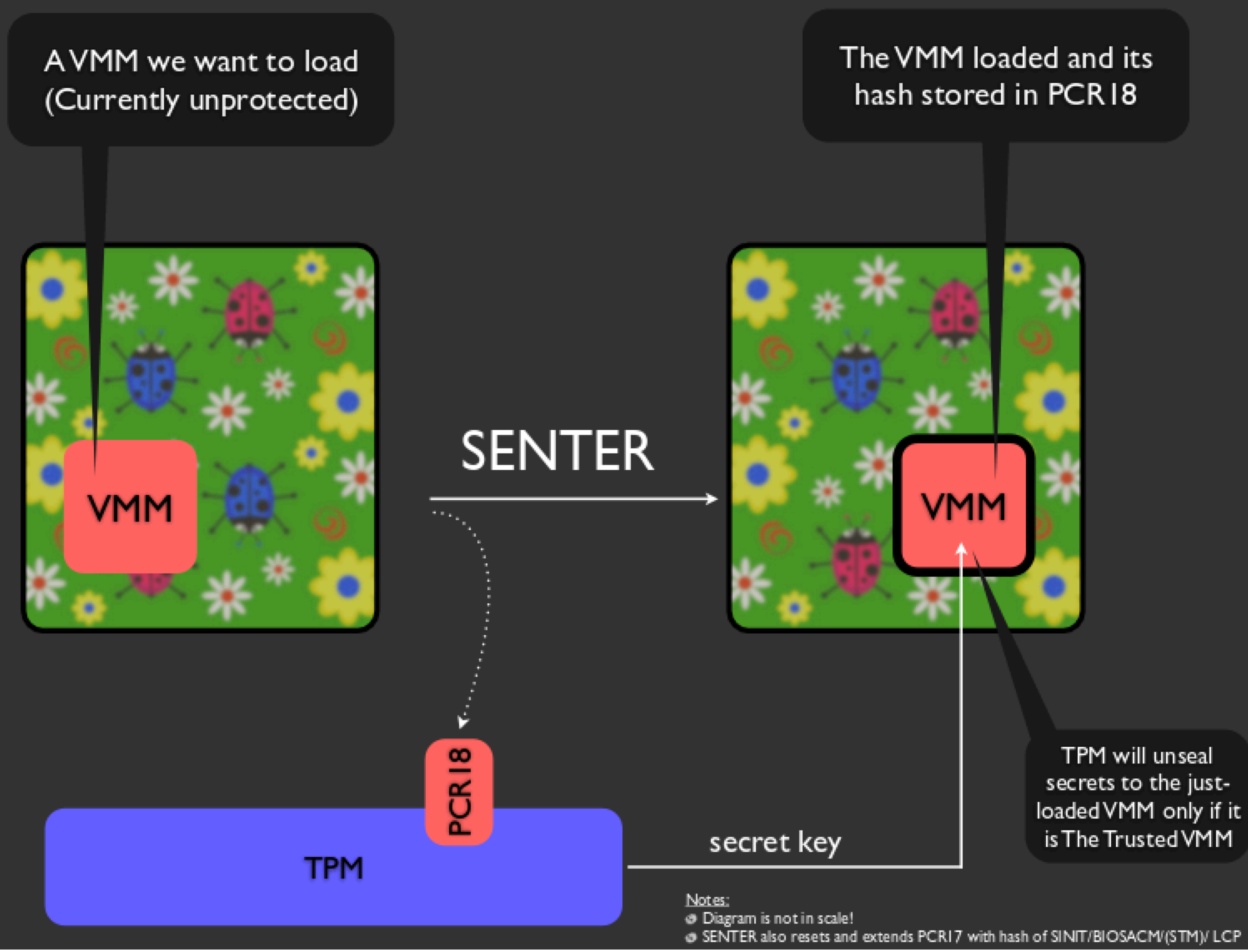
- Domas claimed that it’s unpatchable
- It can be patched by making the SMI handler entry point check if the APIC is mapped overlapping SMRAM, and then setting it back to the typical default address (either temporarily or permanently.)
- Yes, that breaks the “feature” of being able to relocate the APIC, but it’s highly unlikely anyone’s using it anyway, and if they were, technically they should be reading the current location anyway

Other things to do from SMM

Defeat Intel TXT

- Intel added new CPU instructions ("Safer Mode Extensions" in the manual, "Trusted Execution Technology" (TXT) for marketing) that try to make the system more secure
- The basic idea of TXT is to tear down your existing computing environment and build it back up from a secure starting point, so that you can trust whatever runs next

http://invisiblethingslab.com/resources/bh09dc/Attacking%20Intel%20TXT%20-%20slides.pdf

