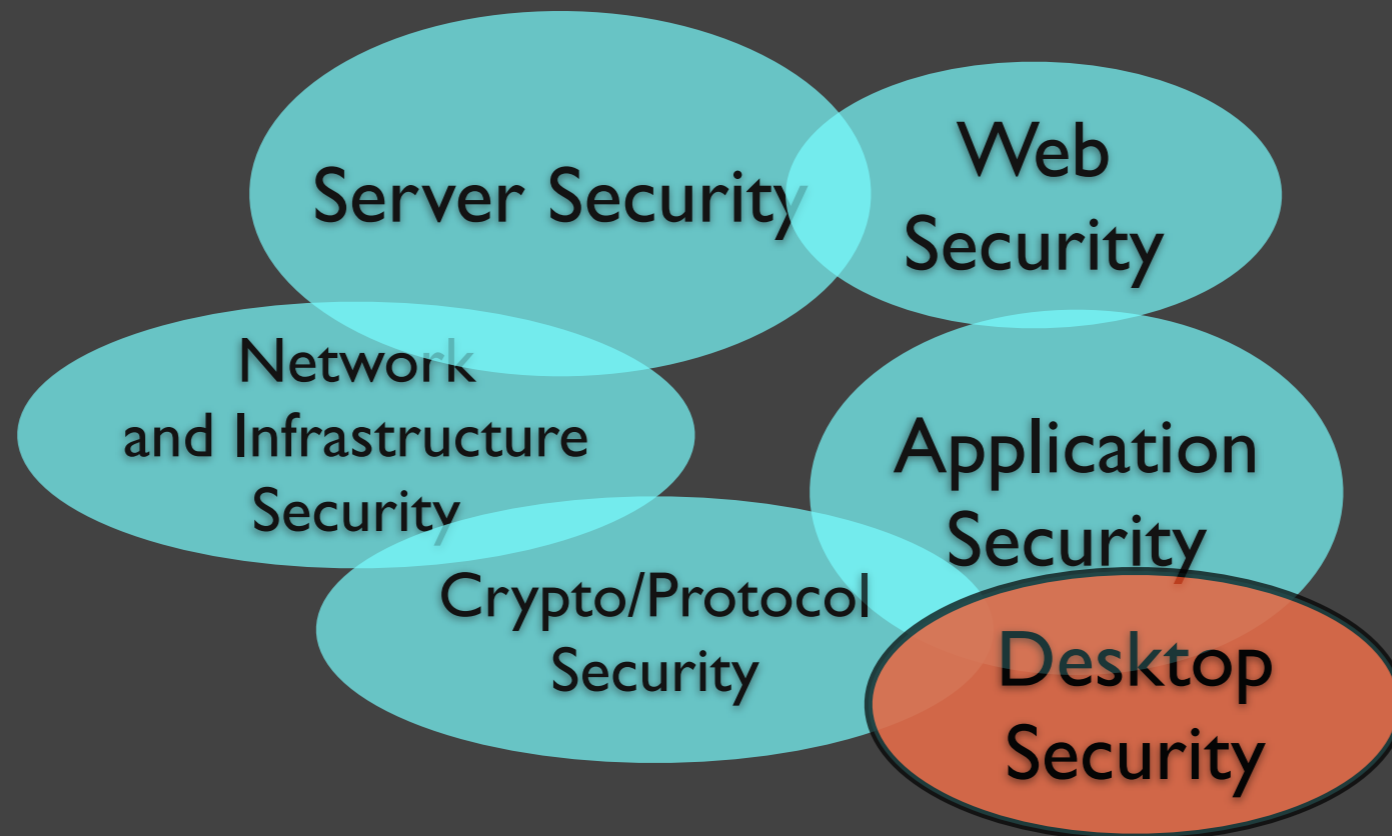


On Trusted Computing, Desktop Security, and Why This All Matters?

