Introducing Ring -3 Rootkits

Alexander Tereshkin and Rafal Wojtczuk

Black Hat USA, July 29 2009
Las Vegas, NV

https://t.me/learningnets

1 Introducing **Ring -3**

2 **Getting** there

3 Writing useful **Ring -3 rootkits**

https://t.me/learningnets

A Quest to Ring -3

https://t.me/learningnets

Ring 3 **Usermode** rootkits

Ring 0 **Kernelmode** rootkits

Ring -1 **Hypervisor** rootkits (Bluepill)

Ring -2 **SMM** rootkits

https://t.me/learningnets

Ring -3?

https://t.me/learningnets

What is this?

https://t.me/learningnets

MTG1.5

C67

C66

L3 R1426

C1089

C111 C113 C112 C116 C115 C114

https://t.me/learningnets

Yes, it is a chipset (MCH)

(More precisely Intel Q35 on this picture)

https://t.me/learningnets

Did you know it's also a standalone web server?

https://t.me/learningnets

Many (all?) vPro chipsets (MCHs) have:

- ✓ An Independent CPU (not IA32!)
- ✓ Access to dedicated DRAM memory
- ✓ Special interface to the Network Card (NIC)
- ✓ Execution environment called Management Engine (ME)

https://t.me/learningnets

Your chipset is a little computer. It can execute programs **in parallel and independently** from the main CPU!

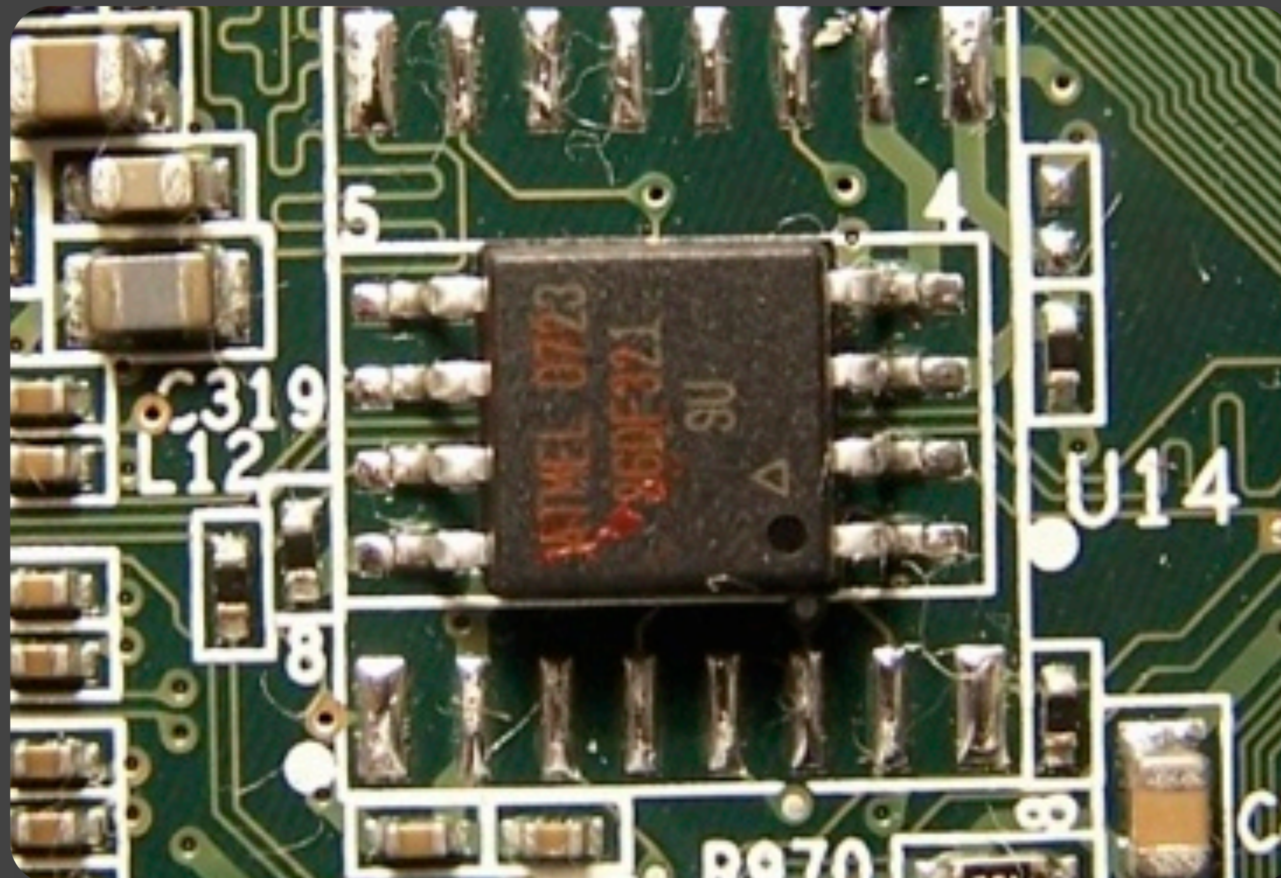
https://t.me/learningnets

Where is the software for the chipset kept?

https://t.me/learningnets

On the SPI-flash chip (the same one used for the BIOS code)

It is a separate chip on a motherboard:



https://t.me/learningnets

Of course one cannot reflash the SPI chip at will!

vPro-compatible systems do not allow unsigned updates to its firmware (e.g. BIOS reflash).

https://t.me/learningnets

But see our talk tomorrow about breaking into the Intel BIOS ;)

Attacking Intel® BIOS

Rafal Wojtczuk and Alexander Tereshkin

Black Hat USA, July 30 2009
Las Vegas, NV

https://t.me/learningnets

Anyway:

- The chipset runs programs.
- The programs are stored in the (well protected) flash memory, together with BIOS firmware.

https://t.me/learningnets

So, what programs run on the chipset?

https://t.me/learningnets

Intel Active Management Technology (AMT)

http://www.intel.com/technology/platform-technology/intel-amt/

https://t.me/learningnets

Intel® Active Management Technology - Mozilla Firefox

File Edit View History Bookmarks Tools Help

http://192.168.0.22:16992/hw-proc.htm

Google

Często odwiedz... Smart Bookmarks Release Notes Fedora Project Red Hat Free Content

Intel® Active Management Technology
Computer: iDBO

System Status
Hardware Information
- System
- Processor
- Memory
- Disk

Event Log
Remote Control
Power Policies
Network Settings
User Accounts

Processor Information

Processor 1

Manufacturer	Intel(R) Corporation
Family	Intel® Pentium® D Processor
Socket	J1PR
Version	Intel(R) Core(TM)2 Duo CPU E8400 @ 3.00GHz
ID	13829424153406539386
Maximum socket speed	4000 MHz
Speed	3000 MHz
Status	Enabled
Upgrade method	Unknown
Populated?	Yes

Done https://t.me/learningnets

Intel® Active Management Technology - Mozilla Firefox

File Edit View History Bookmarks Tools Help

http://192.168.0.22:16992/remote.htm

Często odwiedz... Smart Bookmarks Release Notes Fedora Project Red Hat Free Content

Intel® Active Management Technology
Computer: iDBO

System Status
Hardware Information
- System
- Processor
- Memory
- Disk

Event Log
Remote Control
Power Policies
Network Settings
User Accounts

Remote Control

Power state: On
Send a command to this computer:

- Turn power off*
- Cycle power off and on*
- Reset*

Select a boot option:
- Normal boot
- Boot from local hard drive

***Caution**: These commands may cause user application data loss.

Send Command

Done https://t.me/learningnets

Manageability Commander Tool

File Edit View Help

Network
192.168.0.22 / admin
192.168.0.66 / admin

Intel® AMT Certificate Store - 192.168.0.22

Certificates | Trusted Roots | Revocation

Stored Certificates

Certificate

New... Import... Remove View...

Close

Connect & Control

In this window, you can connect to an Intel® AMT computer. Once connected, you can control the computer remotely, remotely turn on or off the Intel® AMT computer you are connected to, control network policies and filters, boot the computer to a remote drive, view the hardware asset inventory, and read the computer's event log.

Connection | Remote Control | Intel® Management Engine | Networking

Intel® Management Engine

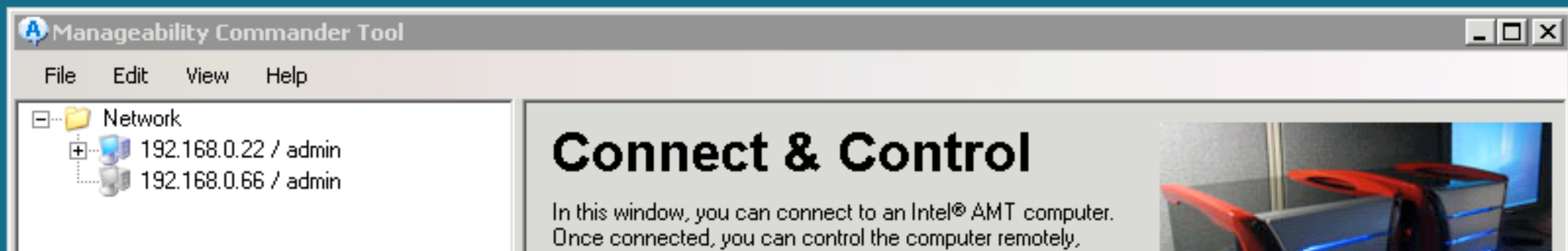
Computer Hostname / Domain	iDBO.somedomain.org
Intel® AMT Version	3.2.1
BIOS Version	JOQ3510J.86A.0933.2008.0707.2248
Intel® Management Engine P	ON in S0
User Accounts	2 User Accounts
Interaction Type	EOI (SOAP) only
Certificate & CRL Store	0 certificate(s), 0 trusted root(s)
Kerberos* Setup	Disabled
Remote Access	Unsupported

Set Time... WSMAN Browser...

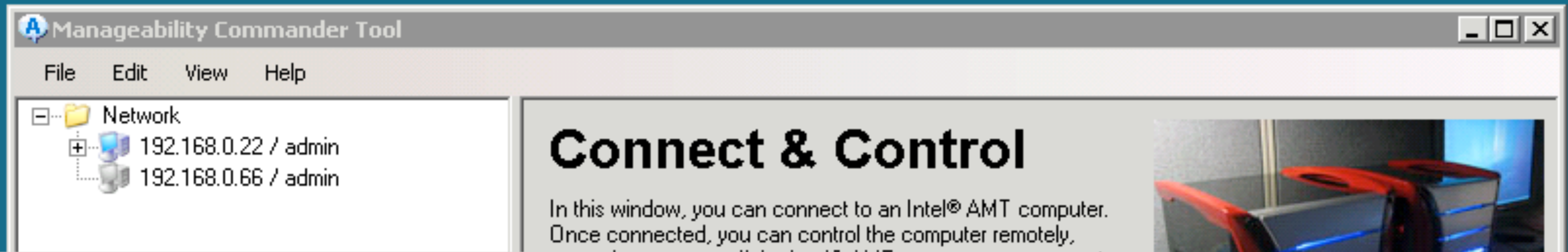
Manageability Commander Tool - v0.6.0937.2

https://t.me/learningnets

https://t.me/learningnets



https://t.me/learningnets



https://t.me/learningnets

If abused, AMT offers powerful backdoor capability:
it can survive **OS reinstall** or other OS change!

https://t.me/learningnets

But AMT is turned off by default...

https://t.me/learningnets

- There are a few methods to enable AMT...
- ... but most require physical presence during the BIOS boot
- We do have ideas how to do it remotely,
- But let's skip it and talk about something better...

https://t.me/learningnets

But turns out that some AMT code is executed **regardless of whether AMT is enabled** in BIOS or not!
And we can hook this code (see later)!

