# Preventing and Detecting Xen Hypervisor Subversions

Joanna Rutkowska & Rafał Wojtczuk
Invisible Things Lab

Black Hat USA 2008, August 7th, Las Vegas, NV

# Xen 0wning Trilogy

# Part Two

Previously on Xen 0wning Trilogy...

## Part 1: "Subverting the Xen Hypervisor"

by Rafal Wojtczuk (Invisible Things Lab)

- ✓ Hypervisor attacks via DMA
  - ✓ TG3 network card "manual" attack
  - ✓ Generic attack using disk controller
- ✓ "Xen Loadable Modules" framework :)
- ✓ Hypervisor backdooring
  - ✓ "DR" backdoor
  - ✓ "Foreign" backdoor

Now, in this part...

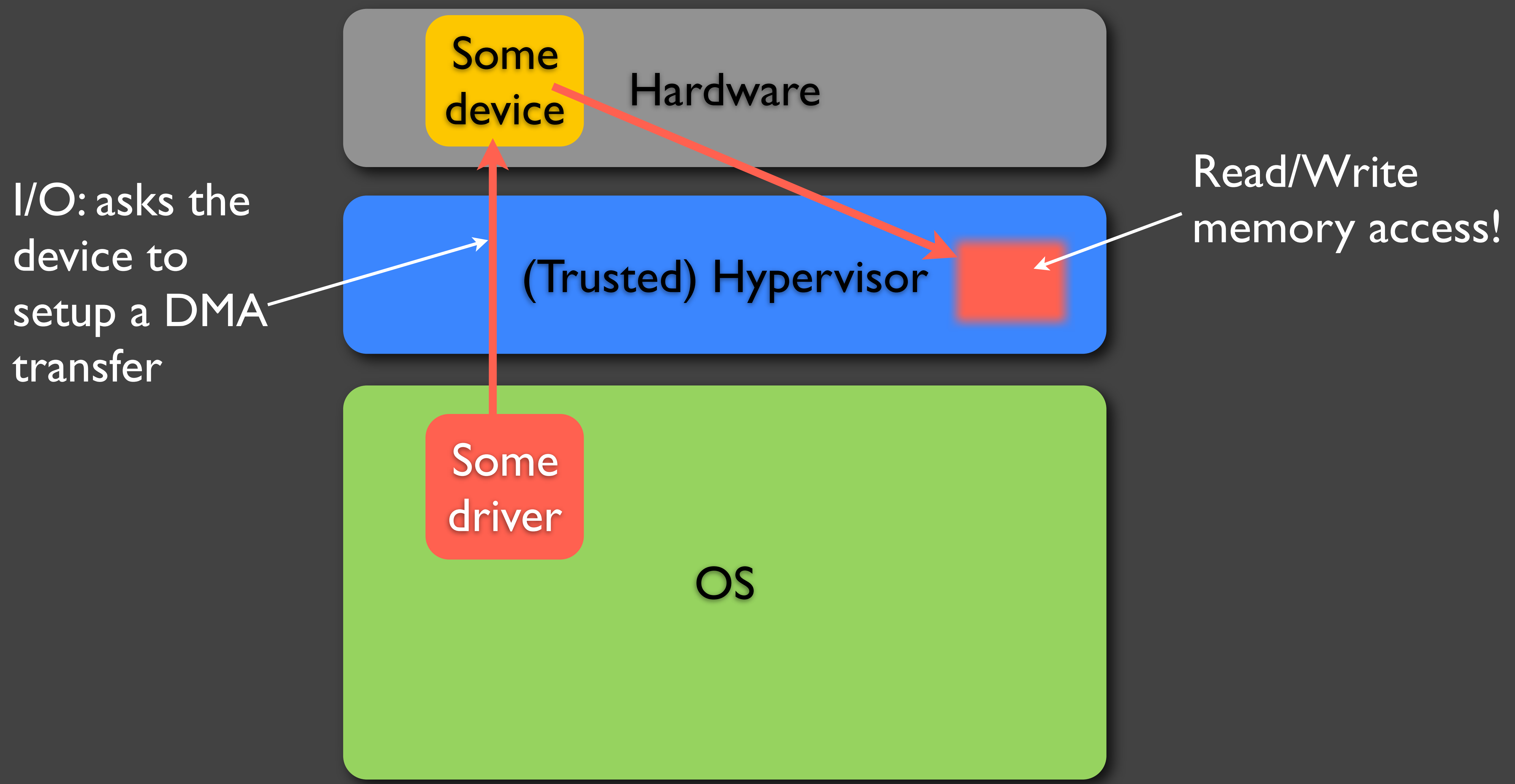
1 **Protecting** the (Xen) hypervisor

2 … and how the **protection fails**

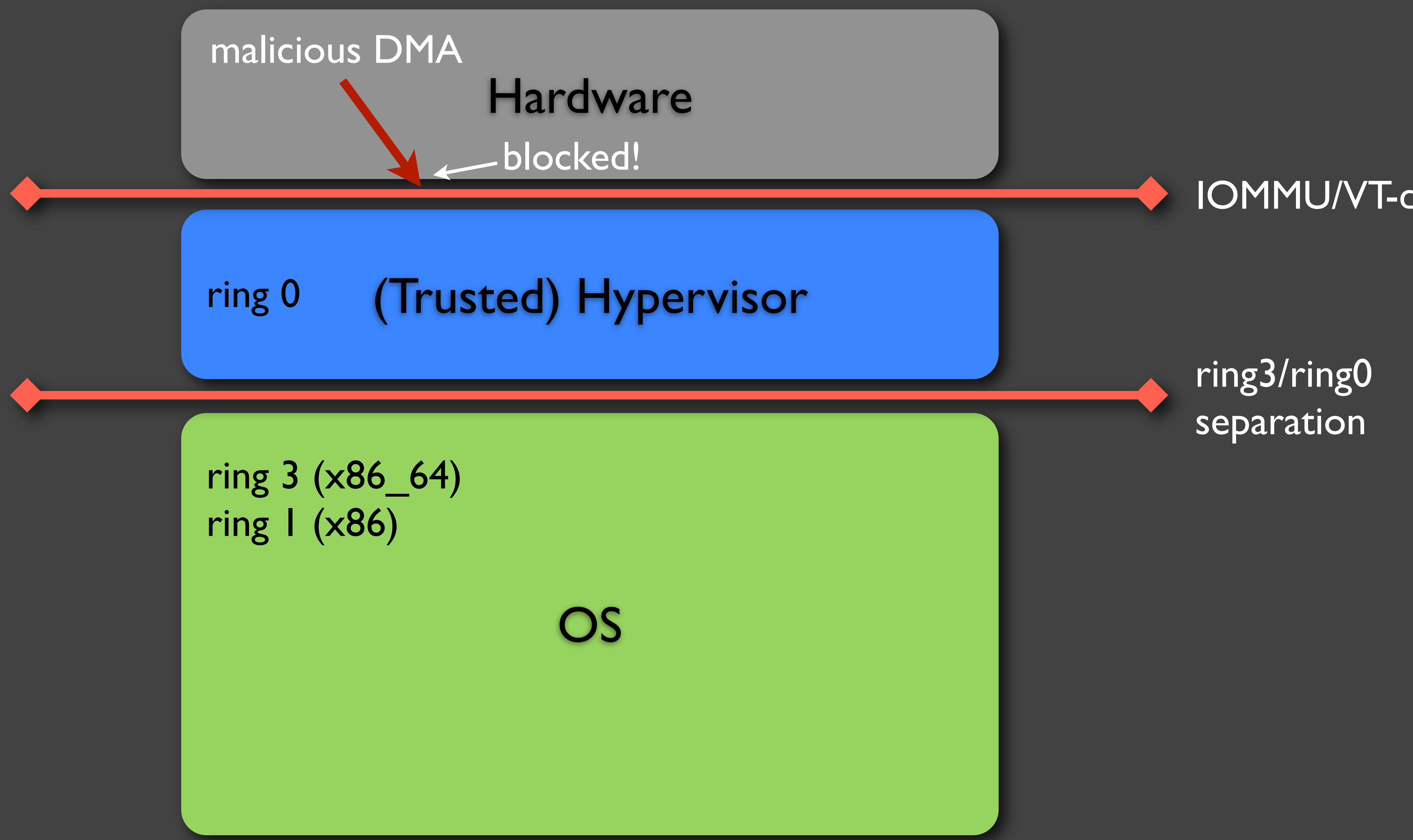
3 Checking (Xen) **hypervisor integrity**

4 … and **challenges** with integrity scanning

# Dealing with DMA attacks

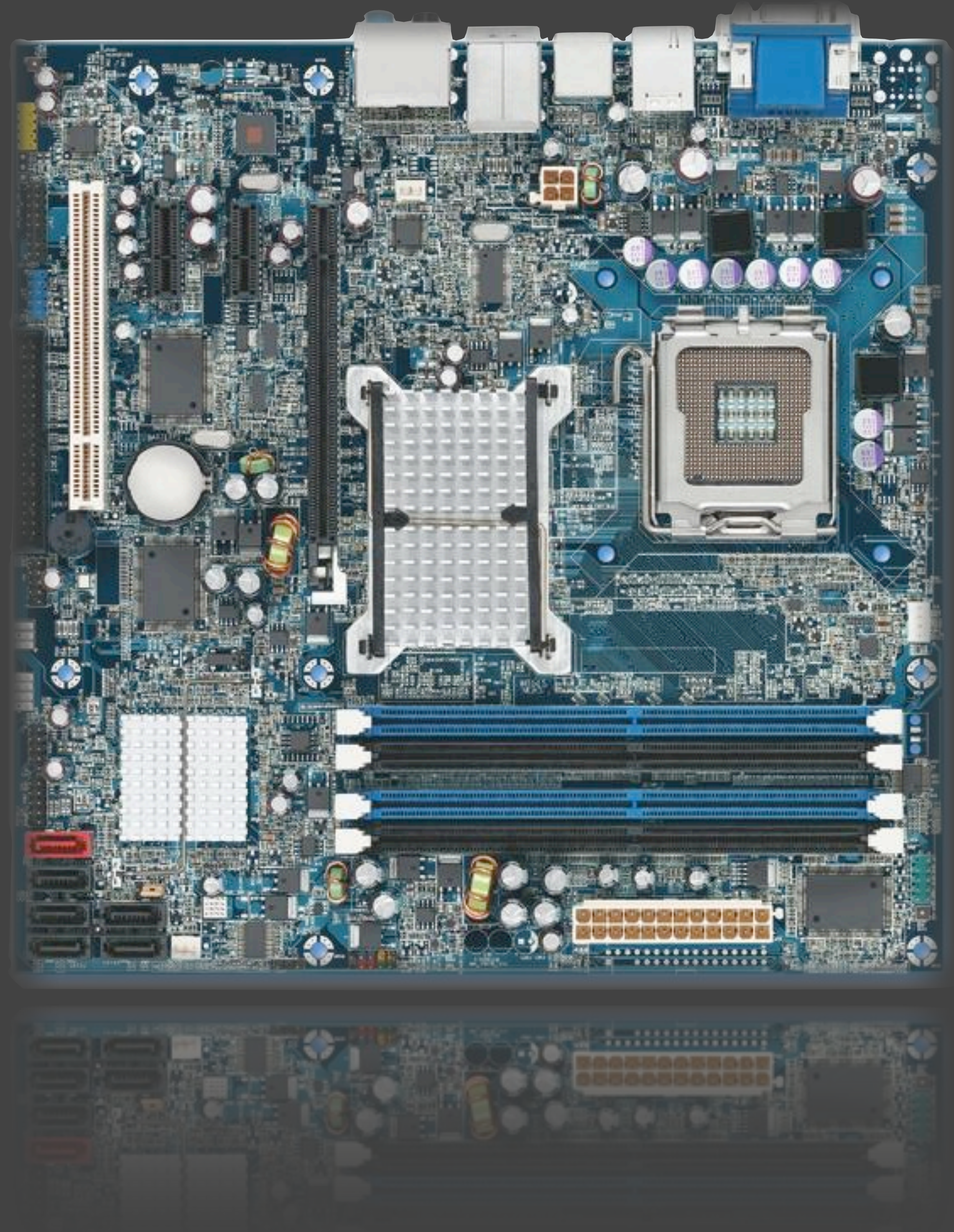

Some
device
Hardware
I/O: asks the
device to
setup a DMA
transfer
(Trusted) Hypervisor
Read/Write
memory access!
Some
driver
OS