Joanna Rutkowska

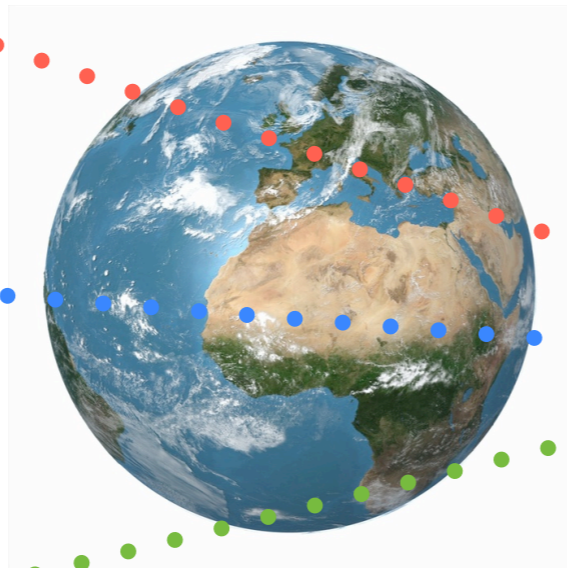






Super-secure servers!

Encrypted traffic!



All your **data** finally makes it here...



...where malware can easily steal it all!

Insecure desktops make all other security mechanisms useless!

Why desktop security is hard?

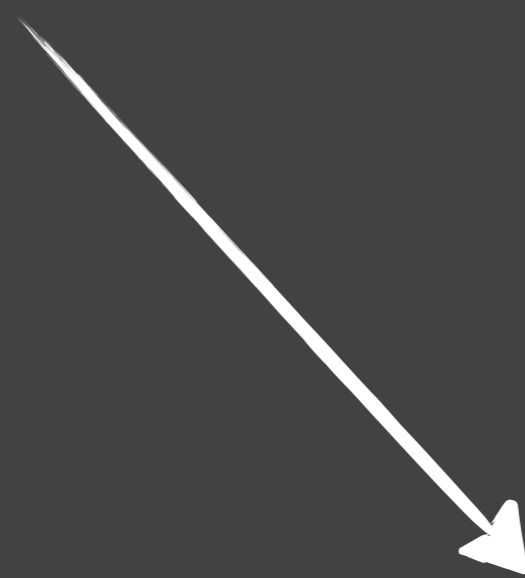
- **Lots of apps** installed by users (some buggy, some malicious)
- Lots of **different hardware** ⇒ Lots of (buggy) drivers
- Drivers in kernel ⇒ **Fat kernels** ⇒ Kernel exploits
- Fancy GUI ⇒ **complex GUI** subsystem ⇒ GUI 0wning exploits
- Users are dump... (“**human factor**”)

Two approaches to solve that...



Security by **Correctness**

- 👁 Patches, patches, patches...
- 👁 Bug hunting, fuzzing, developer education
- 👁 Safe languages, compiler tricks (e.g. stack canaries), NX memory



Security by **Isolation**

- 👁 User accounts, ACLs, chroot(), SELinux, etc
- 👁 Usermode/kernelmode separation, process address space separation

The ***Security by Correctness*** approach simply doesn't work...

Home > Security > Cybercrime and Hacking

News

Hacker busts IE8 on Windows 7 in minutes

Dutch researcher byp
browser

By Gregg Keizer

March 25, 2010 06:56 AM ET

Computerworld - Two
Pwn2Own hacking cor
Windows 7.

Home > Vulnerabilities >

April 9, 2010, 9:37AM

Serious New Java Flaw Affects All Current Versions of Windows

by Dennis Fisher

Share Recommend (14) Print E-mail

17 Comments



There is a serious vulnerability in Java that leaves users running any of the current versions of Windows open to simple Web-based attacks that could lead to a complete compromise of the affected system. Two separate researchers released information on the vulnerability on Friday, saying that it has been present in Java for years.