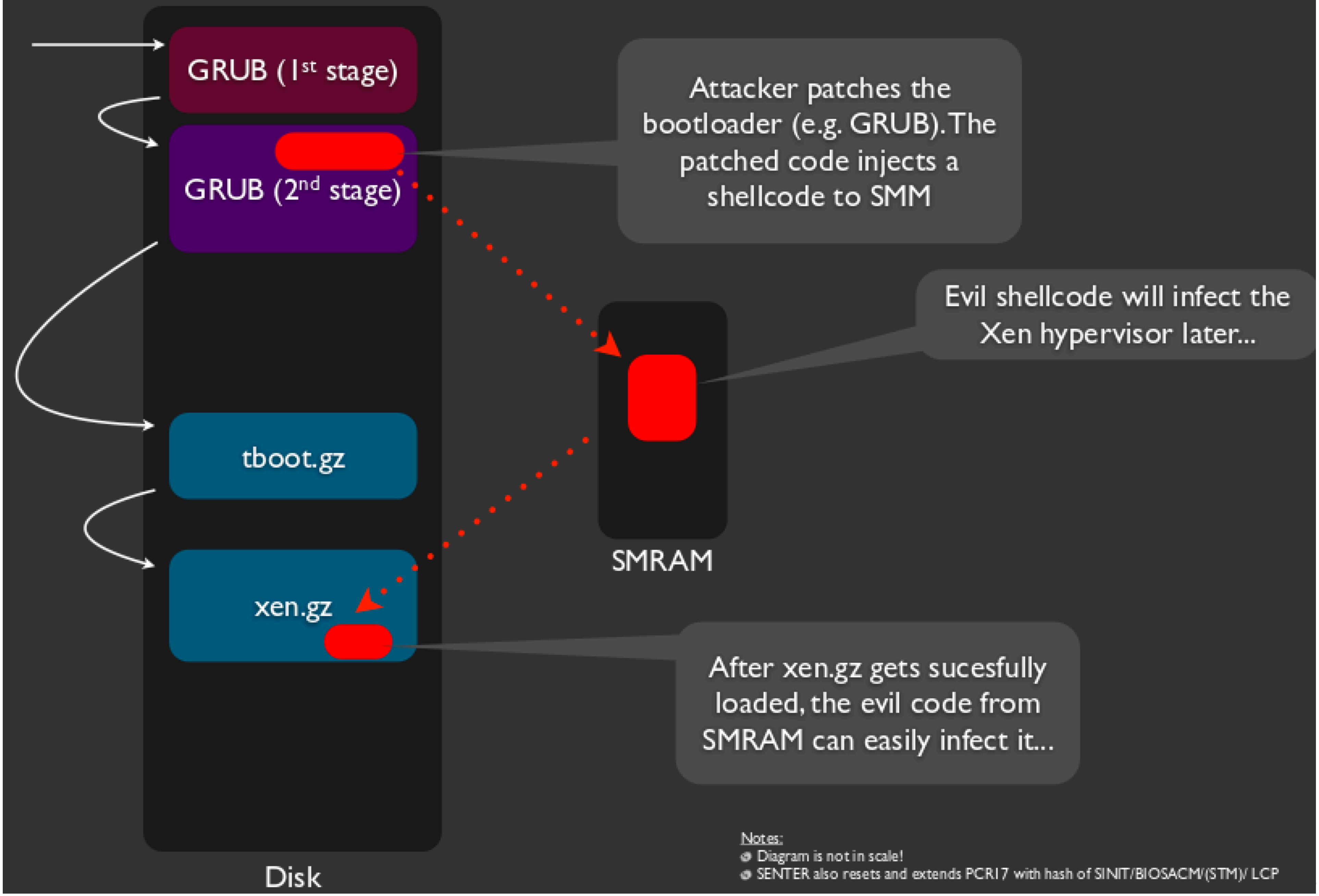


Defeat Intel TXT

- Unfortunately TXT does not measure SMRAM, and thus an attacker who has already broken into SMRAM can remain un-measured

TXT attack sketch (using tboot+Xen as example)

http://invisiblethingslab.com/resources/bh09dc/Attacking%20Intel%20TXT%20-%20slides.pdf



http://invisiblethingslab.com/resources/bh09dc/Attacking%20Intel%20TXT%20-%20slides.pdf

Intel

Invisible Things Lab

Solution to the TXT attack is called: STM

Can we take a look at this STM?