# Xen and VT-d


malicious DMA
Hardware
blocked!
IOMMU/VT-d
ring 0
(Trusted) Hypervisor
ring3/ring0
separation
ring 3 (x86_64)
ring 1 (x86)
OS

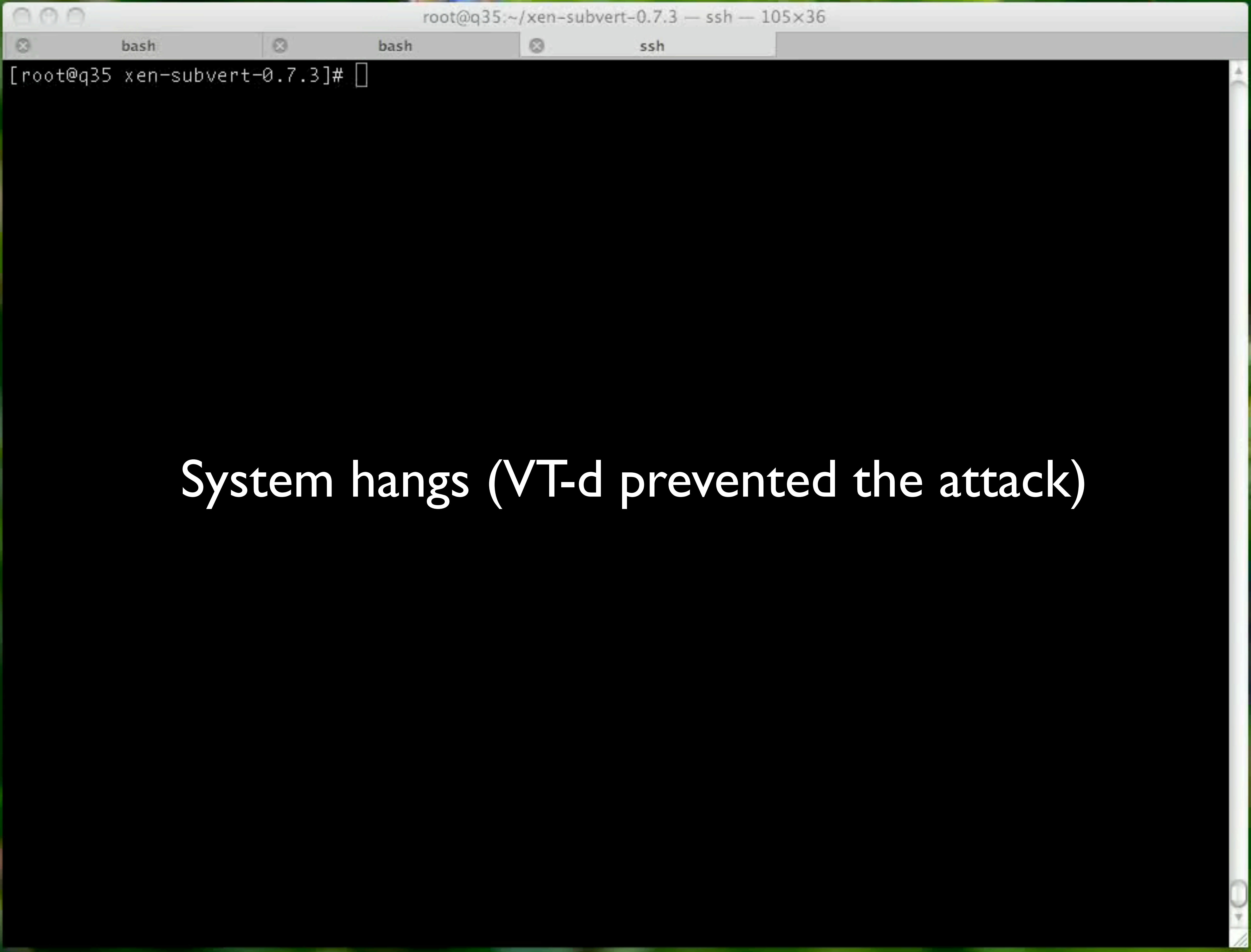
```c
static int intel_iommu_domain_init(struct domain *d)
{
    /.../

    if ( d->domain_id == 0 )
    {
        extern int xen_in_range(paddr_t start, paddr_t end);
        extern int tboot_in_range(paddr_t start, paddr_t end);

        /*
         * Set up 1:1 page table for dom0 except the critical segments
         * like Xen and tboot.
         */
        for ( i = 0; i < max_page; i++ )
        {
            if ( xen_in_range(i << PAGE_SHIFT_4K, (i + 1) << PAGE_SHIFT_4K) ||
                 tboot_in_range(i << PAGE_SHIFT_4K, (i + 1) << PAGE_SHIFT_4K) )
                continue;

            iommu_map_page(d, i, i);
        }

        setup_dom0_devices(d);
        setup_dom0_rmrr(d);

        iommu_flush_all();

        /.../

    }

    return 0;
}
```

Rafal's DMA attack (speech #1) will not work on Xen 3.3 running on Q35 chipset!

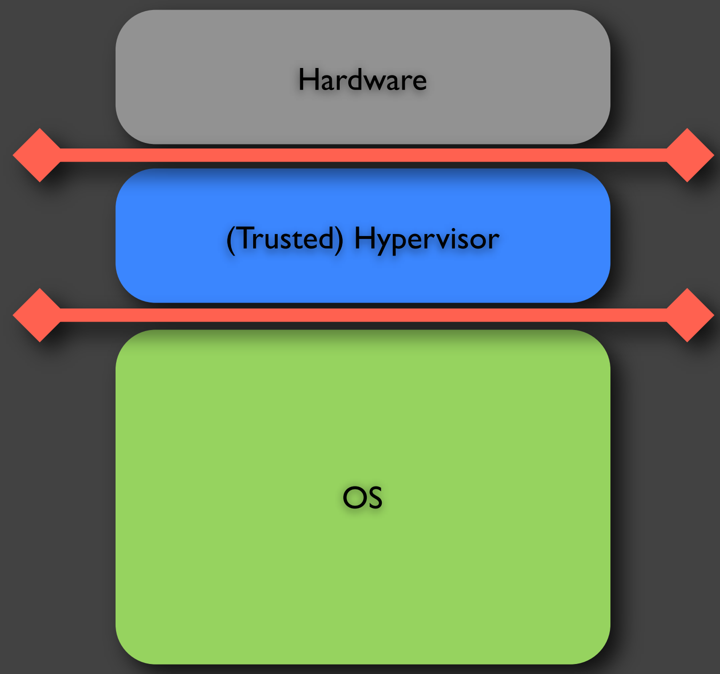


- ✓ Intel Core 2 Duo/Quad
- ✓ Up to 8GB RAM
- ✓ TPM 1.2
- ✓ Q35 Express chipset
- ✓ VT-d (IOMMU)

Intel DQ35JO motherboard: First IOMMU for desktops!
(available in shops since around October 2007)


root@q35:~/xen-subvert-0.7.3 — ssh — 105×36
bash
bash
ssh
[root@q35 xen-subvert-0.7.3]#
System hangs (VT-d prevented the attack)

So, how to get around?




So, how to get around?

Break ring3/ring0 separation?

Break VT-d protection?

None of them! :)

The slide has been removed upon request from Intel.

(it will be published after the patch is available from Intel)

The slide has been removed upon request from Intel.

(it will be published after the patch is available from Intel)

The slide has been removed upon request from Intel.

(it will be published after the patch is available from Intel)

The slide has been removed upon request from Intel.

(it will be published after the patch is available from Intel)

The slide has been removed upon request from Intel.

(it will be published after the patch is available from Intel)

The slide has been removed upon request from Intel.

(it will be published after the patch is available from Intel)

Demo: modifying Xen 3.3 hypevisor from Dom0 root@q35:~ — ssh — 105×36

bash | bash | ssh

```
[root@q35 ~]# 
```

This attack is not limited to Q35 chipsets only!

This attack can also be used to modify SMM handler on the fly, without reboot!

So, whose fault it is?

## Xen's fault?

- Allowing Dom0/Driver domains to access some chipset registers *might* be needed for some reasons... (Really?)
- But Xen cannot know everything about the chipset registers and features!

## Chipset's fault?

- Maybe chipset should do some basic validation of XXX

The details have been removed upon request from Intel.
(it will be published after the patch is available from Intel)

## BIOS’s fault?

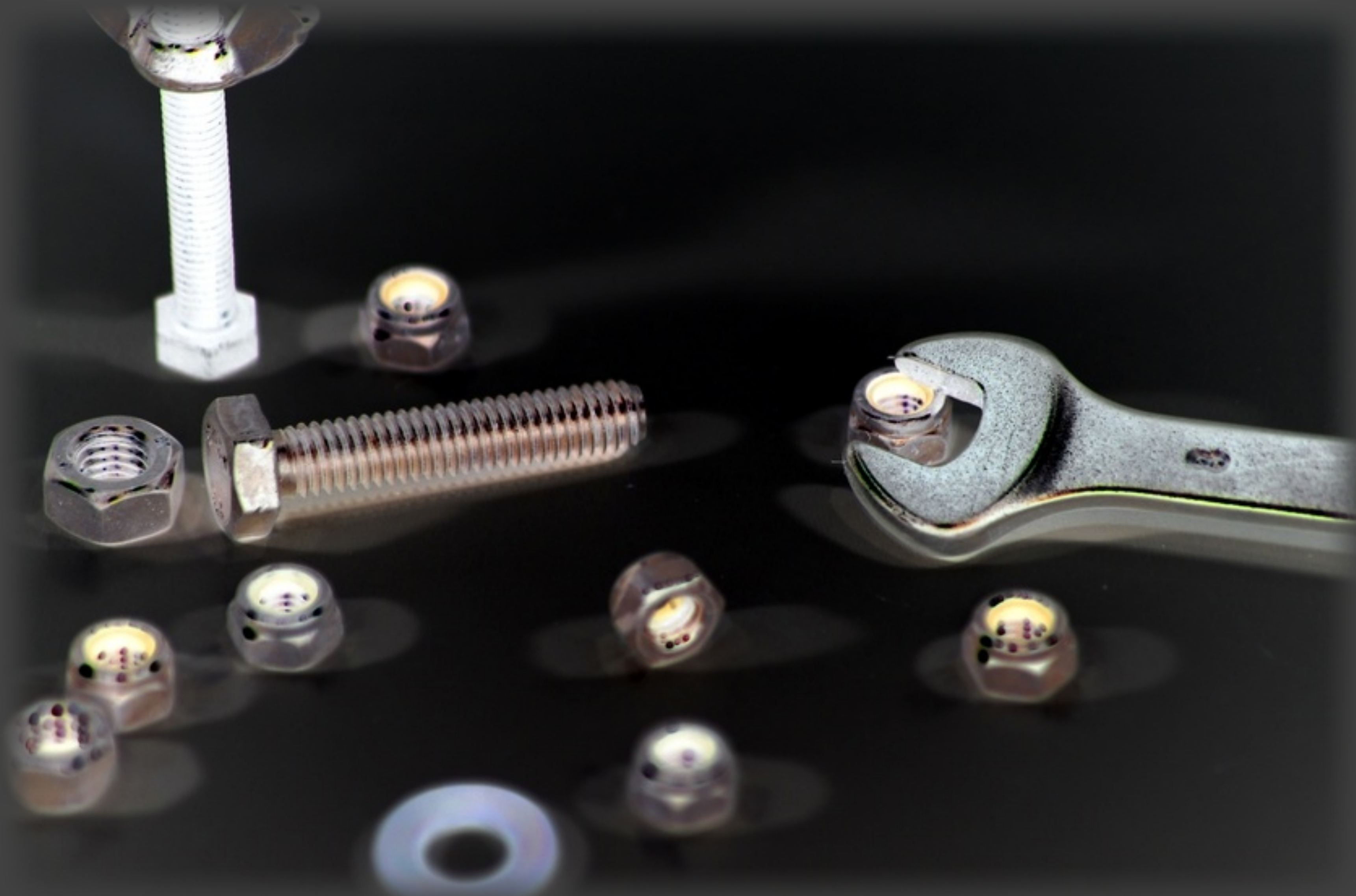
- Intel told us that using a special lock mechanism is recommended in the Intel’s *BIOS Specification (*)*
- Obviously, we’re not talking about D_LCK!
- That lock should prevent our attack
- So, this seems to be the BIOS Writer’s fault in the end...

(*)This document is available only to Intel partners (i.e. BIOS vendors).