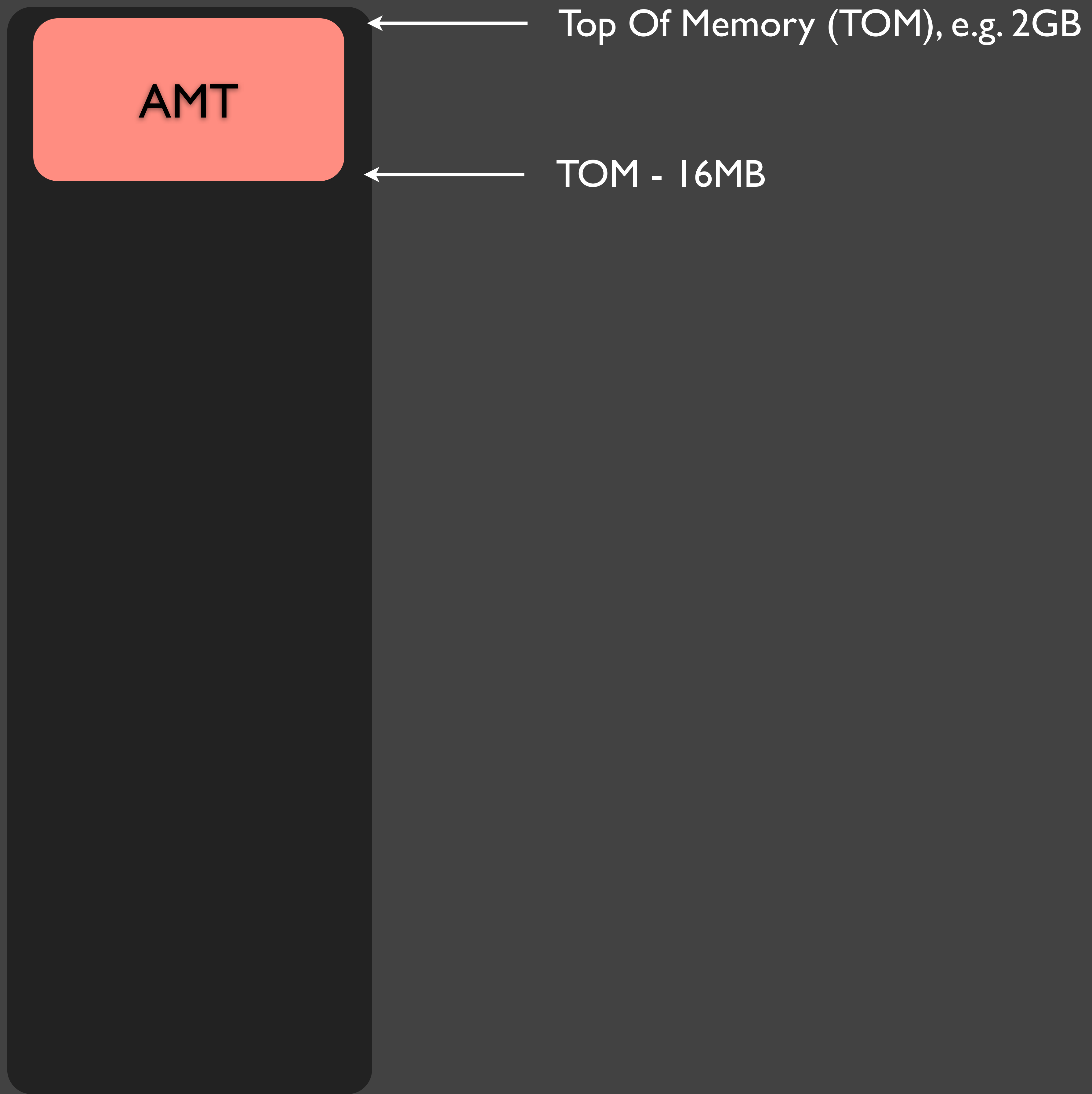
https://t.me/learningnets

Injecting Code into AMT/ME

https://t.me/learningnets

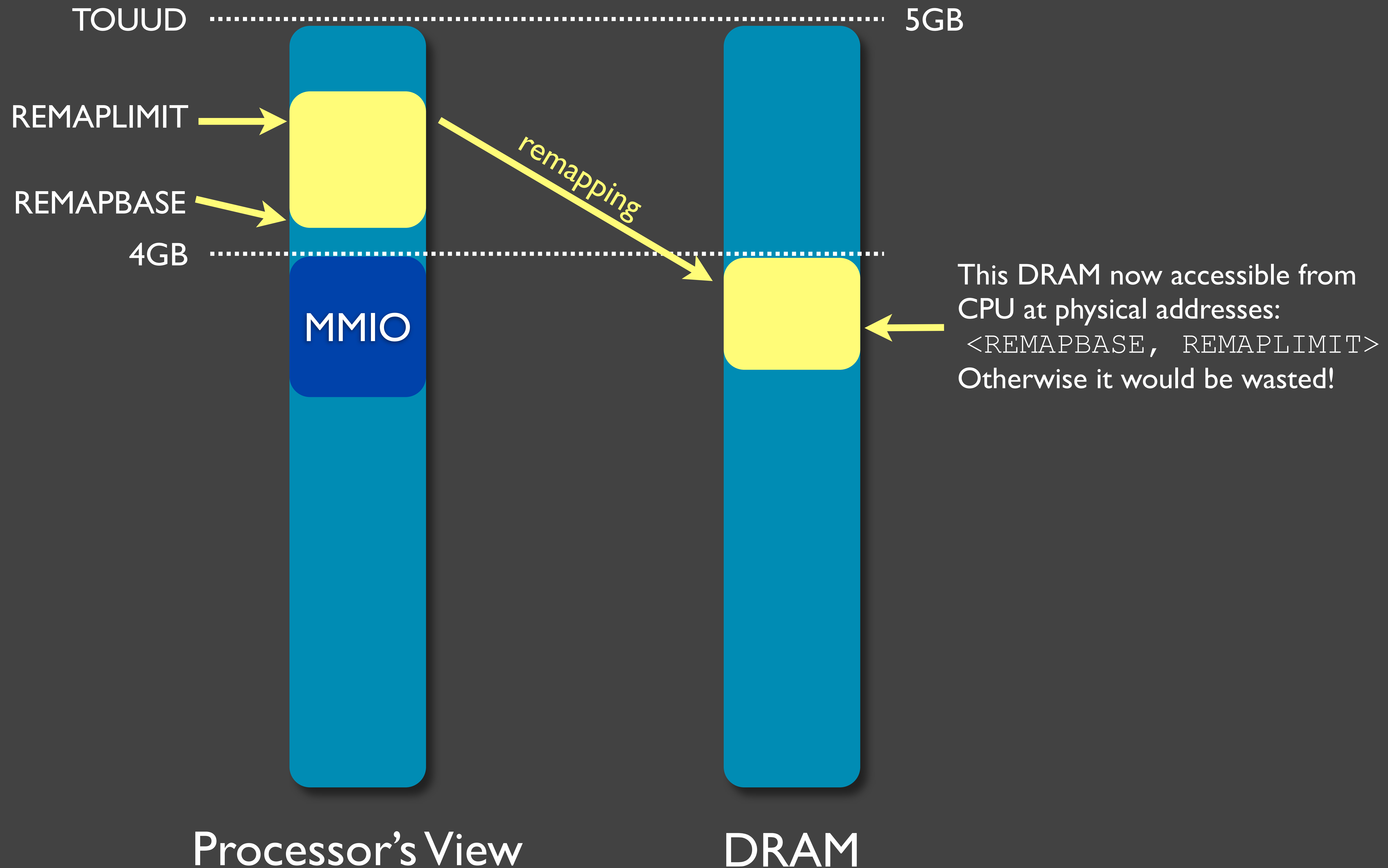
Ok, so how we get our code executed inside AMT/ME environment?

https://t.me/learningnets



https://t.me/learningnets

Memory Remapping on Q35 chipset



https://t.me/learningnets

Applying this to AMT case

https://t.me/learningnets

```
remap_base          = 0x100000000 (4G)
remap_limit         = 0x183ffffff
touud               = 0x184000000
reclaim_mapped_to = 0x7c000000
```

AMT normally at: **0x7f000000**,

Now remapped to : **0x103000000** (and freely accessible by the OS!)

(Offsets for a system with 2GB of DRAM)

https://t.me/learningnets

Fixed? No problem - just revert to the older BIOS!

(turns out no user consent is needed to downgrade Intel BIOS to an earlier version - malware can perfectly use this technique, it only introduces one additional reboot)