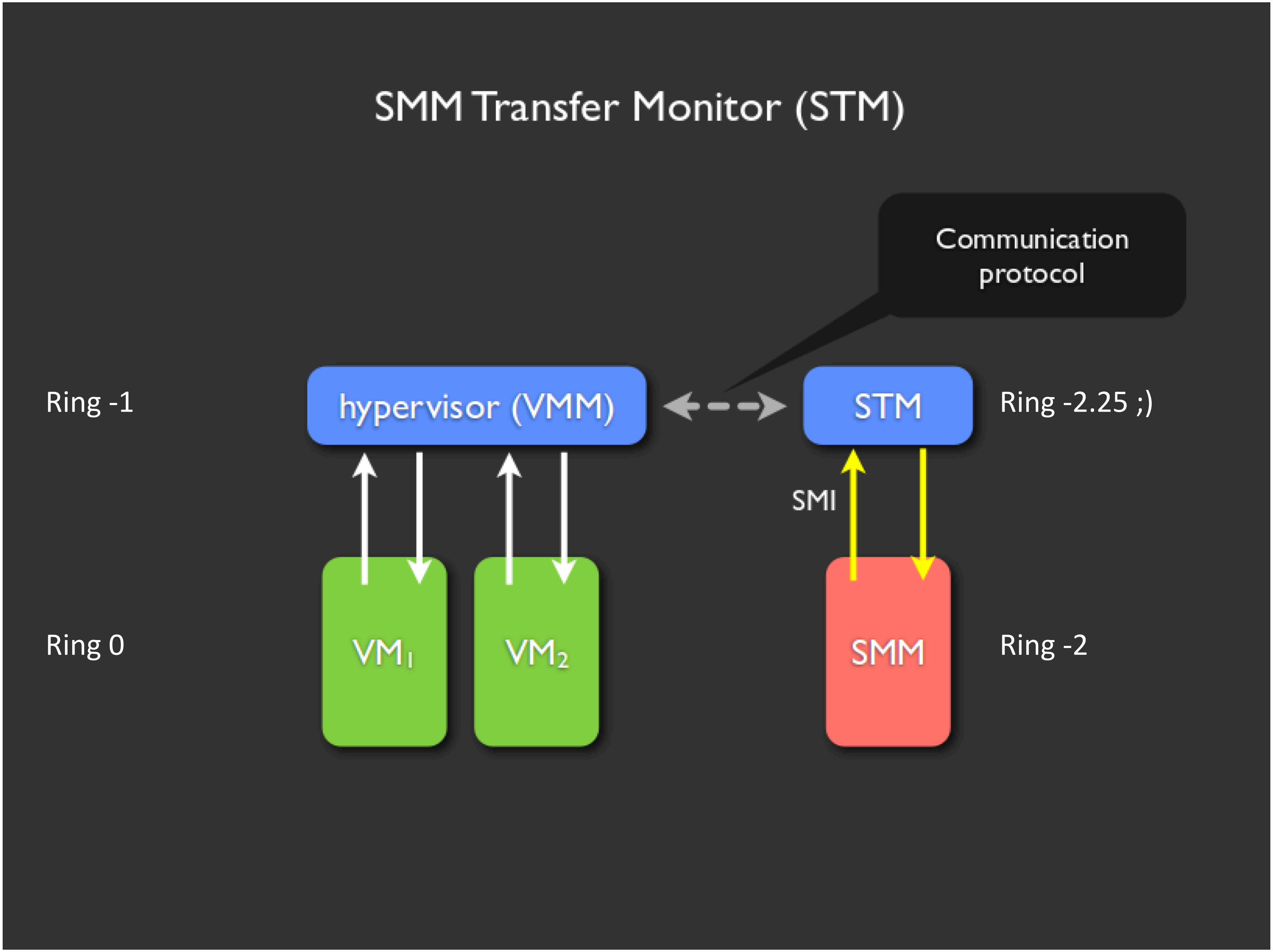
STM is currently not available.

?

It is simple to write. There was just no market demand yet.

?

The dialogs between ITL and Intel presented here have been modified for brevity and for better dramatic effect.


SMM Transfer Monitor (STM)
Communication protocol
Ring -1
hypervisor (VMM)
STM
Ring -2.25 ;)
SMI
Ring 0
VM1
VM2
SMM
Ring -2

So *IF* you had an STM

- Then you could place it in a portion of SMRAM called MSEG ("measured segment"), and when you do a TXT launch, you would get a measurement of whether your special SMM-jailing-hypervisor (STM) is intact or not.
- And then that STM would need to be the main entry point to SMM so that it could run before any potentially malicious code

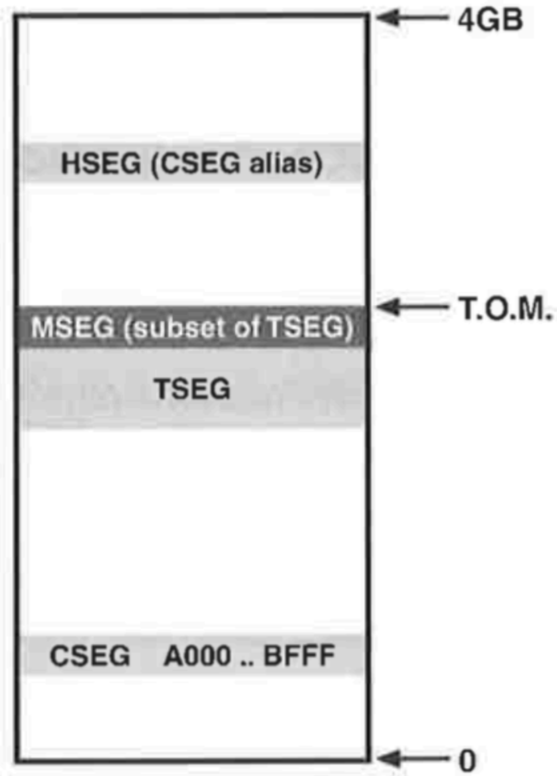


Figure 17.7 STM SMRAM

- Building an STM is one of LegbaCore's core business goals – because we're all *architecturally* vulnerable until that happens
  - But you have to tell your OEM that you want to *not* be vulnerable, otherwise they won't deploy it