# Related attacks

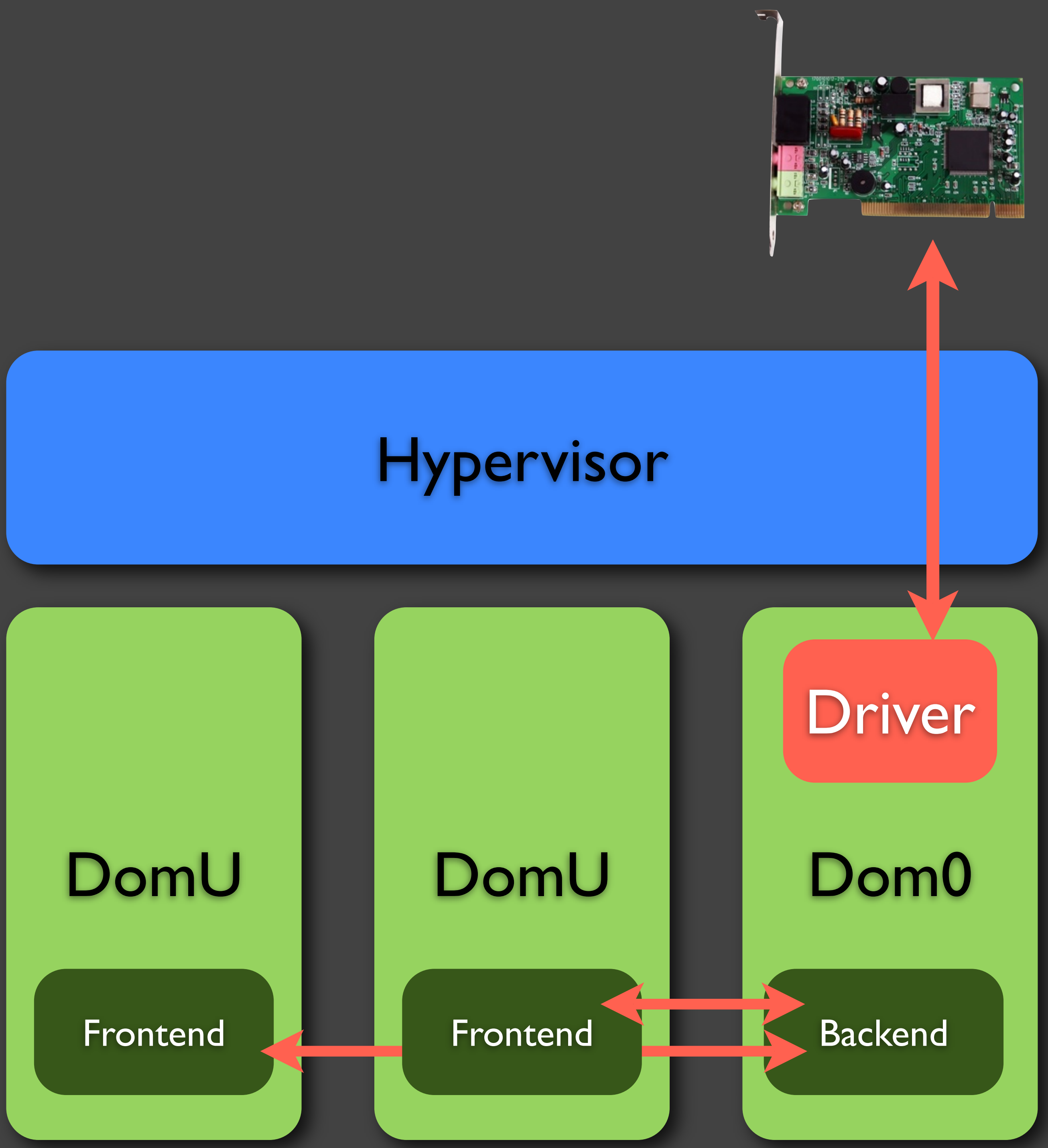
- Loic Duflot (2006) - jump to SMM and then to kernel from there (against OpenBSD securelevel). Now prevented by most BIOSes (thanks to the D_LCK bit set).
- Sun Bing (2007) - exploit TOP_SWAP feature of some Intel chipsets to load malicious code before the BIOS locks the SMM and get your code into SMM. But this requires reboot. Now prevented by BIOSes setting the BILD lock.

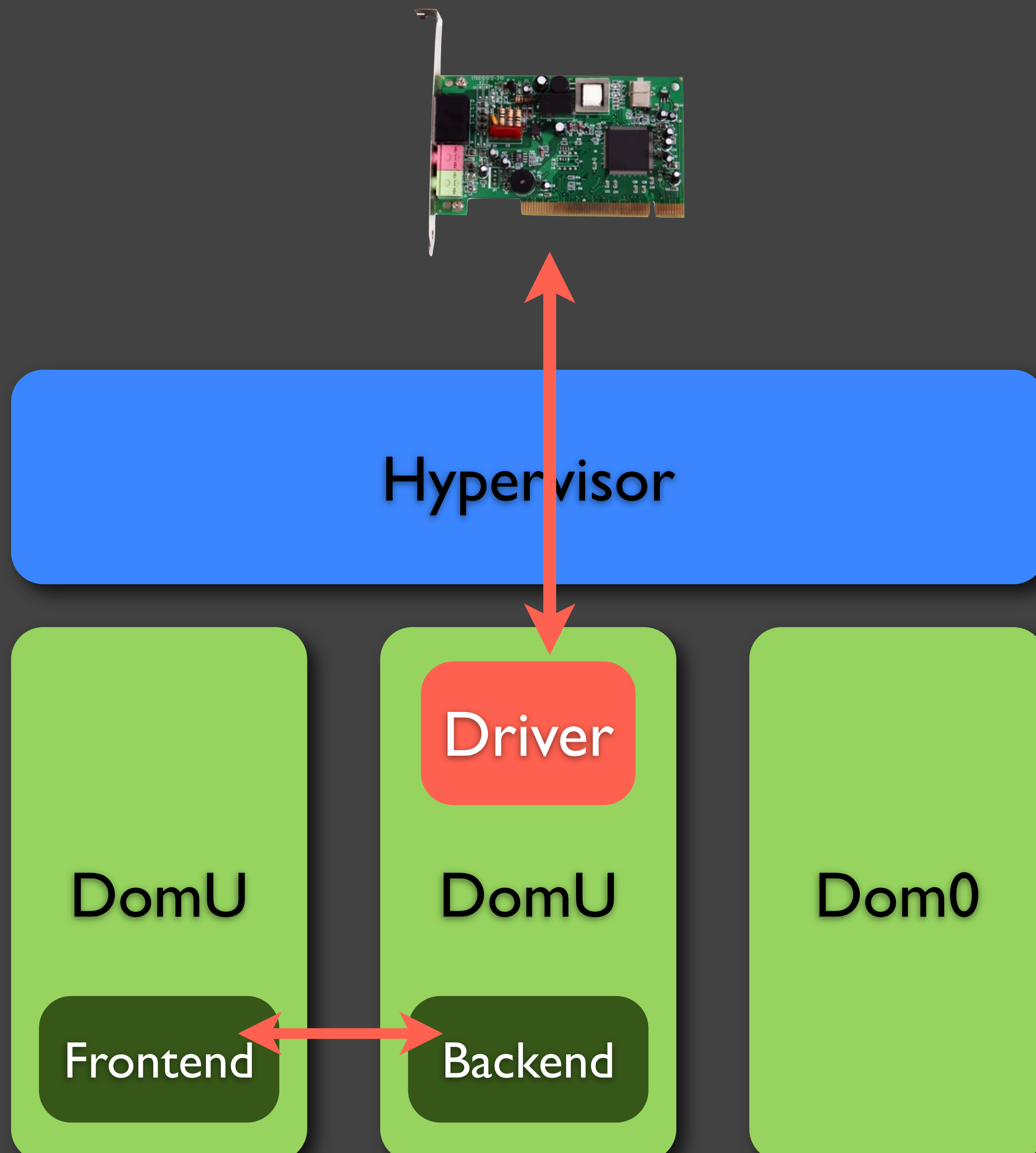
Lesson: protecting hypervisor memory is hard!