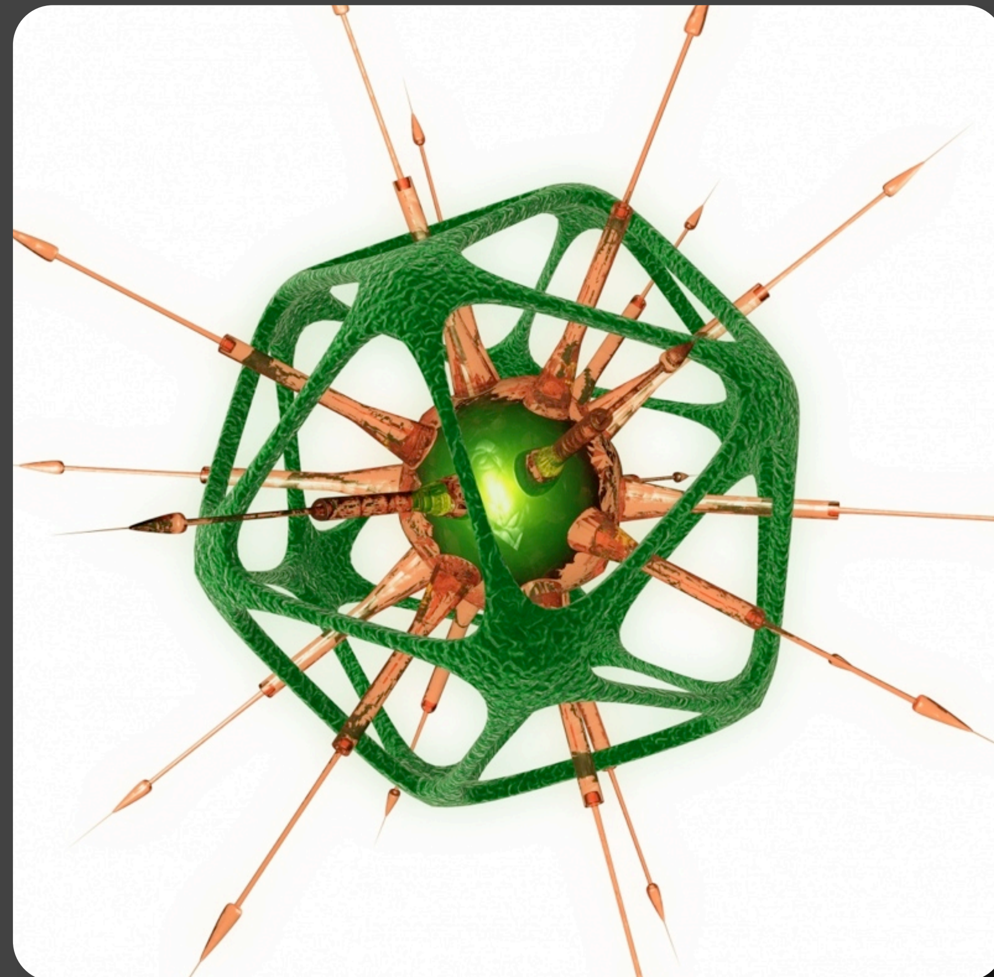
https://t.me/learningnets

How about other chipsets?

https://t.me/learningnets

This attack doesn't work against the Intel Q45-based boards.
The AMT region seems to be **additionally protected**.
(We are investigating how to get access to it...)

https://t.me/learningnets



Writing Useful Ring -3 Rootkits

https://t.me/learningnets

Justifying the "Ring -3" name

https://t.me/learningnets

- **Independent** of main CPU
- Can **access host memory** via DMA (with restrictions)
- Dedicated link to **NIC**, and its filtering capabilities
- Can **force** host OS to **reboot** at any time (and boot the system from the emulated CDROM)
- Active even in **S3 sleep!**

https://t.me/learningnets

Unified execution environment

https://t.me/learningnets

A few words about the ARC4 processor (integrated in the MCH)

- RISC architecture
- 32-bit general purpose registers and memory space
- "Auxiliary" registers space, which is used to access hardware
- On Q35 boards, the `0x01000000–0x02000000` memory range (of the ARC4 processor) is mapped to the top 16MB of host DRAM

https://t.me/learningnets

The ARC compiler suite (arc-gnu-tools) used to be freely available (a few months ago)...

Now it seems to be a commercial product only:

`http://www.arc.com/software/gnutools/`

(we were luckily enough to download it when it was still free)

https://t.me/learningnets

Better portability between different hardware than SMM rootkits

(Unified ARC4 execution environment)

https://t.me/learningnets

Getting our code periodically executed

https://t.me/learningnets

Executable modules found in the AMT memory dump:

(names and numbers taken from their headers)

```
LOADER    : 0x000000..0x0122B8, code: 0x000050..0x0013E0, entry: 0x000050
KERNEL    : 0x0122D0..0x28979C, code: 0x012320..0x05F068, entry: 0x031A10
PMHWSEQ   : 0x2897B0..0x28DDF0, code: 0x289800..0x28CAD8, entry: 0x28A170
QST       : 0x28DE00..0x2A79E8, code: 0x28DE50..0x29B3F4, entry: 0x291B48
OS        : 0x2A7A00..0x88EE28, code: 0x2A7A50..0x5ADA48, entry: 0x4ECC58
ADMIN_CM  : 0x88EE40..0x98CCF8, code: 0x88EE90..0x91A810, entry: 0x8B2994
AMT_CM    : 0x98CD10..0xAA35FC, code: 0x98CD60..0xA2089C, entry: 0x9BB964
ASF_CM    : 0xAA3610..0xAB4DEC, code: 0xAA3660..0xAAD59C, entry: 0xAABC58
```