Zero Day

Ryan Naraine and Dancho Danchev

Get Zero Day via: [Mobile](#) [RSS](#) [Email Alerts](#) Bios: [Ryan's Blo](#) [Dancho's Blo](#)

Pick a blog category view

March 24th, 2010

Pwn2Own hack topples Firefox on Windows

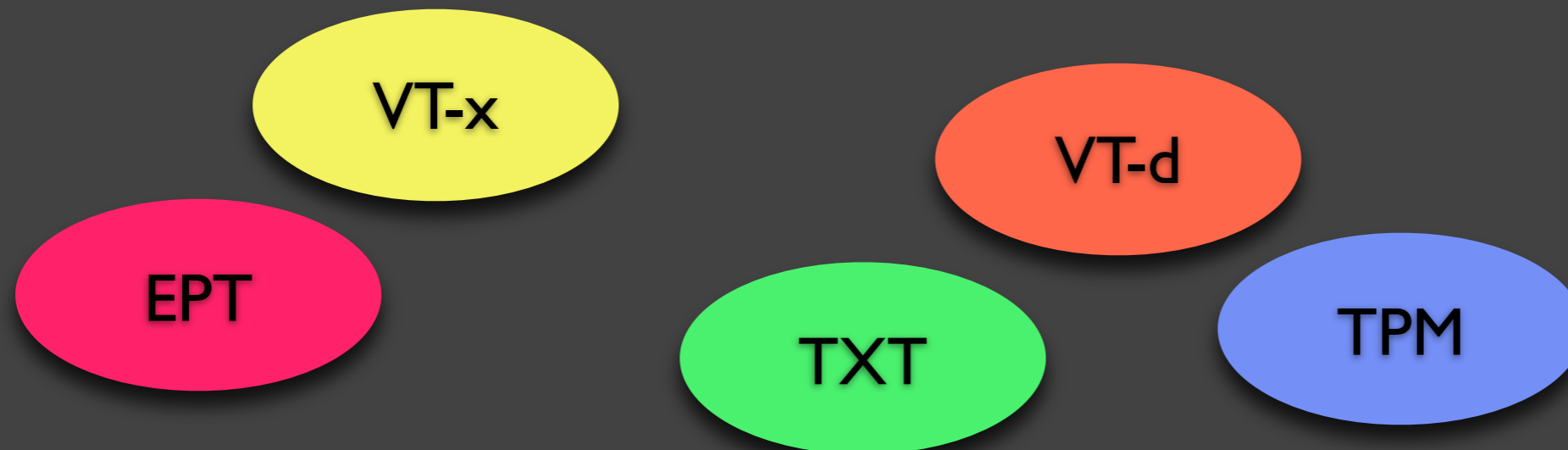
Posted by Ryan Naraine @ 6:27 pm

The ***Security by Isolation*** approach seems like the only option left for us...

But can we implement effective **Security by Isolation**
in **software** only?

- Too many drivers \implies too fat kernel \implies too **buggy kernel**
- Complex GPUs \implies complex and **buggy GUI** server
- Software virtualization \implies binary translation \implies complex instruction parsing/emulation

New **hardware** to the rescue!



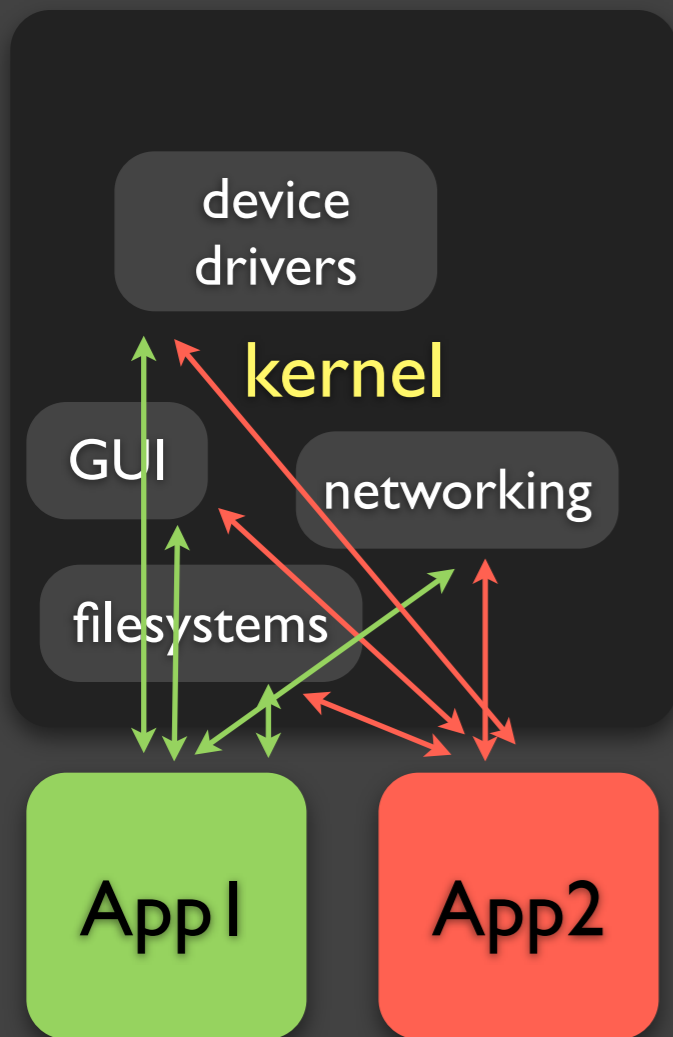
Just because something is in hardware, doesn't mean it's more secure...