# “Domain 0” Disaggregation

# Driver domains

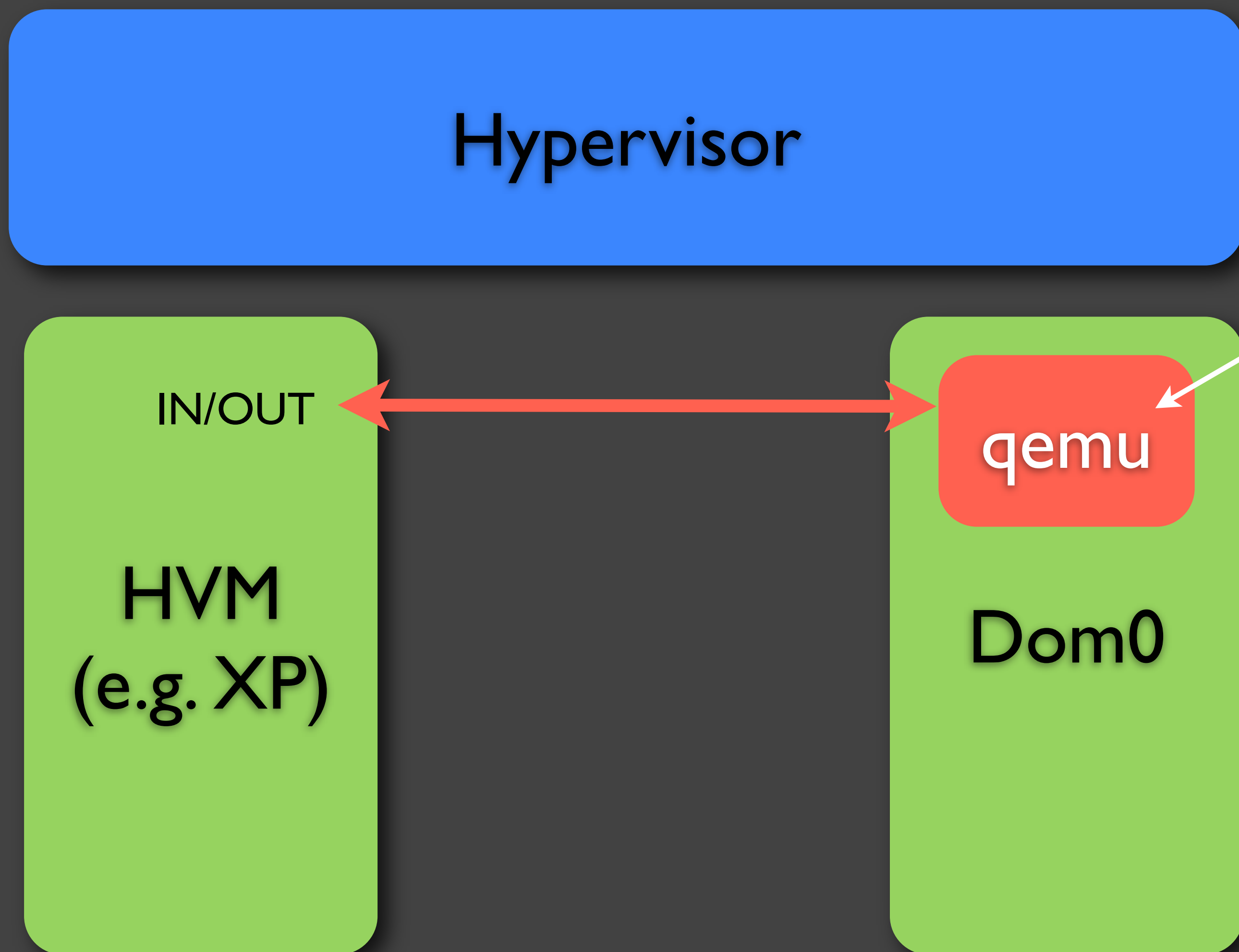

Hypervisor
Driver
DomU
DomU
Dom0
Frontend
Frontend
Backend


Hypervisor
Driver
DomU
DomU
Dom0
Frontend
Backend

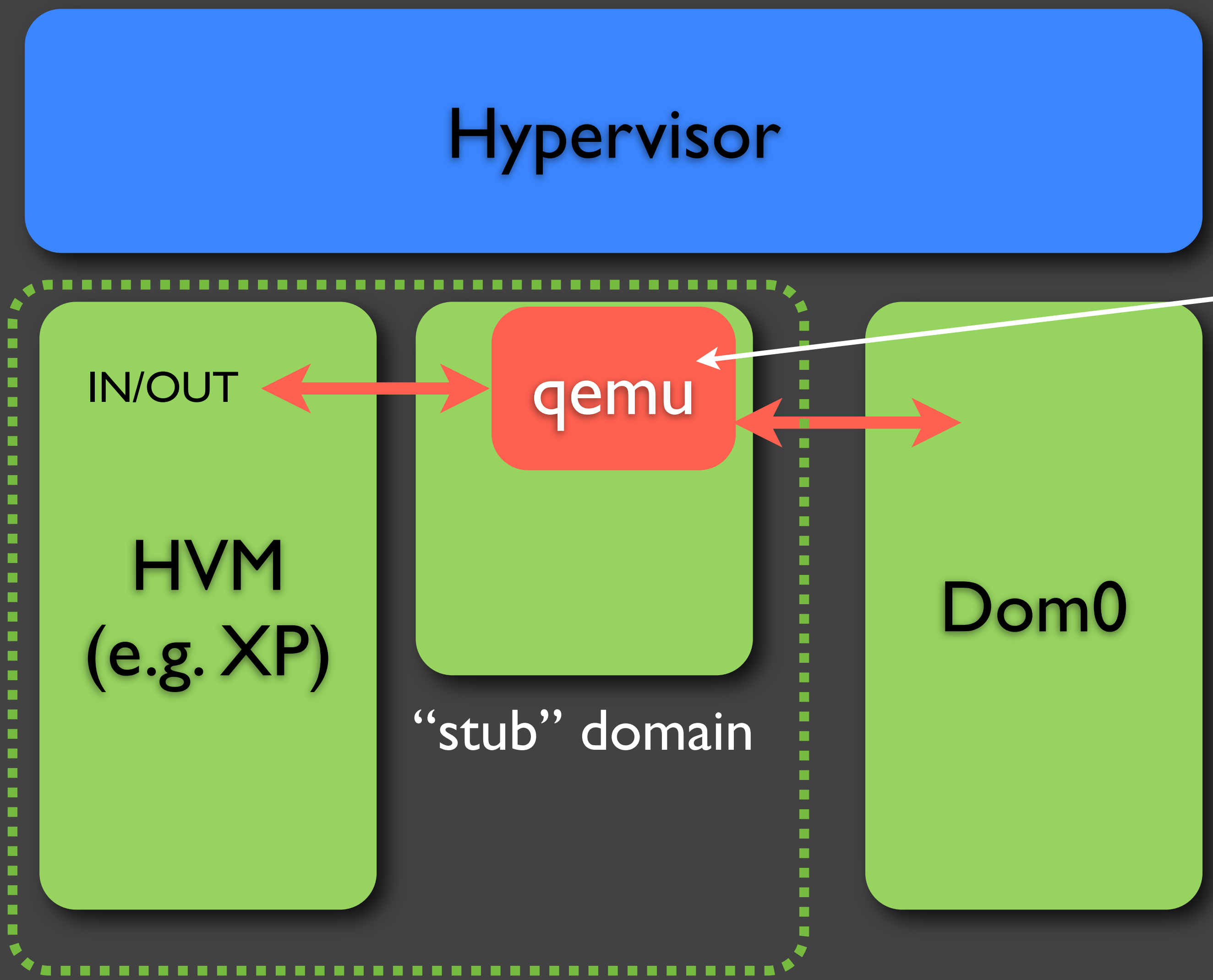
IOMMU/VT-d needed for delegating drivers to other domains (otherwise we can use DMA attacks from DomU)

Advantage: compromise of a driver != Dom0 access

# Stub domains

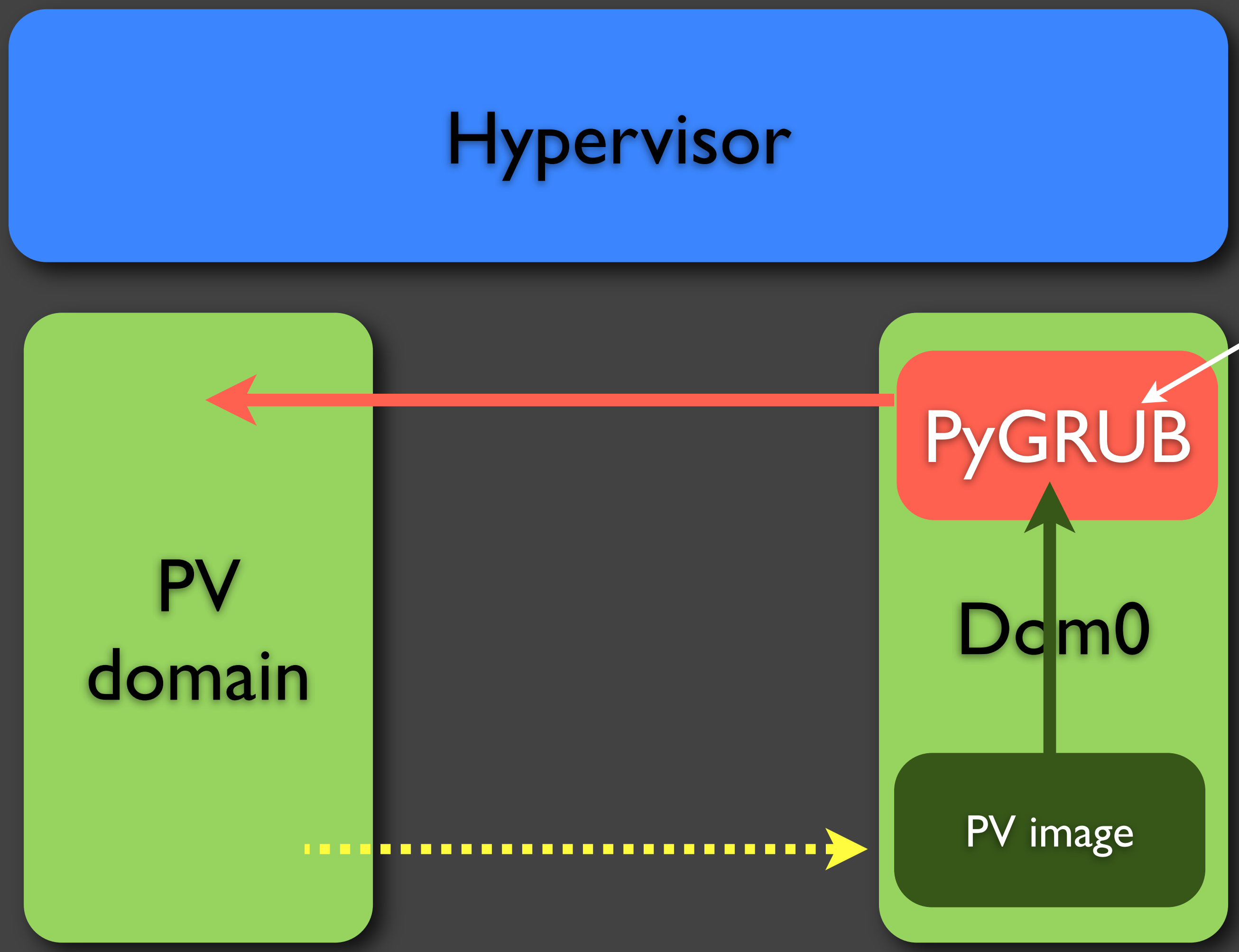



Usermode process that runs as root in Dom0 (Device Virtualization Model)

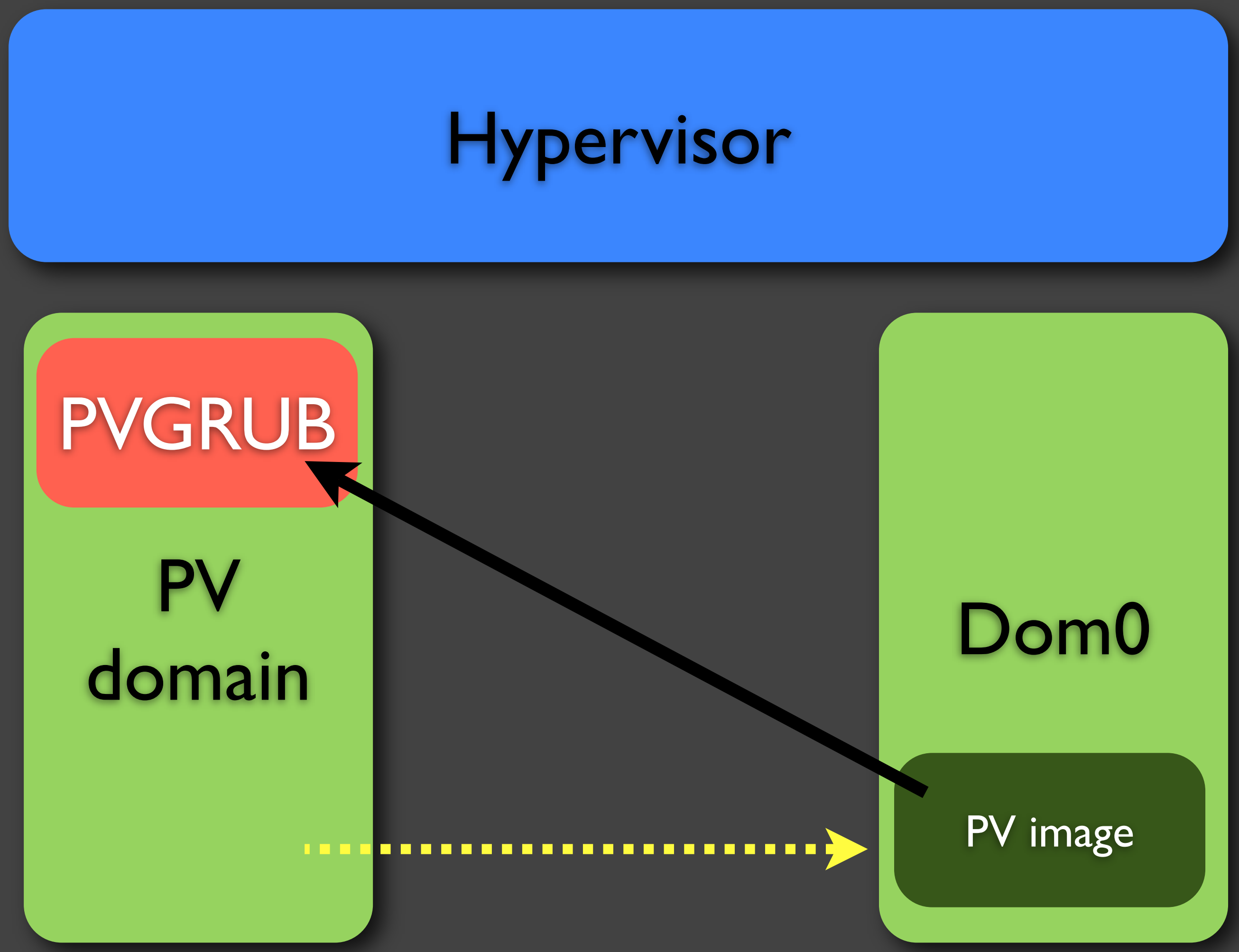



Now:
qemu compromises !=
Dom0 comrpomise

# PyGRUB vs. PVGRUB




Runs in Dom0 with root privileges and process the PV domain image (untrusted)


Hypervisor
PVGRUB
PV
domain
Dom0
PV image

Xen vs. competition?

| | Xen 3.3 | Hyper-V (**) | ESX |
|---|---|---|---|
| IOMMU/VT-d support? | **Yes** | **No** | **?** |
| Hypervisor protected from the Admin Domain (including DMA attacks)? | **Yes** | **No** | **?** |
| Driver domains? | **Yes** (drivers in unprivileged domain) | **No** (drivers in the root domain) | **No?** Drivers in the hypervisor?! (*) |
| I/O Emulator placement? (Device Virtualization) | **Unprivileged Domain** ("stub domains") | **Unprivileged process** (vmwp.exe running as NETWORK_SERVICE in the root domain) | **?** |
| Trusted Boot support? (DRTM/SRTM) | **Yes** Xen tboot: DRTM via Intel TXT | **No** | **?** |

(*) based on the VMWare's presentation by Oded Horovitz at CanSecWest, March 2008 (slide #3)
(**) based on the information provided by Brandon Baker (Microsoft) via email, July 2008

Ok, so does it really work?