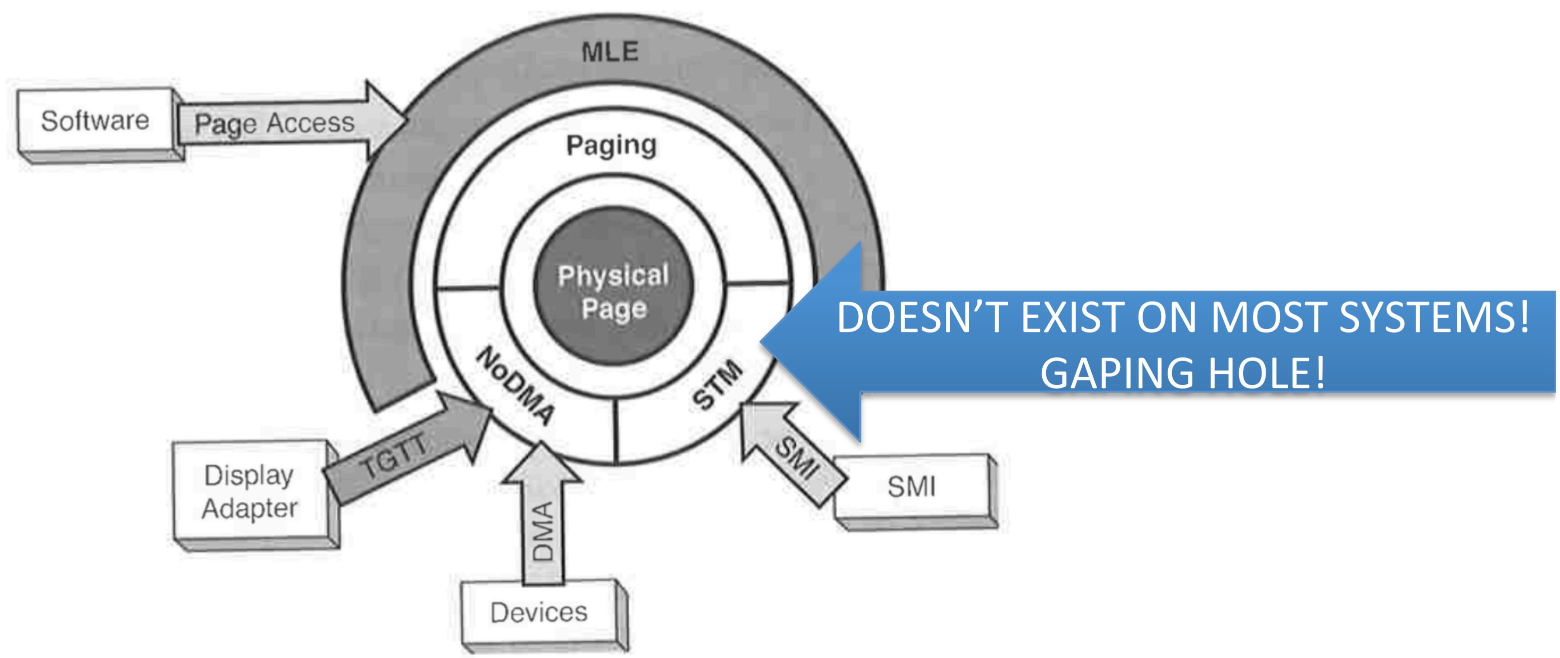


Figure 12.4 Physical Page Protections

From Intel Press "Dynamics of a Trusted Platform" - Grawrock

Late-breaking news!

- As of TODO, Intel finally released their reference STM and STM spec documentation!
- Still doesn't mean it's on anyone's machines… but it' a start!
- https://firmware.intel.com/content/smi-transfer-monitor-stm