https://t.me/learningnets

```
01012E60      mov.f lp_count, r2
01012E64      or r4, r0, r1
01012E68      jz.f [blink]
01012E6C      and.f 0, r4, 3
01012E70      shr r4, r2, 3
01012E74      bnz loc_1012EFC
01012E78      lsr.f lp_count, r4
01012E7C      sub r1, r1, 4
01012E80      sub r3, r0, 4
01012E84      lpnz loc_1012EA8
01012E88      ld.a r4, [r1+4]
01012E8C      ld.a r5, [r1+4]
01012E90      ld.a r6, [r1+4]
01012E94      ld.a r7, [r1+4]
01012E98      st.a r4, [r3+4]
01012E9C      st.a r5, [r3+4]
01012EA0      st.a r6, [r3+4]
01012EA4      st.a r7, [r3+4]
01012EA8      bc.d loc_1012ED8
```

This function from the KERNEL module is called quite often probably by a timer interrupt handler.

https://t.me/learningnets

Accessing the host memory

https://t.me/learningnets

PROGRAMMING uC DMA WITH BARE HANDS

```
arc> dump 0x2000000
02000000: 00000000 00000000 000
02000010: 00000000 00000000 000
02000020: 00000000 00000000 000
02000030: 00000000 00000000 000
02000040: 00000000 00000000 000
02000050: 00000000 00000000 0000000        ¦ ................
02000060: 00000000 00000000 000000  00000000 ¦ ................
02000070: 00000000 00000000 0000  00 00000000 ¦ ................
arc> arc:dma(1,0x73000,0,0x2000000,64)
Transferred 64 B of data from Host 0x00073000
General Status = 1
02000000: fa 55 8b ec 81 ec 84 00 00 00 89 45 b4 89 5d b8 ¦ .U.........E..].
02000010: 89 4d bc 89 55 c0 89 75 cc 89 7d d0 0f 20 d0 89 ¦ .M..U..u..}.. ..
02000020: 45 c4 8d 45 f8 50 68 02 44 00 00 e8 b8 08 00 00 ¦ E..E.Ph.D.......
02000030: 8d 45 d4 50 68 1e 68 00 00 e8          00 00 8d 45 ¦ .E.Ph.h........E
02000040: 00 00 00 00 00 00 00 00 00 00 00 00
02000050: 00 00 00 00 00 00 00 00 00 00 00 00
02000060: 00 00 00 00 00 00 00 00 00 00 00 00
02000070: 00 00 00 00 00 00 00 00 00 00 00 00
arc> dump 0x2000000
02000000: ec8b55fa 0084ec81 45890000 b85d89b4 ¦ .U.........E..].
02000010: 89bc4d89 7589c055 d07d89cc 89d0200f ¦ .M..U..u..}.. ..
```

Programming internal DMA hardware in JTAG debugger to copy 64 bytes from 0x73000 host phys addr to internal memory

DMA-ed malicious VM Exit handler

source: **Yuriy Bulygin, Intel**, Black Hat USA 2008

https://t.me/learningnets

AMT code can access host memory via DMA

https://t.me/learningnets

But how to program it? Of course this is not documented anywhere...