Yes! No doubt it's a way to go!

Xen is well done!

but...

Overflows in hypervisor :o

So far, not a single overflow in Xen 3 hypervisor found!

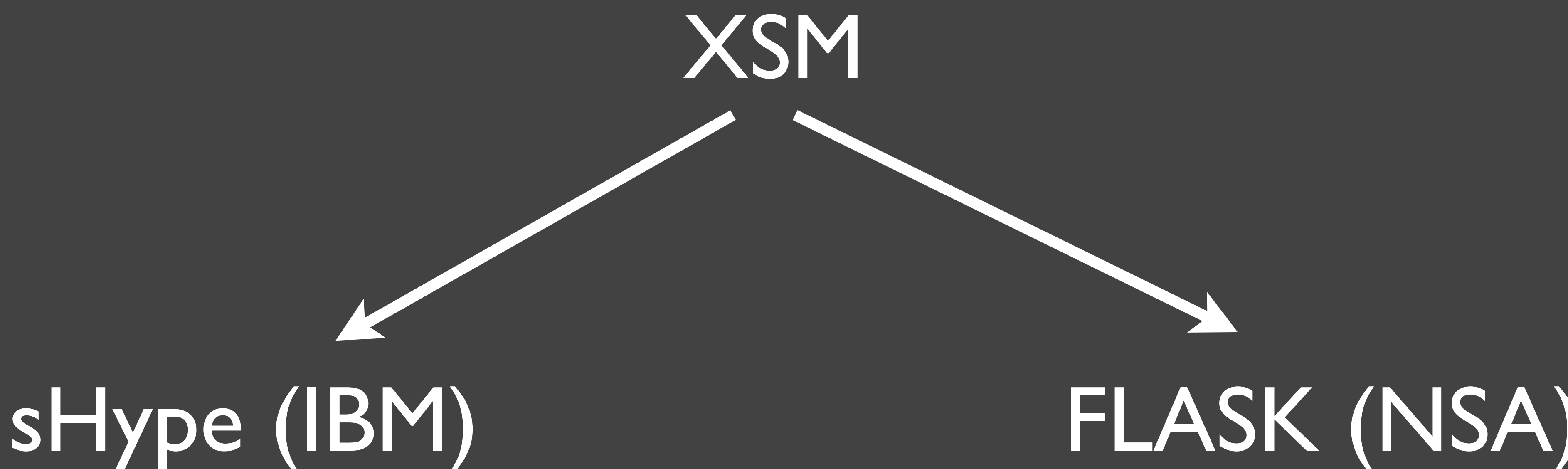
... until Rafal looked at it :)

# The FLASK bug

# What is FLASK?

# FLASK

- One of the implementation of XSM
- XSM = Xen Security Modules
- XSM is supposed to fine grain control over security decisions
- XSM based on LSM (Linux Security Modules)


XSM
sHype (IBM)
FLASK (NSA)

# FLASK is not compiled in by default into XEN

```
# Enable XSM security module.  Enabling XSM requires selection of an
# XSM security module (FLASK_ENABLE or ACM_SECURITY).
XSM_ENABLE ?= n
FLASK_ENABLE ?= n
ACM_SECURITY ?= n
```

Ok, so where are the bugs?

```
static int flask_security_user(char *buf, int size)
{
    char *page = NULL;
    char *con, *user, *ptr;
    u32 sid, *sids;
    int length;
    char *newcon;
    int i, rc;
    u32 len, nsids;

    length = domain_has_security(current->domain, SECURITY__COMPUTE_USER);
    if ( length )
        return length;

    length = -ENOMEM;
    con = xmalloc_array(char, size+1);
    if ( !con )
        return length;
    memset(con, 0, size+1);

    user = xmalloc_array(char, size+1);
    if ( !user )
        goto out;
    memset(user, 0, size+1);

    length = -ENOMEM;
    page = xmalloc_bytes(PAGE_SIZE);
    if ( !page )
        goto out2;
    memset(page, 0, PAGE_SIZE);

    length = -EFAULT;
    if ( copy_from_user(page, buf, size) )
        goto out2;

    length = -EINVAL;
    if ( sscanf(page, "%s %s", con, user) != 2 )
        goto out2;

    length = security_context_to_sid(con, strlen(con)+1, &sid);
    if ( length < 0 )
        goto out2;
```

Passed as hypercall arguments

page buffer is always 4096 bytes big!

```c
static int flask_security_relabel(char *buf, int size)
{
    char *scon, *tcon;
    u32 ssid, tsid, newsid;
    u16 tclass;
    int length;
    char *newcon;
    u32 len;

    length = domain_has_security(current->domain, SECURITY__COMPUTE_RELABEL);
    if ( length )
        return length;

    length = -ENOMEM;
    scon = xmalloc_array(char, size+1);
    if ( !scon )
        return length;
    memset(scon, 0, size+1);

    tcon = xmalloc_array(char, size+1);
    if ( !tcon )
        goto out;
    memset(tcon, 0, size+1);

    length = -EINVAL;
    if ( sscanf(buf, "%s %s %hu", scon, tcon, &tclass) != 3 )
        goto out2;

    length = security_context_to_sid(scon, strlen(scon)+1, &ssid);
    if ( length < 0 )
        goto out2;
    length = security_context_to_sid(tcon, strlen(tcon)+1, &tsid);
    if ( length < 0 )
        goto out2;

    length = security_change_sid(ssid, tsid, tclass, &newsid);
    if ( length < 0 )
        goto out2;

    length = security_sid_to_context(newsid, &newcon, &len);
    if ( length < 0 )
        goto out2;
```

Passed as hypercall arguments

Yes this is sscanf()! Welcome back 90's!

So, how do we exploit it?

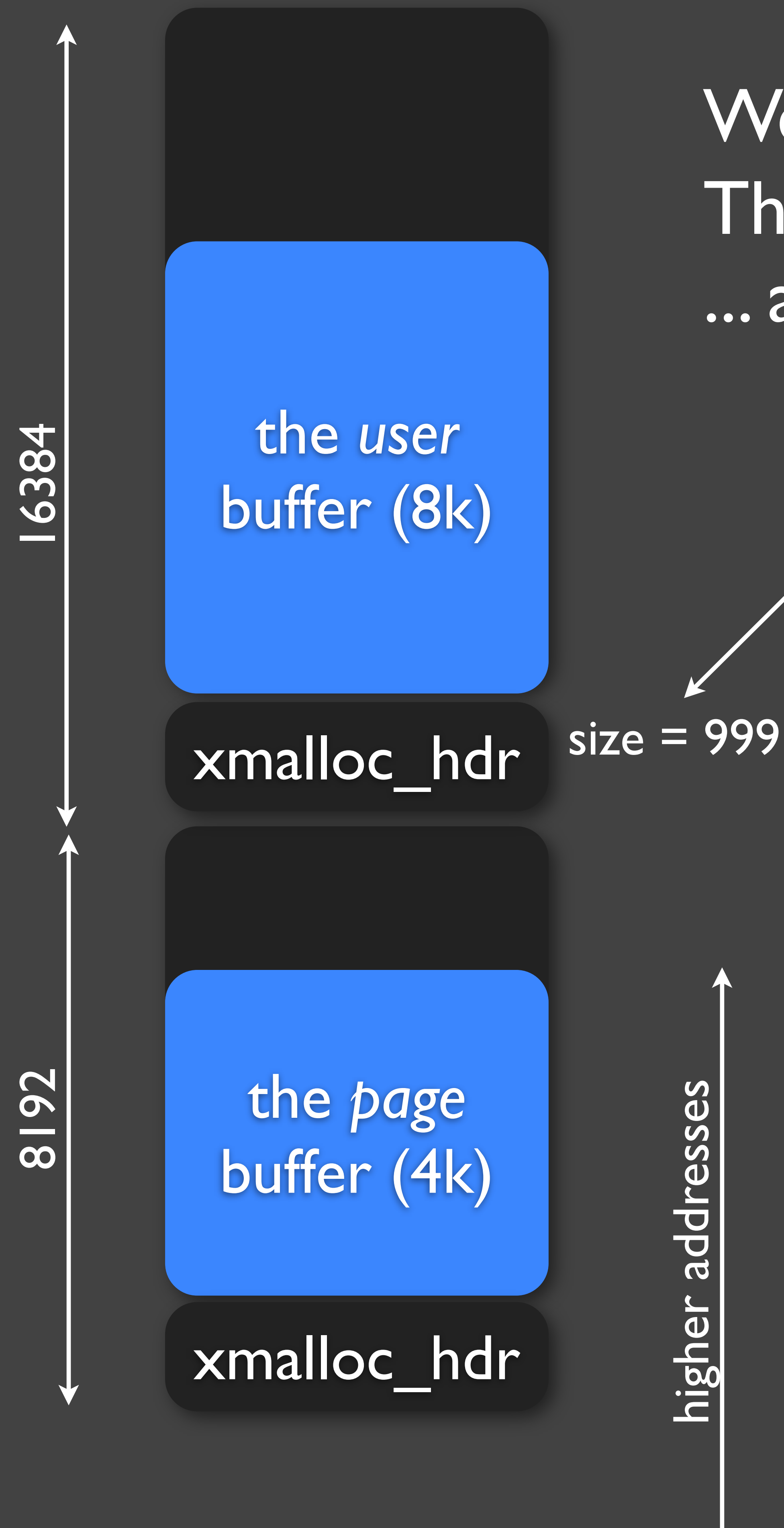
```
struct xmalloc_hdr
{
    size_t size;
    struct list_head freelist;
} __cacheline_aligned;


struct list_head {
    struct list_head *next, *prev;
};
```

Step 1: flask_user (buf, 8192)

We set: **`buf[8192-hdr_sz]=999`**

Then *buf* overwrites *page*...

... and *user* hdr's size field gets a new value!



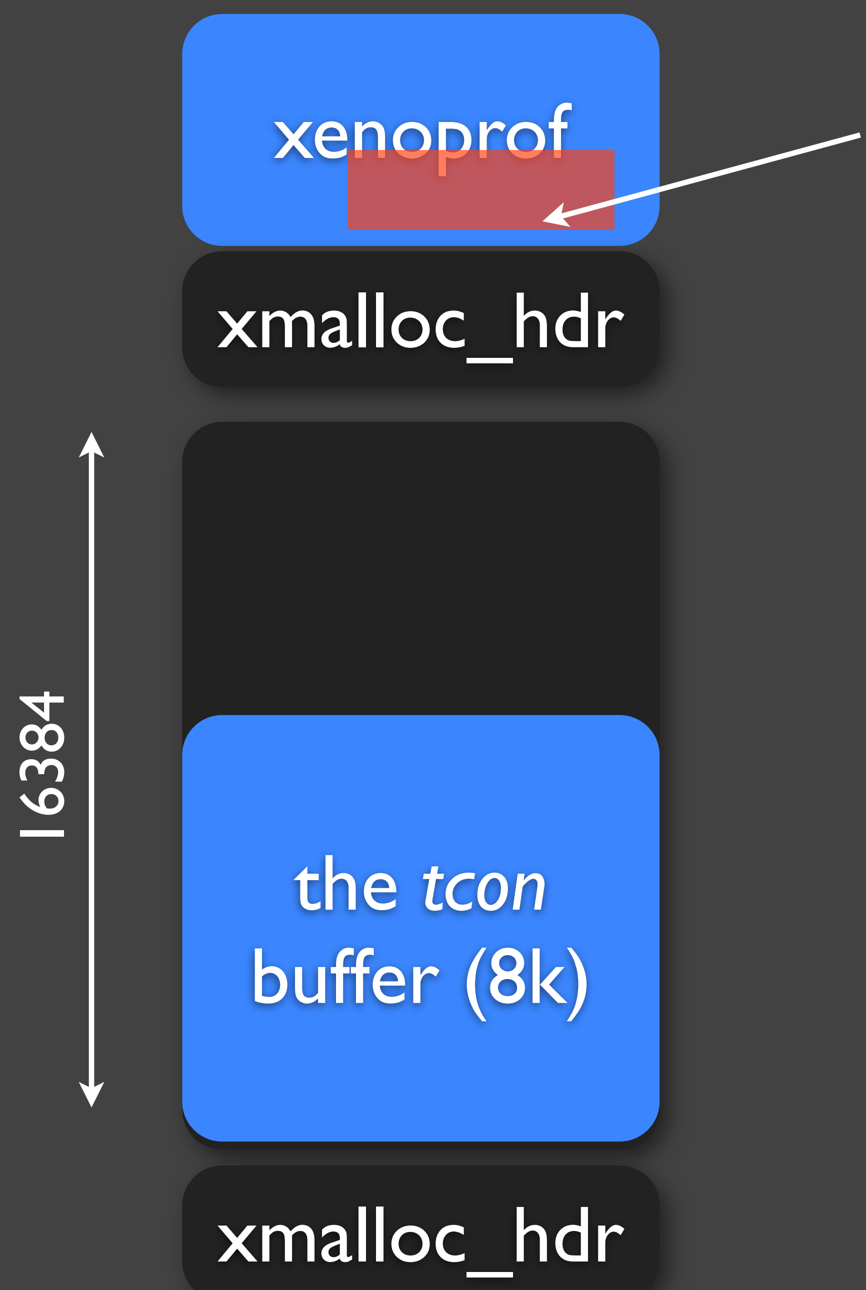

- After freeing buf xmalloc will put it on a list of small free chunks and use for the next allocation of a small chunk!
- '999' is a cosmic constant that satisfies the requirement:
  sizeof (struct xenoprof) < 999 < 4096
- "Small" chunks: chunks for buffers that are less then 4096 bytes

Step 2: flask_relabel (buf, 8192)

some pointers that are later nullified by xenoprof_reset_buf()...