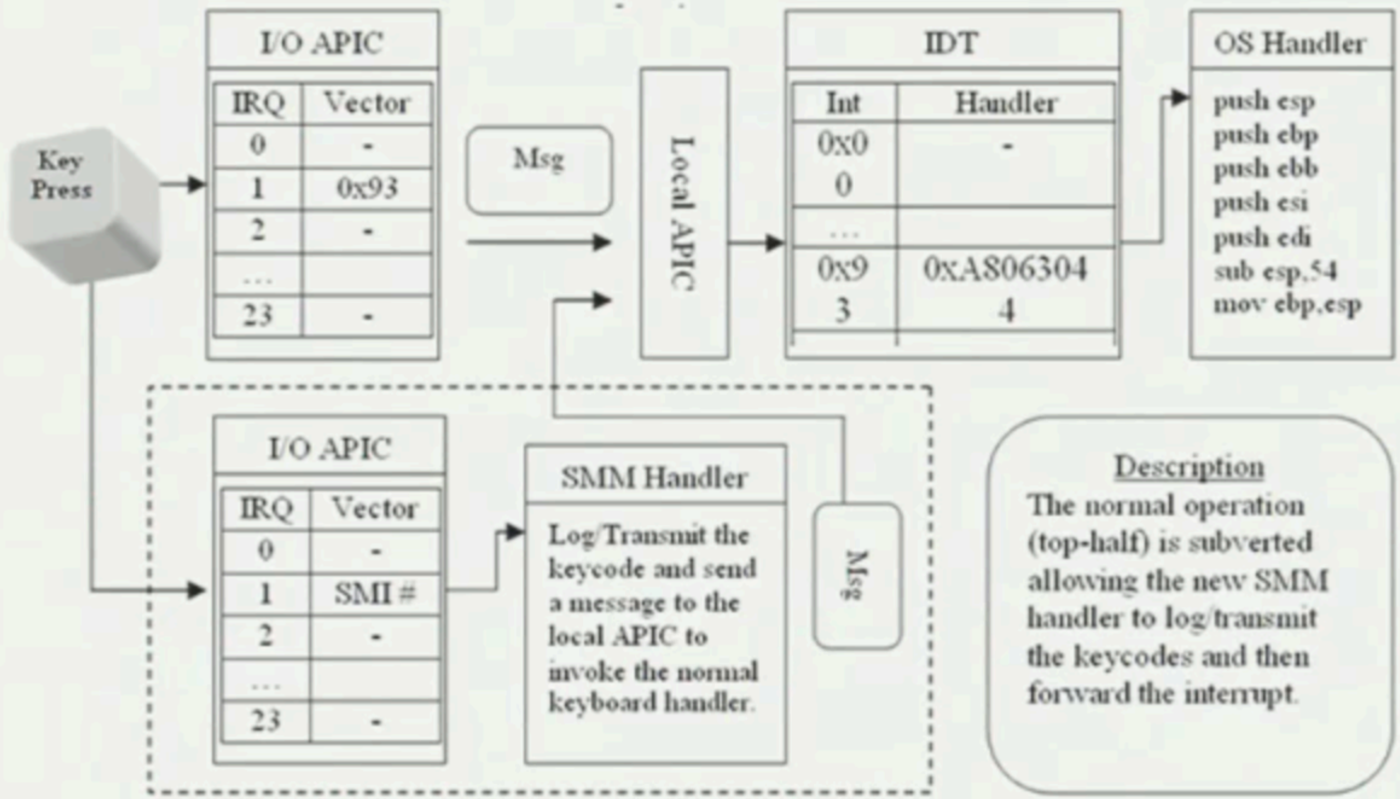
MitM Copernicus!

- And all the other flash tools
- We found a generic way for an SMM attacker to MitM flash reader tools' reading of the BIOS, so that the SMM attacker can hide his changes to the SPI flash chip
- *Moved into the SPI Programming slide deck*

What might an SMM backdoor look like?

Means to implement legacy PS2 keylogging without having to modify anything in the OS