(And the rootkit can't just use ARC4 JTAG debugger, of course)

https://t.me/learningnets

Idea of how to learn how AMT code does DMA to host memory

https://t.me/learningnets

We know that AMT emulates "Virtual CDROM" that might be used by remote admin to boot system into OS installer...

https://t.me/learningnets

...we can also debug the AMT code using function hooking and counters...

https://t.me/learningnets

counter_X++

An AMT function_X...

counter_Y++

An AMT function_Y...

Our debugging stubs

(The counter_* variables are also located in the AMT memory -- we read them using the remapping trick)

Most of the functions can be spotted by looking for the following prologue signature:

```
04 3E 0E 10          st blink, [sp+4]
```

https://t.me/learningnets

So we can boot off AMT CDROM e.g. a Linux OS and try to access the AMT virtual CDROM...

https://t.me/learningnets

...at the same time we trace which AMT code has been executed.

https://t.me/learningnets

Q: How is the AMT CDROM presented to BIOS/OS?
A: As a PCI device...

https://t.me/learningnets

root@dom0:~

```
[root@q35 ~]# lspci -s 00:03.2 -v
00:03.2 IDE interface: Intel Corporation PT IDER Controll
er (rev 02) (prog-if 85 [Master SecO PriO])
        Subsystem: Intel Corporation Unknown device 4f4a
        Flags: bus master, 66MHz, fast devsel, latency 0,
 IRQ 9
        I/O ports at 2480 [size=8]
        I/O ports at 24a4 [size=4]
        I/O ports at 2478 [size=8]
        I/O ports at 24a0 [size=4]
        I/O ports at 2440 [size=16]
        Capabilities: [c8] Power Management version 3
        Capabilities: [d0] Message Signalled Interrupts:
Mask- 64bit+ Queue=0/0 Enable-

[root@q35 ~]#
```

https://t.me/learningnets

We have traced BIOS accesses to AMT CDROM during boot; it turned out that BIOS did not use DMA transfers, it used PIO data transfers :(

https://t.me/learningnets

Fortunately, the above PCI device fully conforms to ATAPI specifications; as a result, it is properly handled by the Linux **`ata_generic.ko`** driver

(if loaded with `all_generic_ide` flag)

https://t.me/learningnets

root@f9q35:~

```
f9q35 kernel: ACPI: PCI Interrupt 0000:00:03.2[C] -> GSI 18 (level, lo
w) -> IRQ 18
f9q35 kernel: scsi6 : ata_generic
f9q35 kernel: scsi7 : ata_generic
f9q35 kernel: ata7: PATA max UDMA/100 cmd 0x2480 ctl 0x24a4 bmdma 0x24
40 irq 18
f9q35 kernel: ata8: PATA max UDMA/100 cmd 0x2478 ctl 0x24a0 bmdma 0x24
48 irq 18
f9q35 kernel: ata7.00: ATAPI: Intel Virtual LS-120 Floppy   UHD Floppy
, 1.00, max UD
f9q35 kernel: ata7.01: ATAPI: Intel Virtual CD, 1.00, max UDMA/100
f9q35 kernel: ata7.00: configured for UDMA/100
f9q35 kernel: ata7.01: configured for UDMA/100
f9q35 kernel: scsi 6:0:0:0: Direct-Access     Intel    Virtual Floppy
  1.00 PQ: 0 A
f9q35 kernel: sd 6:0:0:0: [sdb] Attached SCSI removable disk
f9q35 kernel: sd 6:0:0:0: Attached scsi generic sg2 type 0
f9q35 kernel: scsi 6:0:1:0: CD-ROM            Intel    Virtual CD
  1.00 PQ: 0 A
[root@f9q35 ~]#
[root@f9q35 ~]#
[root@f9q35 ~]#
[root@f9q35 ~]#
```

https://t.me/learningnets

We can instruct `ata_generic.ko` whether to use or not DMA for the virtual CDROM accesses

we can do the **diffing** between two traces and find out which AMT code is responsible for DMA :)

https://t.me/learningnets

This way we found (at least one) way to do DMA from AMT to the host memory

https://t.me/learningnets

```c
struct dmadesc_t {
    unsigned int src_lo;
    unsigned int src_hi;
    unsigned int dst_lo;
    unsigned int dst_hi;
    unsigned int count;
    unsigned int res1;
    unsigned int res2;
    unsigned int res3;
} dmadesc[NUMBER_OF_DMA_ENGINES];
```

```c
// SR instruction: Store to Auxiliary Register
void sr(unsigned int addr, unsigned int value) {
  asm("sr r1, [r0]");
}
```

```c
void dma_amt2host(unsigned int idx, /* the id of DMA engine */
     unsigned int amt_source_addr,
     unsigned int host_dest_addr,
     unsigned int transfer_length)
{
    unsigned int srbase = 0x5010 + 4 * idx;
    memset(&dmadesc[idx], 0, sizeof dmadesc[idx]);
    dmadesc[idx].src_lo = amt_source_addr;
    dmadesc[idx].dst_lo = host_dest_addr;
    dmadesc[idx].count = transfer_length;
    sr(srbase + 1, &dmadesc[idx]);
    sr(srbase + 2, 0);
    sr(srbase + 3, 0);
    sr(srbase + 0, 0x189);
}
```

https://t.me/learningnets

10.5.2 CMD—Command Register

B/D/F/Type: 0/3/2/PCI
Address Offset: 4-5h
Default Value: 0000h
Access: RO, R/W
Size: 16 bits

Reset: Host System reset or D3->D0 transition of function.