... so if we write *addr* there, then...

we will get `(long)0` written at *addr* :)



# Step 3: freeing xenoprof buffer

xenoprof_enable_virq();

What we got?

A write-zero-to-arbitrary-address primitive

What to overwrite with zero?
How about the upper half of some hypercall address?
This way we will redirect it to usermode!

# Demo: Escape from DomU using the FLASK bug

```
[root@dom0 ~]#
```

```
{
        unsigned int csval = 0;
        asm("movl %%cs, %0\n":"=r"(csval));
        xen_printk("Hello from domain %d, code running with cs=%x\n",
                   current->domain->domain_id, csval);
        xen_printk("All your hypervisor are belong to us !\n");
        return 0xaabbccdd;
}

[root@some_domU flask-bo]#
```

The bug has been patched on July 21st, 2008:

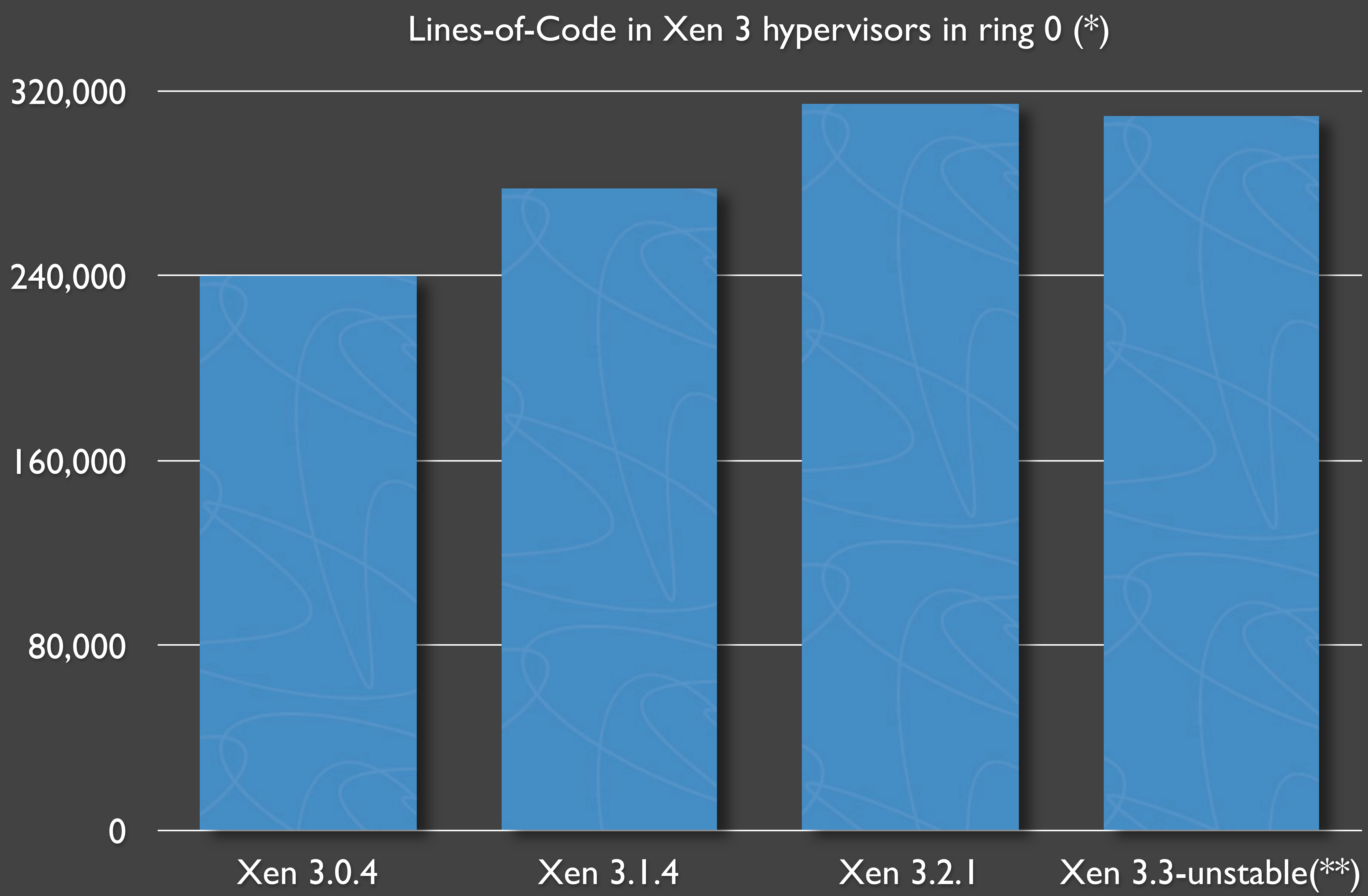
```
changeset:   18096:fa66b33f975a
user:        Keir Fraser <keir.fraser@citrix.com>
date:        Mon Jul 21 09:41:36 2008 +0100
summary:     [XSM][FLASK] Argument handling bugs in XSM:FLASK
```

BTW, note the lack of the “security” word in the patch description ;)


No Planet
Owning!

Can we get rid of all bugs in the hypervisor?

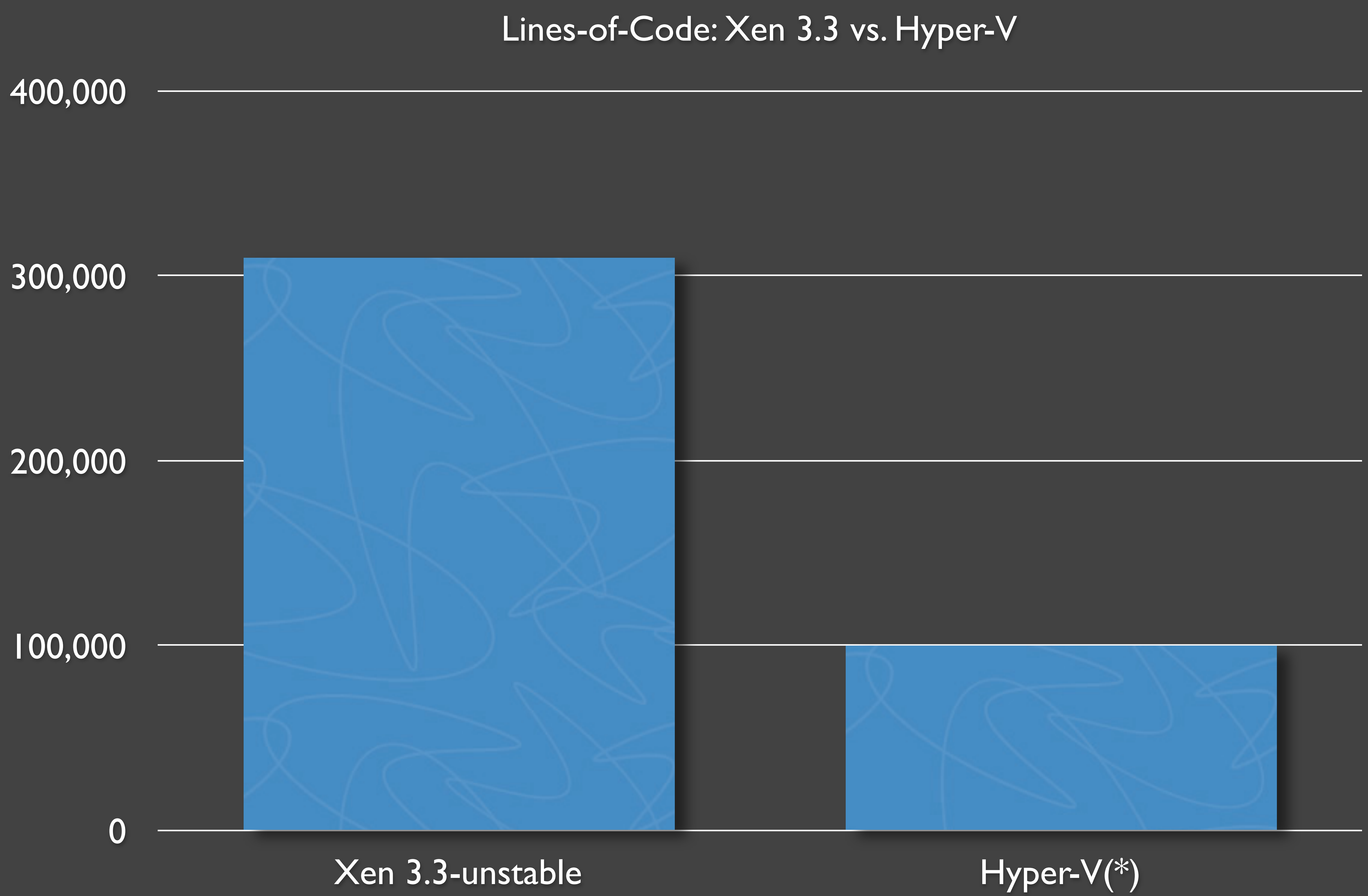
# Xen hypervisor complexity




*Calculated using: `find xen/ -name "*.[chsS]" -print0 | xargs -0 cat | wc -l`

**Retrieved from the Xen unstable mercurial on July 24th, 2008

Trend a bit disturbing...
Xen hypervisor grows over time,
instead of shrinking :(




(*) based on the information provided by Brandon Baker (Microsoft) via email, July 2008

# Lessons learnt

- Hypervisors are not special!
- Hypervisor can be compromised too!
- Computer systems are complex!
- *Prevention* is not enough!

Prevention not enough!