Interrupt Redirection

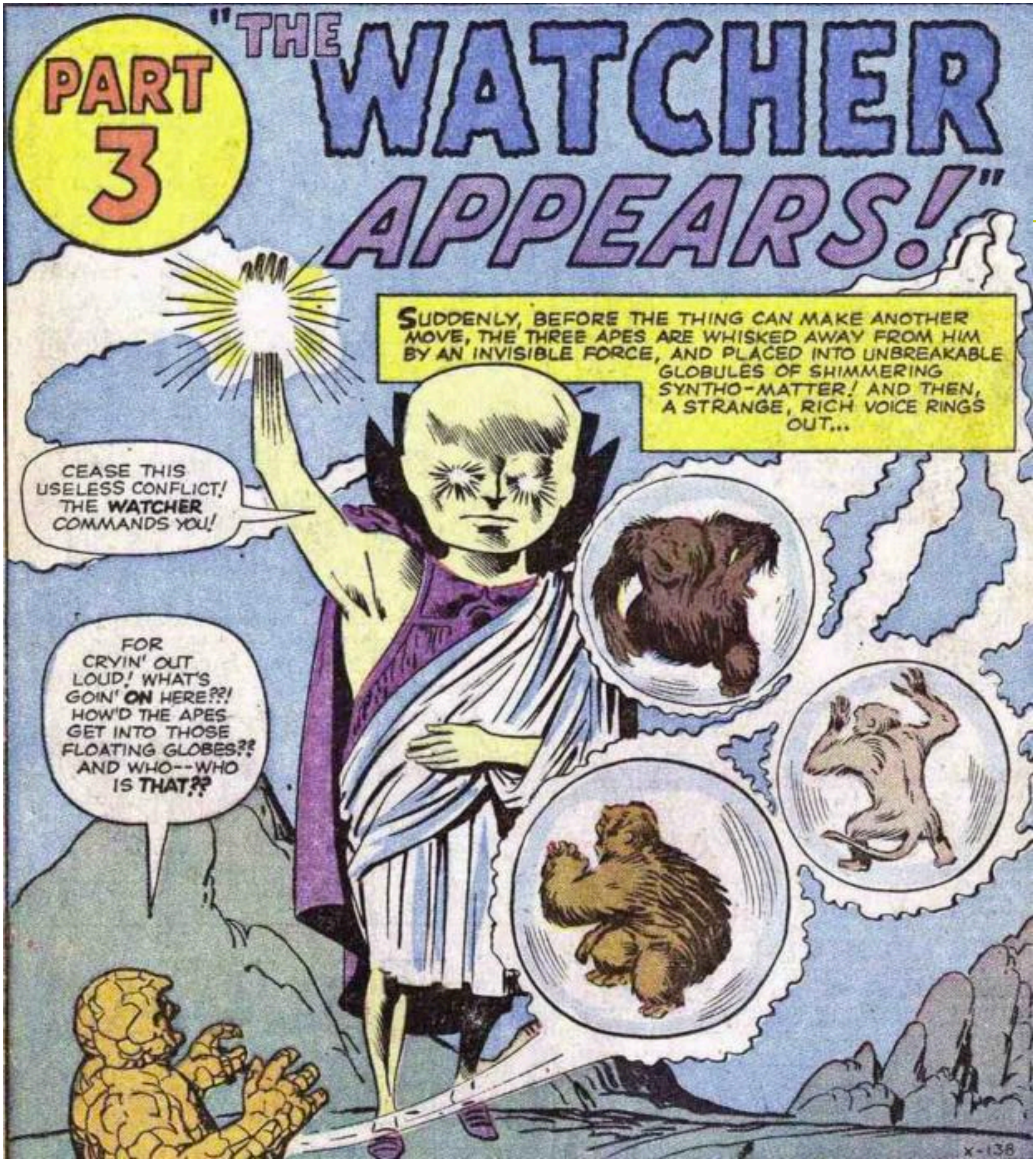


Network Backdoor

- Surprisingly easy… We just need to write to a few registers on the network card (also located in the PCI configuration space)
- Developed for Intel 8255X Chipset
  - Tested on Intel Pro 100B and Intel Pro 100S cards
  - Lots of other cards compatible with the 8255X chipset
  - Open documentation for Intel 8255X chipset
- See our “Deeper Door” talk / slides for details

The Watcher

- From https://www.blackhat.com/docs/us-14/materials/us-14-Kallenberg-Extreme-Privilege-Escalation-On-Windows8-UEFI-Systems.pdf



Marvel Comics
Fantastic Four #13, 1963

Presenting the first appearance of The Watcher!

The Watcher

- **The Watcher lives in SMM (where you can't look for him)**
- **It has no build-in capability except to scan memory for a magic signature**
- **If it finds the signature, it treats the data immediately after the signature as code to be executed**
- **In this way the Watcher performs *arbitrary code execution* on behalf of some controller, and is *completely OS independent***

- **A controller is responsible for placing into memory payloads for The Watcher to find**
- **These payloads can make their way into memory through any means**
  - Could be sent in a network packet which is never even processed by the OS
  - Could be embedded somewhere as non-rendering data in a document
  - Could be generated on the fly by some malicious javascript that's pushed out through an advertisement network
  - Could be pulled down by a low-privilege normal-looking dropper
  - Use your imagination

The Watcher, watching

Design tradeoffs:

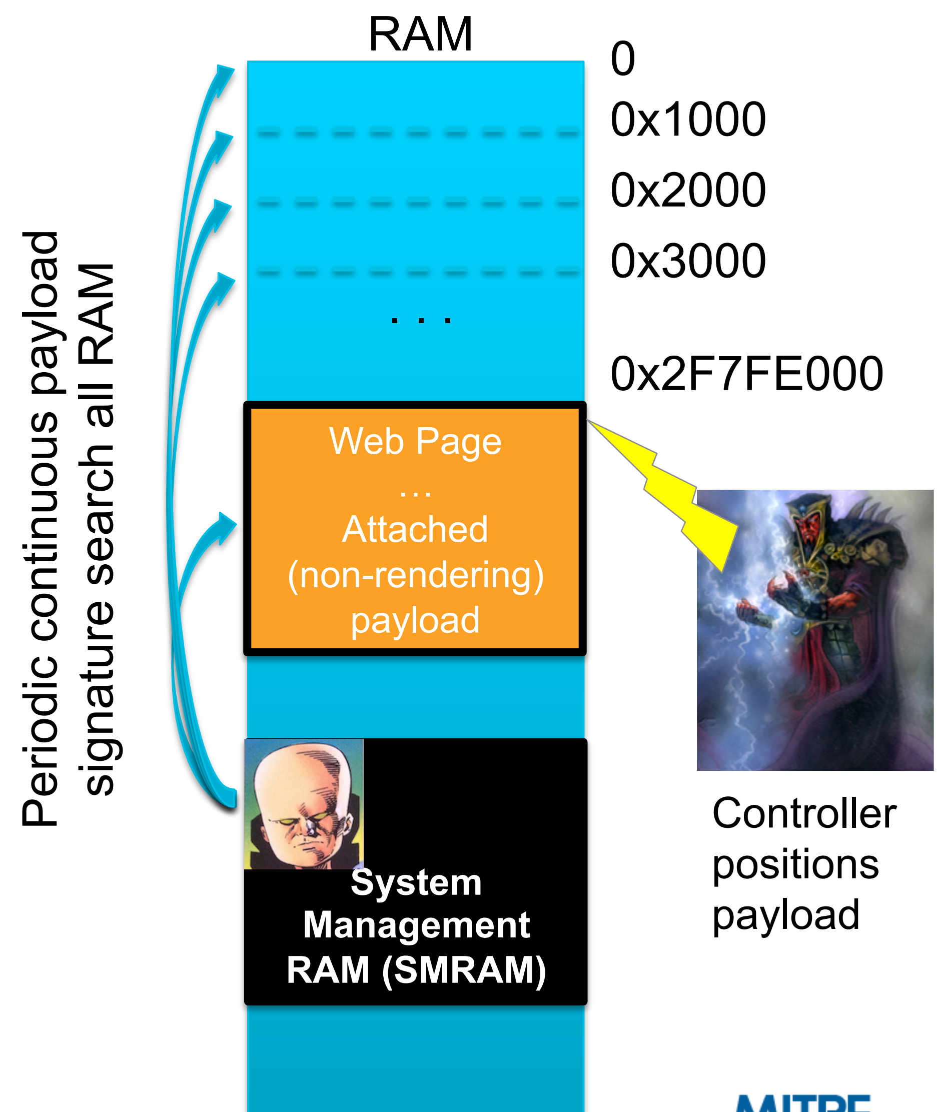
We don't want to scan every 4 byte chunk of memory. So instead we scan every 0x1000-aligned page boundary.