This register provides basic control over the device's ability to respond to and perform Host system related acesses.

Intel® Manageability Engine Subsystem Registers

Bit	Access	Default Value	RST/ PWR	Description
2	R/W	0b	Core	**Bus Master Enable (BME):** This bit controls the PT function's ability to act as a master for data transfers. This bit does not impact the generation of completions for split transaction commands.
1	RO	0b	Core	**Memory Space Enable (MSE):** PT function does not contain target memory space.
0	R/W	0b	Core	**I/ O Space enable (IOSE):** This bit controls access to the PT function's target I/O space.

source: intel.com

https://t.me/learningnets

Unfortunately, upon reboot, the **BME bit** for IDER device is cleared, which prevents DMA transfers...

However: rootkit can detect that a host reboot is in progress (because DMA transfers fail to work), and force reboot to AMT CDROM, that will set BME bit and resume OS boot

https://t.me/learningnets

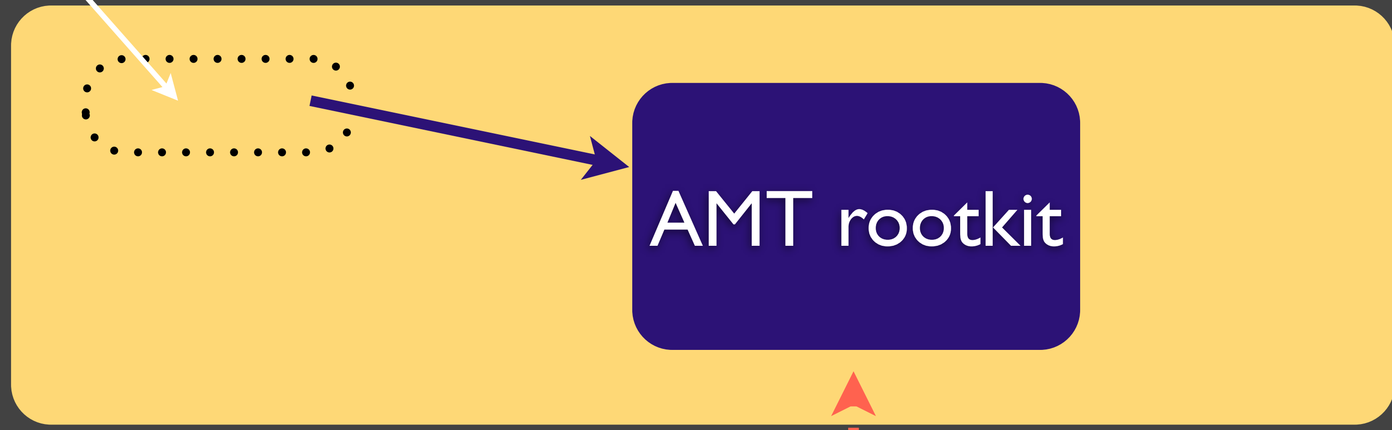
- Possibly, using other ME PCI device bypasses the BME limitation?
- (there is nothing about BME bit in Yuriy Bulygin's talk on DeepWatch from BH US 2008)
- This would allow for SRTM bypass (AMT could inject/replace already-measured code while it's executing)
- But we haven't found any other way to do DMA without BME so far...