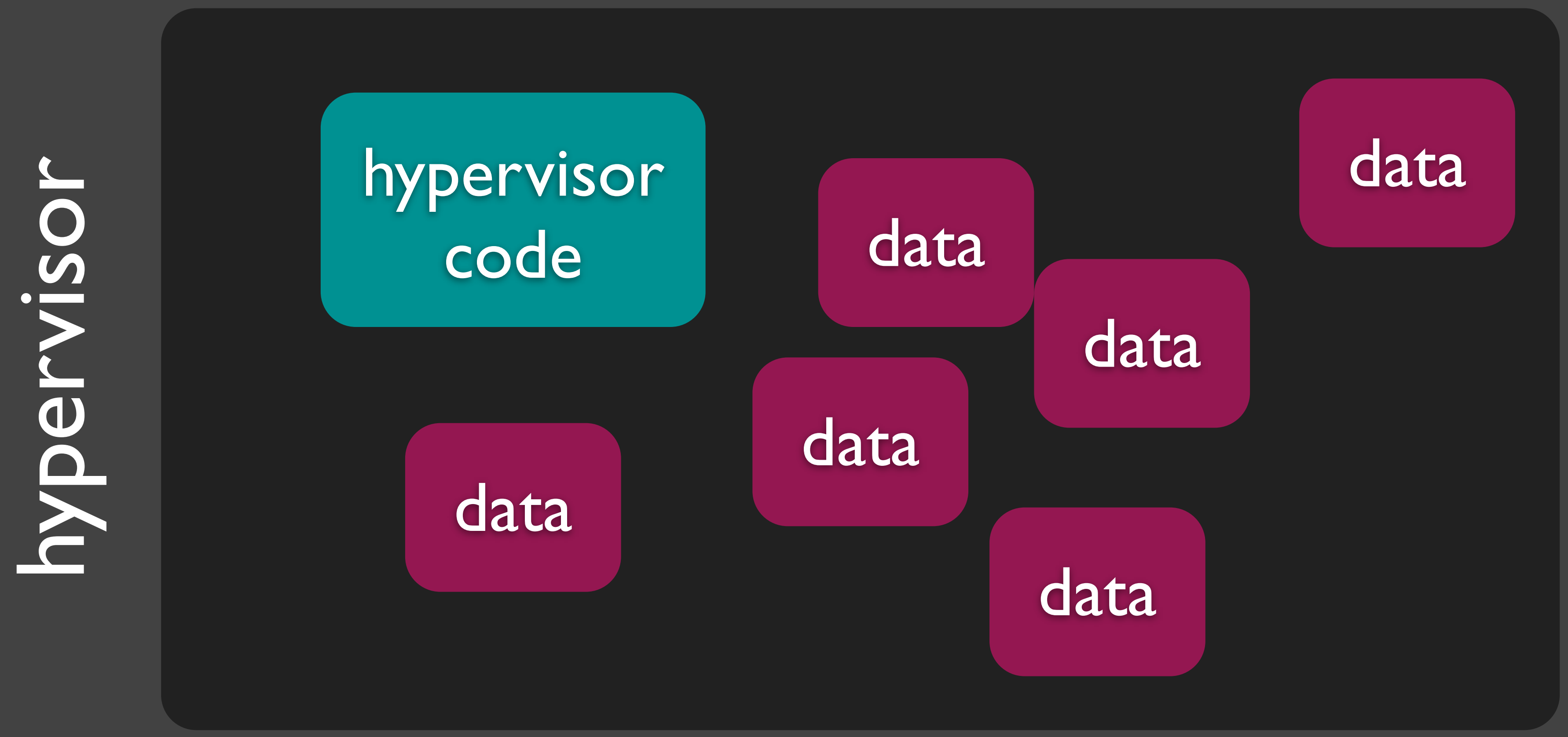
# Ensuring Hypervisor Integrity

# Integrity Scanning

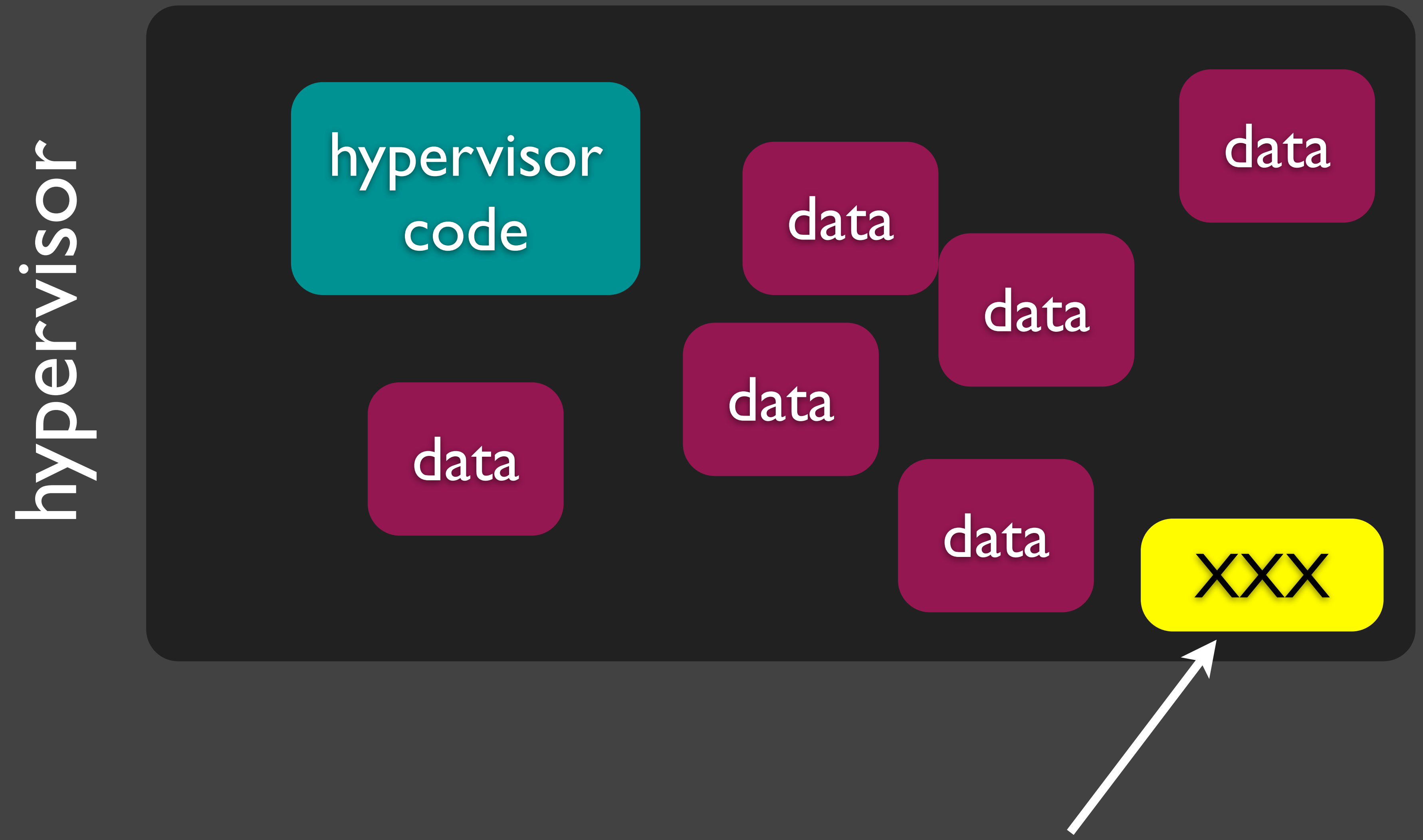
Integrity Scanning

Ensure the hypervisor's code & data are intact

Ensure no untrusted code in hypervisor




Code is easy to verify...
... but data is not!




*Executable* page with *untrusted* code

# Ensuring no untrusted code in the hypervisor

1. Read hypervisor's CR3
2. Parse Page Tables and find all pages that are marked as *executable* and *supervisor* in their PTEs
3. Verify the hashes of those code pages remain the same as during the initialization phase
4. Also: ensure some system wide registers were not modified (CR4, CR0, etc)

To make it work...

- Hypervisor must strictly apply the NX bit (only code pages do not have NX bit set)
- No self-modifying code in the hypervisor
- Hypervisor’s code not pageable

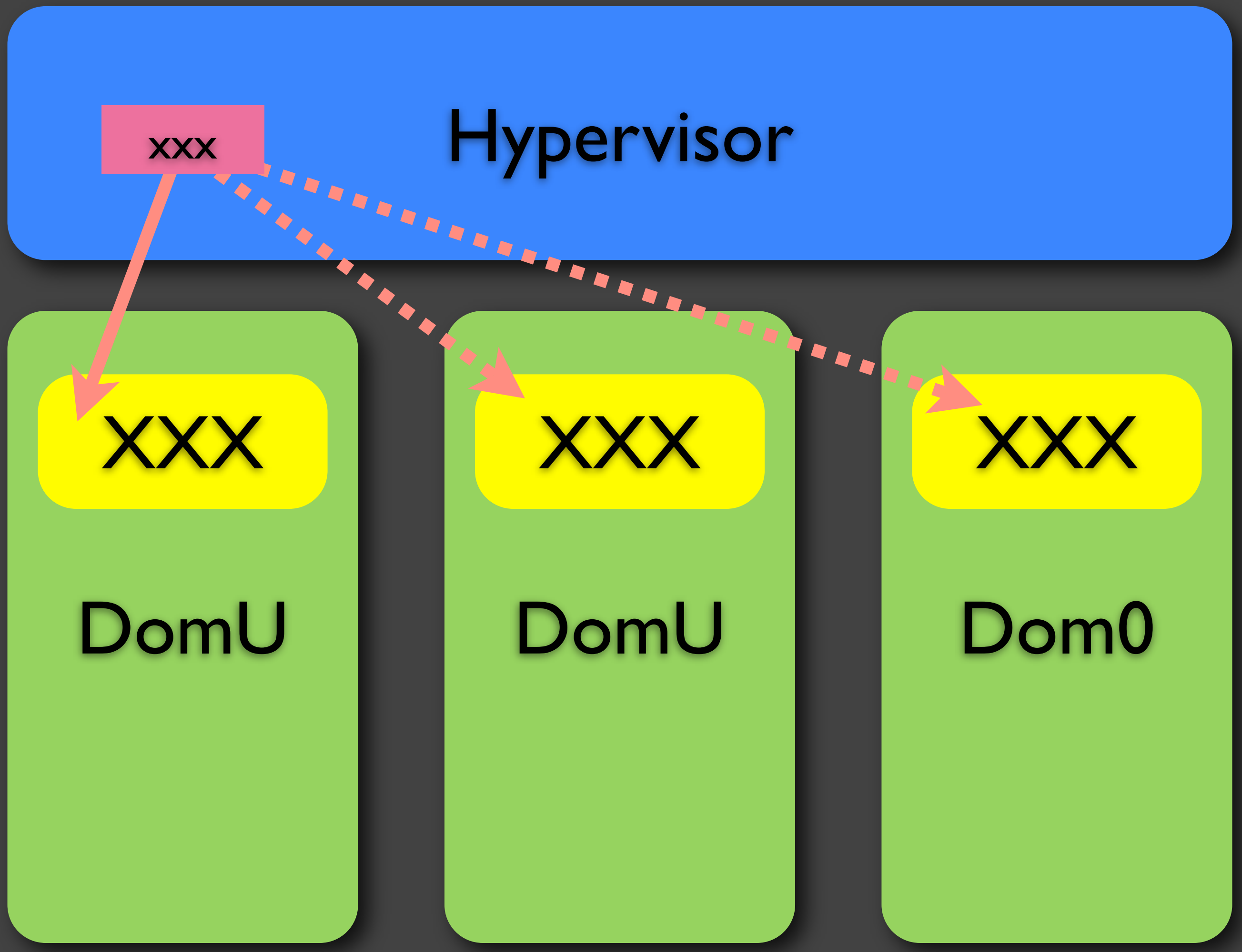
Xen hypervisor can meet those requirements with just few cosmetic workarounds

Hyper-V already meets all those requirements!
(Brandon Baker, Microsoft)

… but, there are traps!

# Trap #1

Rootkit might keep its code in the usermode pages - CPU would still execute them from ring0...



CPU should refuse to execute code from usermode pages when running in ring0

Marketing name: "NX+" or "XD+" :)