See **Loic Duflot's** & co. work on attacking NICs, and also **ITL's** work on attacking chipsets.

...but, some things are just easier to do in hardware!

- **CPU virtualization:** VT-x/SVM *seems* easier than binary translation
- **Memory virtualization:** EPT/NPT *seems* easier than Shadow Page Tables
- **Device virtualization:** VT-d/IOMMU is *not possible* in software at all!
- **Trusted Boot:** DRTM (TXT) is *not possible* in software at all!

Did somebody just say: **virtualization**?



The **isolation** between **App1** and **App2**, in case of a traditional, monolithic-kernel-based OS, is kinda **poor**...

Improving isolation with virtualization..

Hypervisors, unlike standard kernel, do not need to provide all sorts of services like fs, GUI, networking, and the kitchen sink! So, the attack surface (VM-to-hypervisor) is muuuuuch smaller.



shared memory services

hypervisor

kernel

GUI
networking
filesystems

App1

kernel

GUI
networking
filesystems

App2

kernel

NIC
drivers
networking

Net Domain

kernel

storage
drivers
filesystems

Storage
Domain

kernel

graphics
drivers
GUI

GUI domain



Most of the usual services can be delegated to **deprivileged VMs!**

VT-x

VT-d

shared memory services hypervisor

kernel

GUI networking

filesystems

kernel

GUI networking

filesystems

kernel

NIC drivers

networking

kernel

storage drivers

filesystems

kernel

graphics drivers

GUI

App1

App2

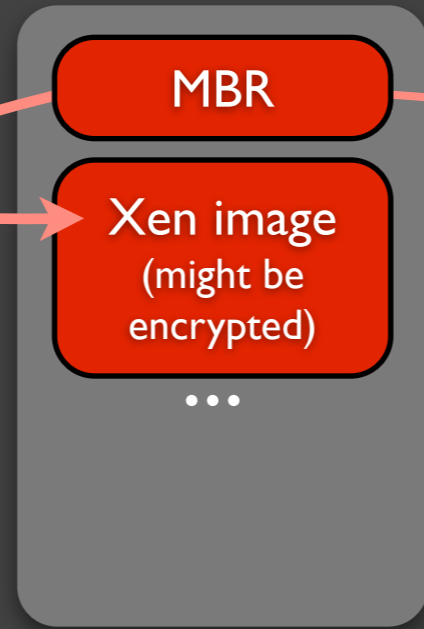
Net Domain

Storage Domain

GUI domain

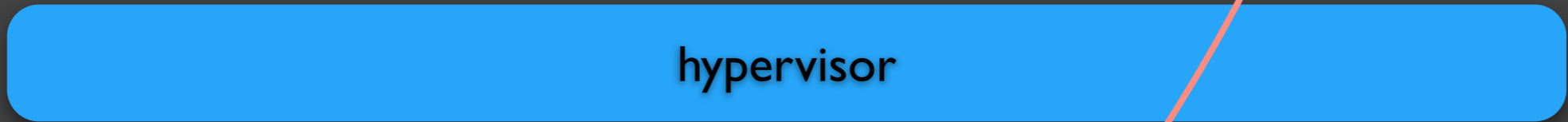
Ok, VT-x and VT-d might be useful... But what about **TXT** and **TPM**?