https://t.me/learningnets

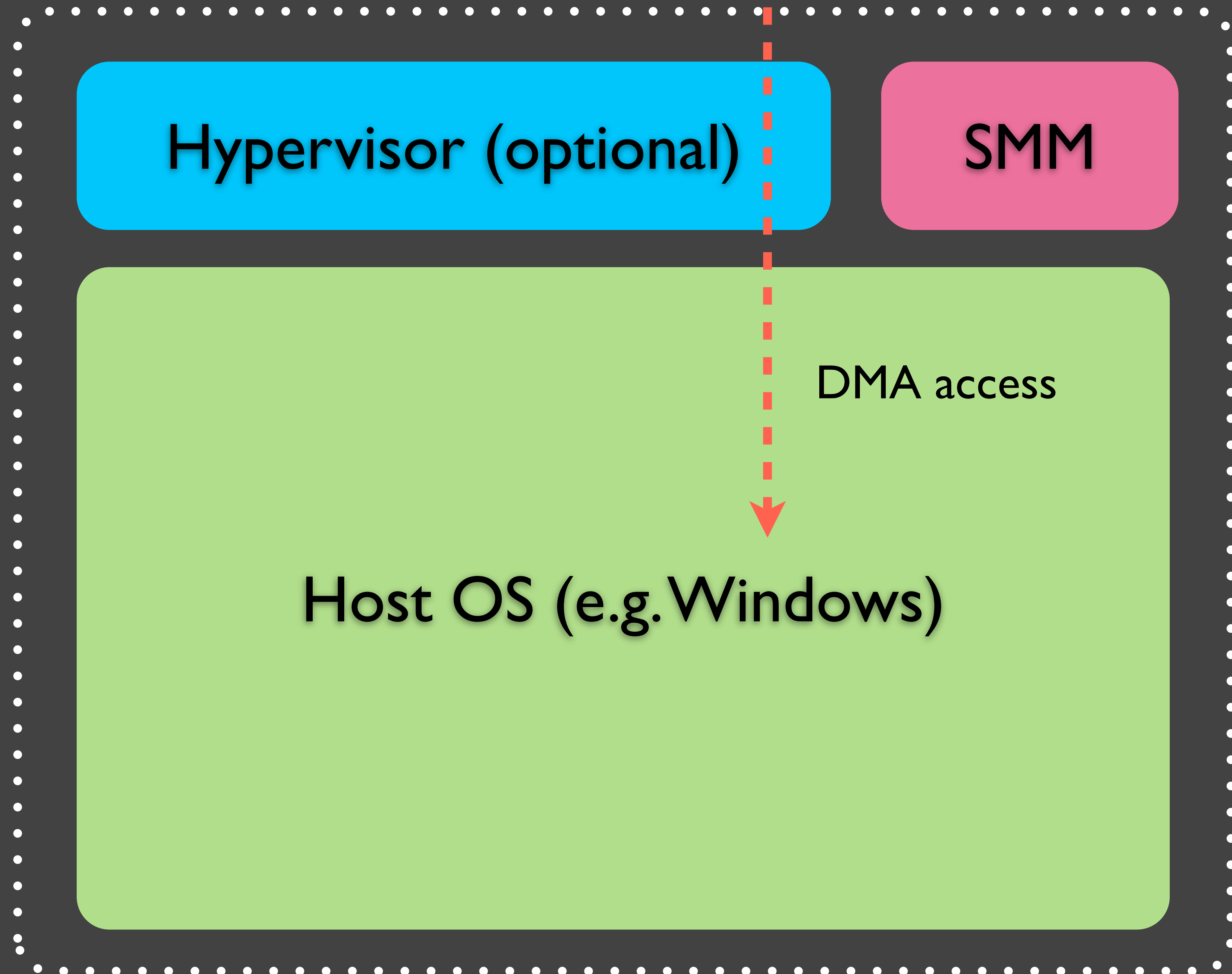
Putting it all together

https://t.me/learningnets

Hooked AMT function that is executed periodically (regardless of whether AMT is enabled or not in the BIOS)



Chipset ME/AMT:
All code executed by the chipset's ARC4 processor, even if the host in sleep mode!



Host Memory:
all code executed on the host CPU(s)

https://t.me/learningnets

What about VT-d? Can the OS protect itself against AMT rootkit?

https://t.me/learningnets

DMA REMAPPING

- VT-d capable chipsets have one or more DMA-remapping engines virtualizing Directed I/O access [12]
- Internal devices are also a subject to DMA-remapping
- Chipset has dedicated register-set for each DMA-remap unit accessible by software as MMIO range which software can use to protect certain memory regions from certain I/O devices
- Rootkit can create DMA-remapping page tables to translate addresses of DMA requests issued by embedded uC (identified by its PCI B/D/F) to different host physical addresses
  - or read/write protect entries in DMAr pages tables
  - or mark context-entry as not Present to cause translation fault
  - or enable PLMR/PHMR DMA-protected regions to prevent any DMA
- And relocate code/data (VMExit handler, VMCS ..) to memory protected by DMAr page tables or to PMR regions

intel

source: **Yuriy Bulygin, Intel**, Black Hat USA 2008

https://t.me/learningnets

SO WHAT CAN WE DO ABOUT THIS ??

- DMA-remapping unit can distinguish DMA requests issued by DeepWatch internal device function inside embedded uC
- by its requester id from DMA requests issued by other internal functions
- and not translate them
- Or disable and lock DMA-remapping of DeepWatch device function if DeepWatch is used
- And allow only trusted software like SMX authenticated code modules (Intel® TXT) to enable and program DMA-remap engine for DeepWatch

intel

source: **Yuriy Bulygin, Intel**, Black Hat USA 2008

https://t.me/learningnets

So, if Intel allowed its AMT/ME code to bypass VT-d (in order to allow rootkit detectors in the chipset), then our AMT rootkit would automatically gain ability to bypass VT-d as well!

https://t.me/learningnets

We have verified that Xen 3.3+ uses VT-d in order to protect its own hypervisor and consequently our AMT rootkit is not able to access this memory of Xen hypervisor

(But still, if ME PCI devices are not delegated to a driver domain, then we can access dom0 memory)

https://t.me/learningnets

Powerful it is, the VT-d

https://t.me/learningnets

Still, an AMT rootkit can, if detected that it has an opponent that uses VT-d for protection, do the following:

- Force OS reboot
- Force booting from Virtual CDROM
- Use its own image for the CDROM that would infect the OS kernel (e.g. xen.gz) and disable the VT-d there

https://t.me/learningnets

How to protect against such scenario?

https://t.me/learningnets

Via Trusted Boot, e.g. SRTM or DRTM (Intel TXT)

(Keep in mind that we can bypass TXT though, if used without STM, and there is still no STM available as of now)

https://t.me/learningnets

Final Thoughts

https://t.me/learningnets

We *do* like many of the new Intel technologies (VT-x, VT-d, TXT), ...

https://t.me/learningnets

But AMT is different in that it can potentially be greatly abused by the attacker

(VT-d or TXT can potentially be bypassed, but they cannot *help* the attacker!)

https://t.me/learningnets

But keep in mind that our attack doesn't work on the latest Q45 chipsets - a sign that Intel treats the security seriously...

https://t.me/learningnets

You do not want this privileged code to fall into enemy's hand, do you?

source: http://freemasonry.bcy.ca/anti-masonry/

https://t.me/learningnets

http://invisiblethingslab.com

ITL

INVISIBLE THINGS LAB

INVISIBLE THINGS LAB

https://t.me/learningnets