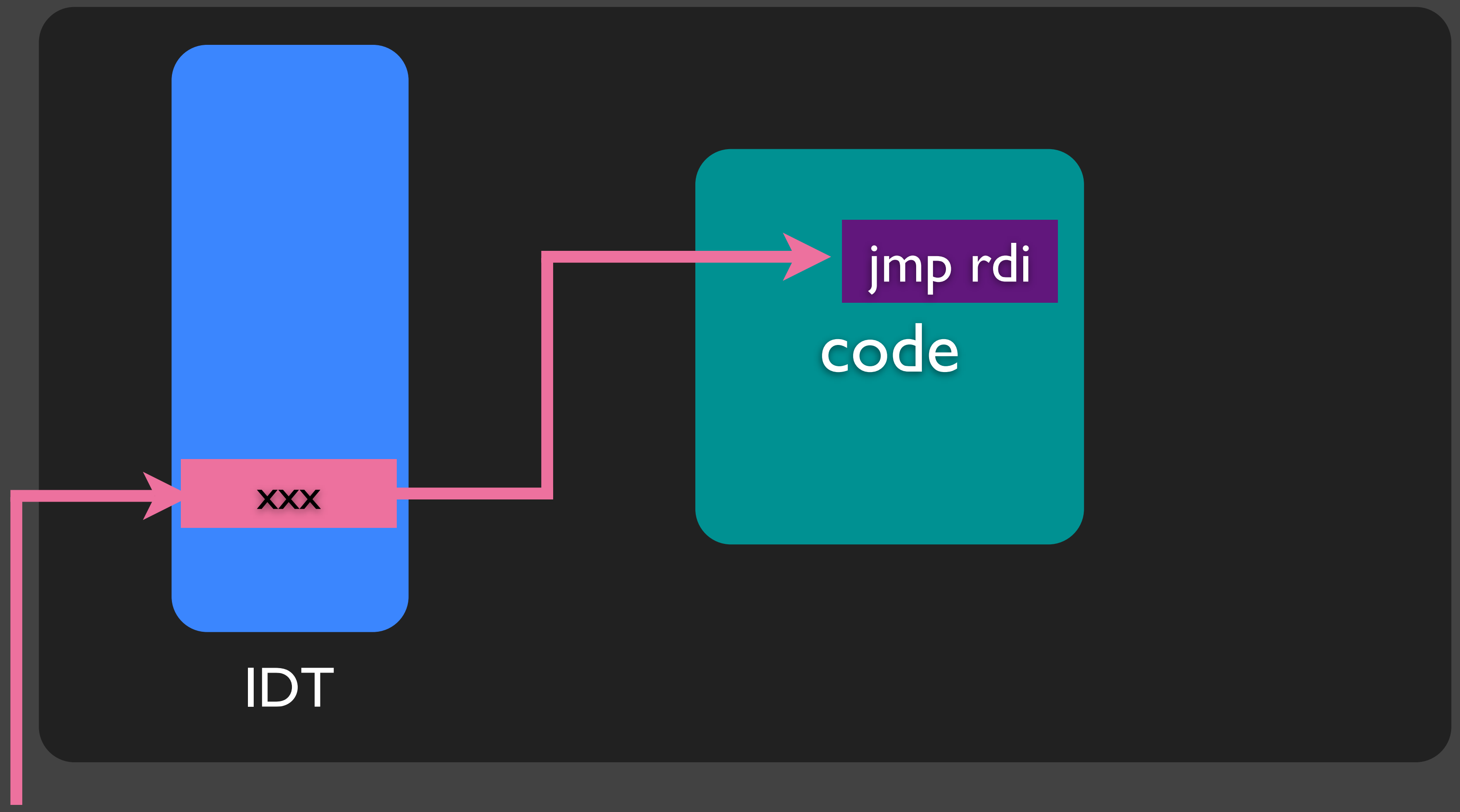
Talks with Intel in progress...

# Trap #2

## Code-less backdoors!

## ‘jmp rdi’ or more advanced ret-into-libc stuff

## (don’t think ret-into-libc not possible on x64!)




Anybody who can issue INT XX
can now get their code executed
in ring0 in the hypervisor!

There only *few* structures (function pointers) that could be used to plant such backdoor!

This is *few* comparing with *lots of* if we were to check all possible function pointers

Examples for Xen: IDT, hypercall_table, exception_table

Hypervisor should provide a sanity function that would be part of the code (static path) that would check those *few* structures.

HyperGuard doesn't need to know about those *few* structures.

Trap #3

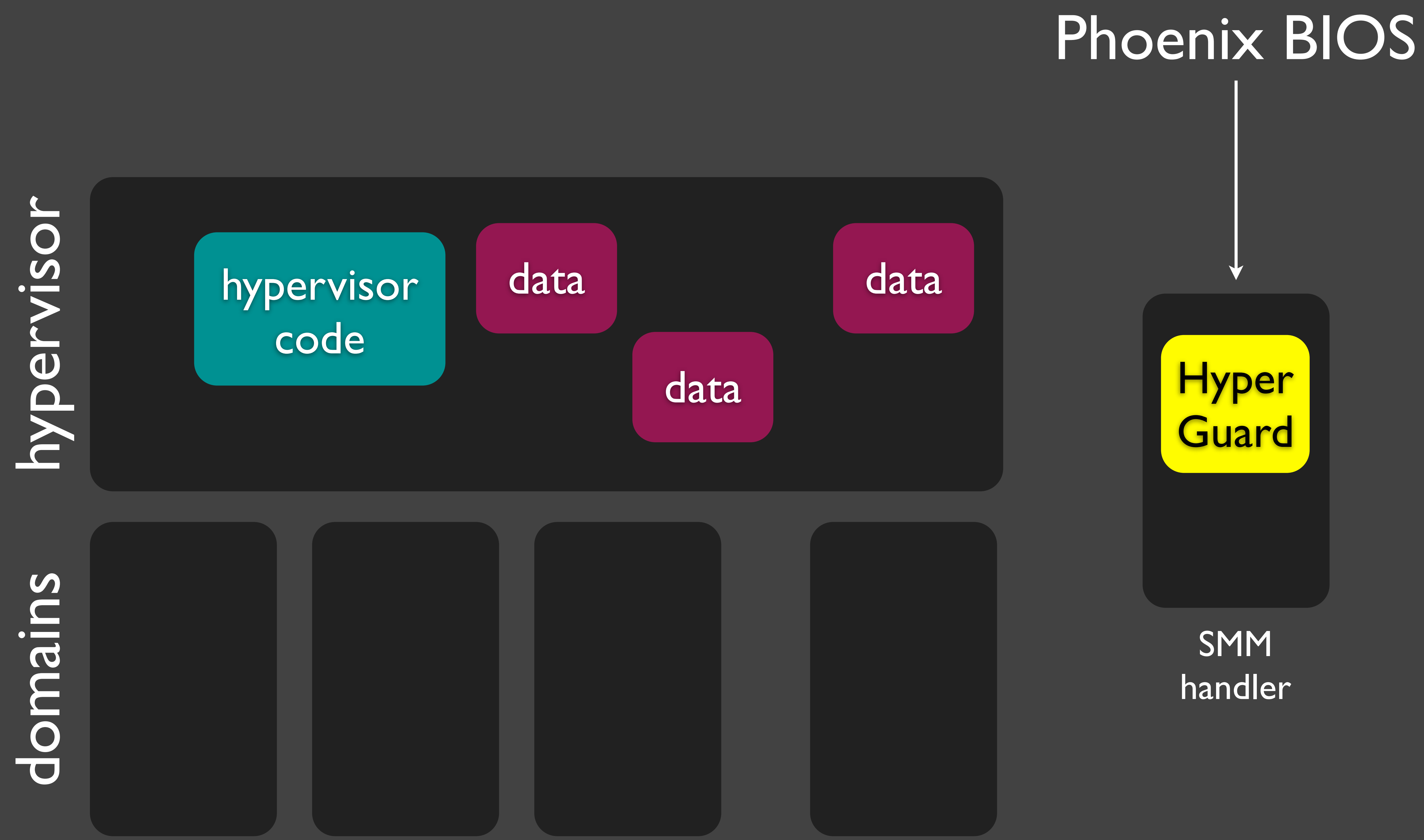
We only check integrity at the very moment...
when we check integrity...
What happens in between?
When should we do the checks?

Solution?

Oh, come on, we need to leave a few aces up in our sleeves ;)

Introducing HyperGuard...

HyperGuard is a project done in cooperation with Phoenix Technologies



Phoenix BIOS
hypervisor
hypervisor code
data
data
data
domains
Hyper Guard
SMM handler

# Why in SMM?

| | SMM handler | PCI device | Chipset |
|---|---|---|---|
| tamper proof? | should be :) (depends very much on the BIOS -- see the Q35 bug) | yes | yes |
| access to CPU state (e.g. registers) | yes | no | no |
| reliable access to DRAM | yes | no (e.g. IOMMU, other redirecting tricks) | yes (can deal with IOMMU) |

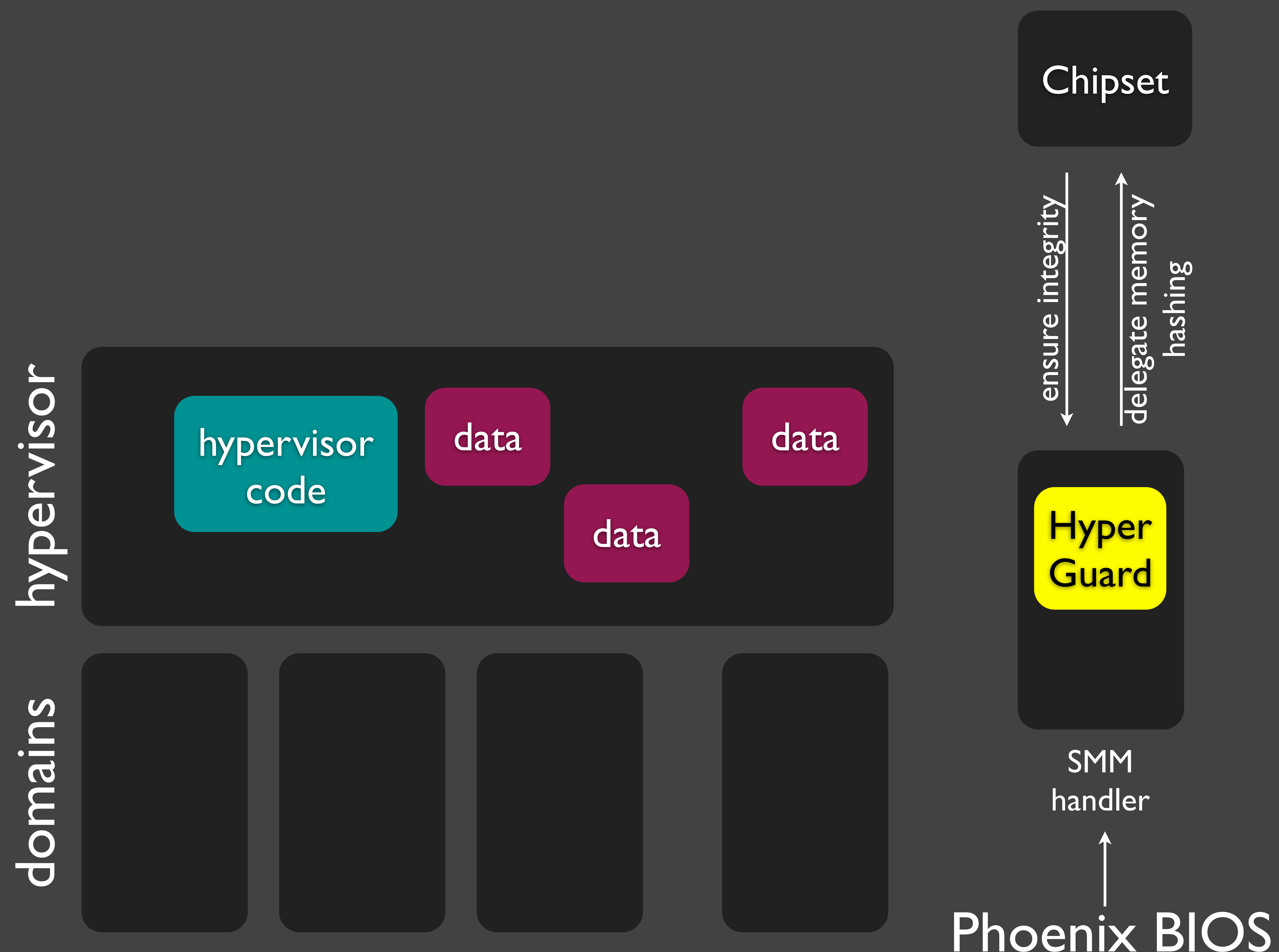
Combining chipset-based scanner (see Yuriy Bulygin's presentation) with SMM-based scanner seems like a good mixture...

# CHIPSET BASED APPROACH TO DETECT VIRTUALIZATION MALWARE
# a.k.a. DeepWatch

Yuriy Bulygin

Joint work with David Samyde

Security Center of Excellence / PSIRT @ Intel Corporation


Chipset
ensure integrity
delegate memory hashing
hypervisor
hypervisor code
data
data
data
Hyper Guard
domains
SMM handler
Phoenix BIOS

# Combining SMM + chipset integrity scanning

| | SMM handler | PCI device | Chipset | SMM + chipset |
|---|---|---|---|---|
| tamper proof? | should be :) | yes | yes | **yes** |
| access to CPU state (e.g. registers) | yes | no | no | **yes** |
| reliable access to DRAM | yes | no | yes | **yes** |

Additionally chipset could provide fast hash calculation service to the HyperGuard

But we should keep the chipset based scanner as simple as possible!

The deeper we are the simpler we are!

Talks with Intel in progress...

HyperGuard might also be used in the future to verify integrity of normal OS kernels (e.g. Windows or Linux)

Slides available at:
http://invisiblethingslab.com/bh08

Demos and code will be available from the same address after Intel releases the patch.

# Credits

- Brandon Baker (Microsoft), for providing lots of information about Hyper-V (that we haven’t played with ourselves yet)

# Thank you!

Xen 0wning Trilogy to be continued in:

**“Bluepilling The Xen Hypervisor”**

by Invisible Things Lab