How do we guarantee a payload will be found on a page-aligned boundary?

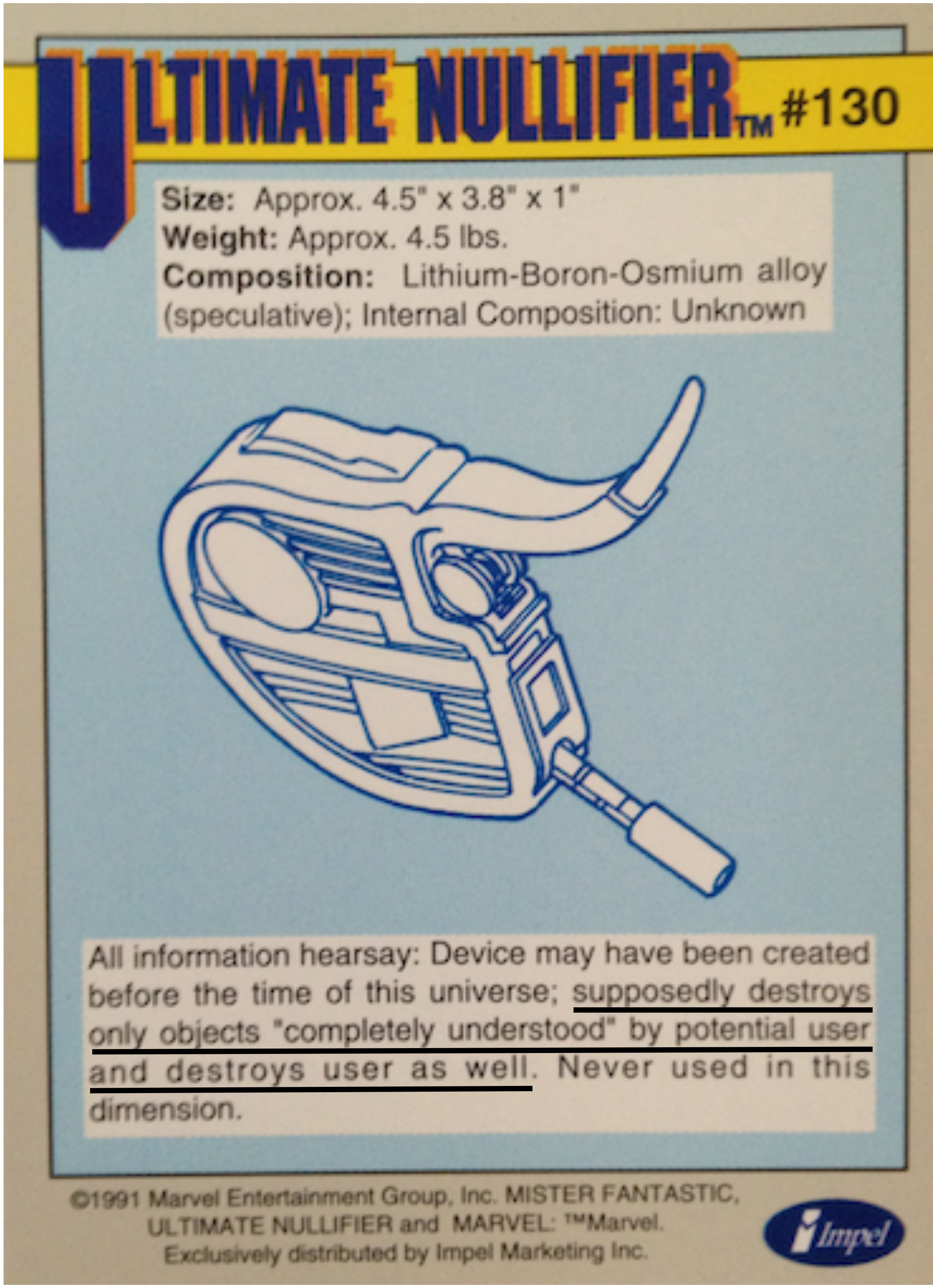
a) Another agent puts it there

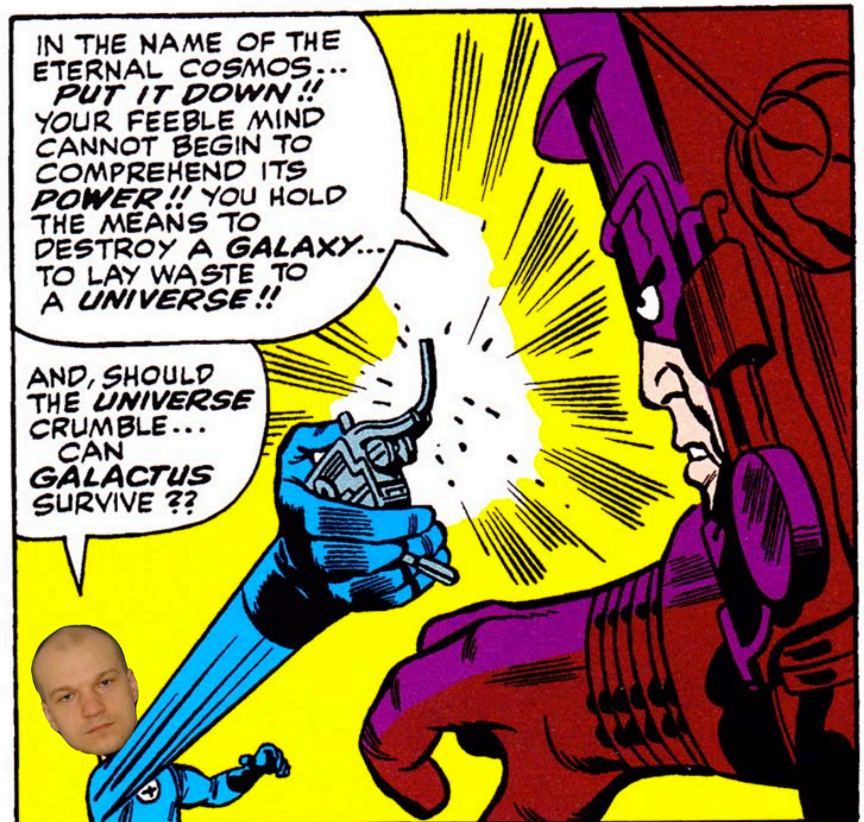
b) Controller prefixes the payload with a full 0x1000 worth of signatures and pointers to the code to be executed (this guarantees a signature will always be found at the boundary or boundary+4)

There are obviously many different ways it could be built.



Demo

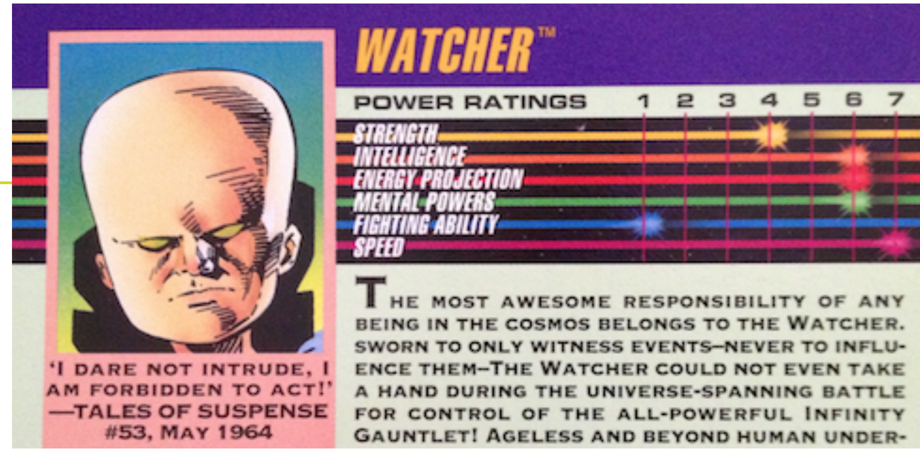




Marvel Comics
Fantastic Four #48, 1966

Impel 1991
Marvel Universe Series 2

Watcher Stats



Impel 1992
Marvel Universe Series 2

- **A week to get dev env set up (I didn't have my SPI programmer) and to find where to insert the code into SMM so it got called on every SMI**
- **2 days to write Watcher + basic print payload**
- **Watcher itself: ~ 60 lines of mixed C and inline assembly**
- **Print payload: 35 bytes + string, 12 instructions**
- **Ultimate Nullifier payload: 37 bytes, 11 instructions**

- **Overall point: very simple, very small, very powerful**
- **How likely do you think it is that there aren't already Watchers watching?**
- **But we can't know until people start integrity checking their BIOSes**

MITRE

LightEater

Hello my friends.
Welcome to my home
in the Deep Dark

From
http://legbacore.com/Research_files/HowManyMillionBIOSWouldYouLikeToInfect_Full2.pdf

Is it safe to use Tails on a compromised system?

Tails runs independently from the operating system installed on the computer. So, if the computer has only been compromised by software, running from inside your regular operating system (virus, trojan, etc.), then it is safe to use Tails. This is true as long as Tails itself has been installed using a trusted system.

If the computer has been compromised by someone having physical access to it and who installed untrusted pieces of hardware, then it might not be safe to use Tails.

- Time to rethink this…

LightEater on HP

- For a change of pace, let's see how easy evil-maid / border-guard / interdiction attacks are!
- NIC-agnostic exfiltration of data via Intel Serial-Over-LAN
- Option to "encrypt" data with bitwise rot13 to stop network defenders from creating a "Papa Legba" snort signature :P

LightEater on ASUS

- Uses hook-and-hop from DXE IPL to SMM
- From SMM attacks Windows 10
- Gets woken up every time a process starts, prints information about the process