(2) ...compromise the hypervisor when it gets loaded



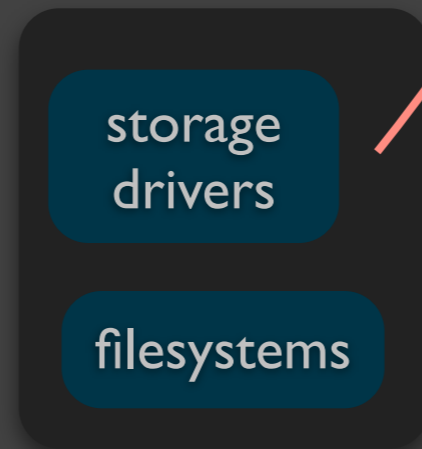
system disk

(1) Infect the MBR or the loader that decrypts the rest of the disk...



hypervisor

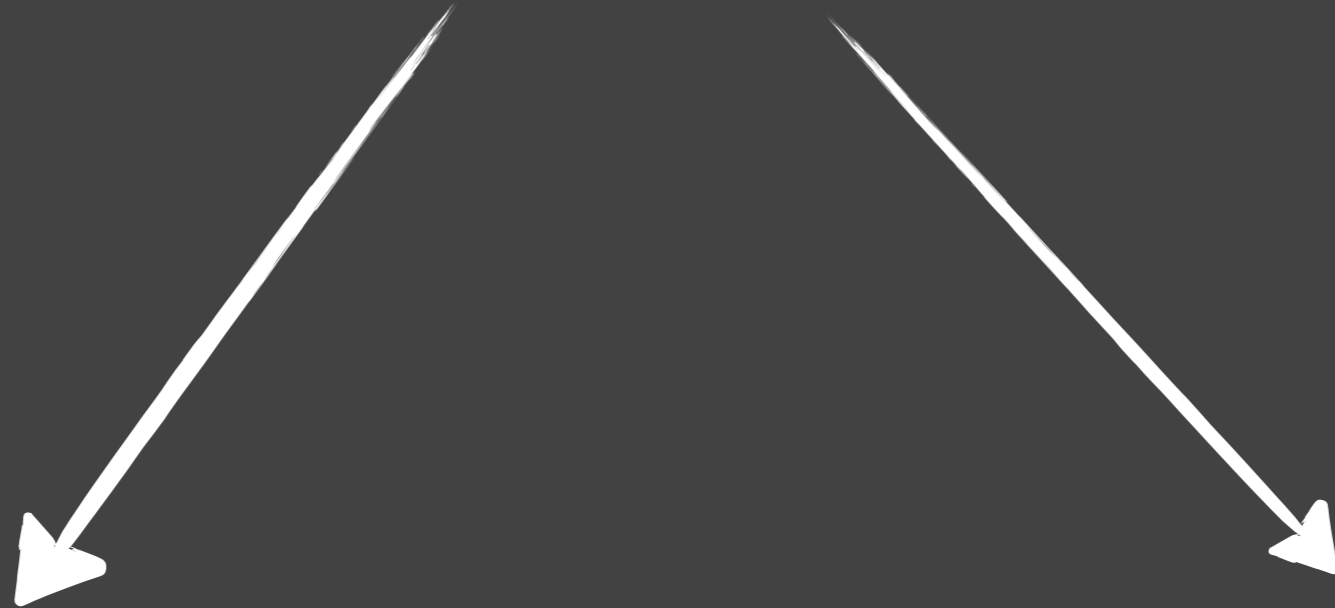
Without **Trusted Boot** we cannot have **unpriviliged storage domain** -- it could always compromise the system, because it has unlimited access to system storage (full disk encryption doesn't help -- there always must be some loader that is unencrypted)



Storage Domain (unpriviliged)

We need help from hardware (TPM) to implement **trusted boot**

Two approaches to trusted boot:



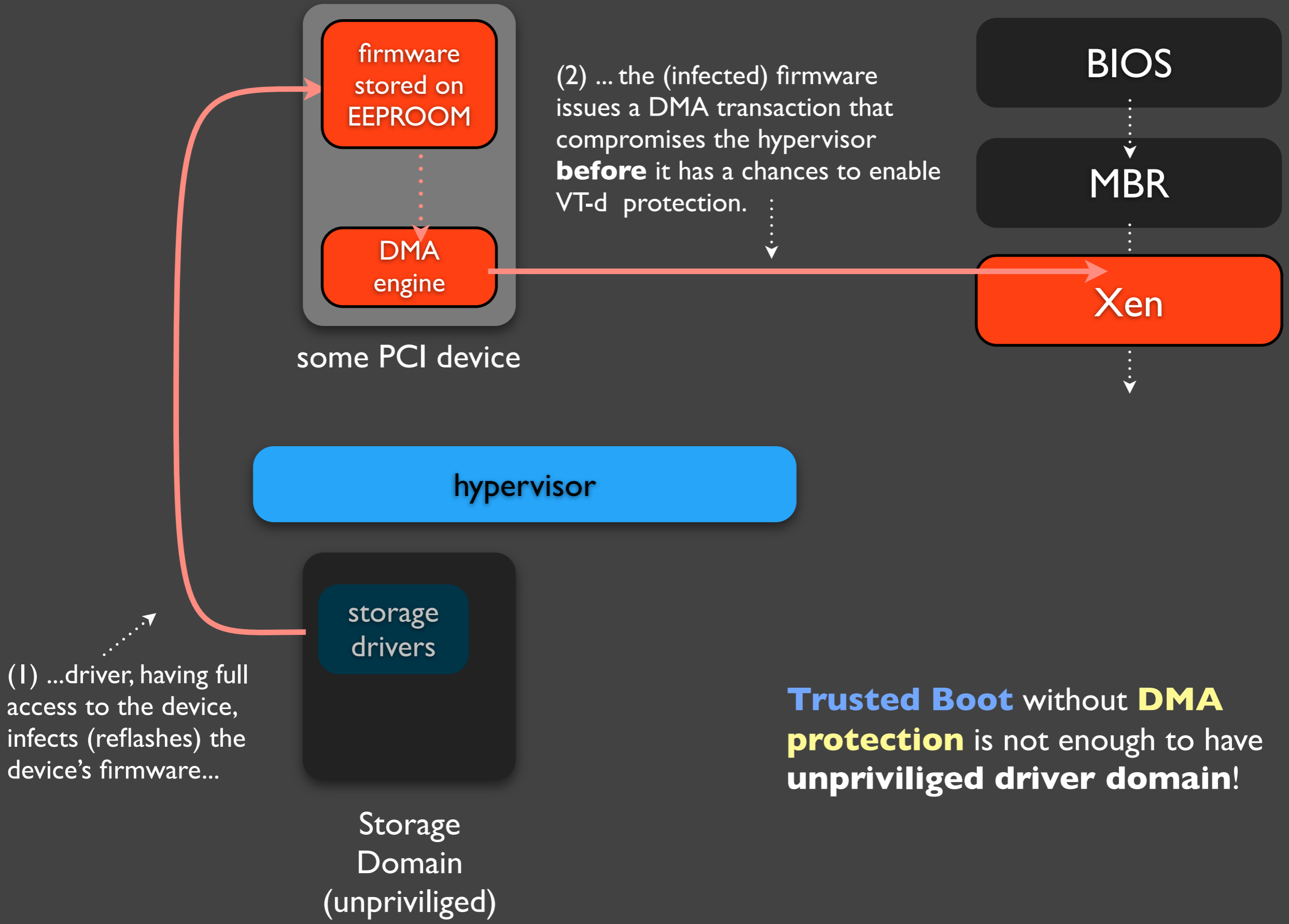
Static RTM

Requires: **TPM**

Dynamic RTM

Requires: **TPM** and Intel **TXT**
(apparently AMD has Presidio)

SRTM is straight forward, intuitive (sort of), and simple...
Why would anybody bother to use TXT instead?



(1) ...driver, having full access to the device, infects (reflashes) the device's firmware...

(2) ...the (infected) firmware issues a DMA transaction that compromises the hypervisor **before** it has a chance to enable VT-d protection.

Trusted Boot without **DMA protection** is not enough to have **unprivileged driver domain!**

Intel TXT beats SRTM because it offers **DMA protection**
(the protection is engaged before the hash of the MLE is calculated)

What else can I do with VT-d, TXT, and TPM?

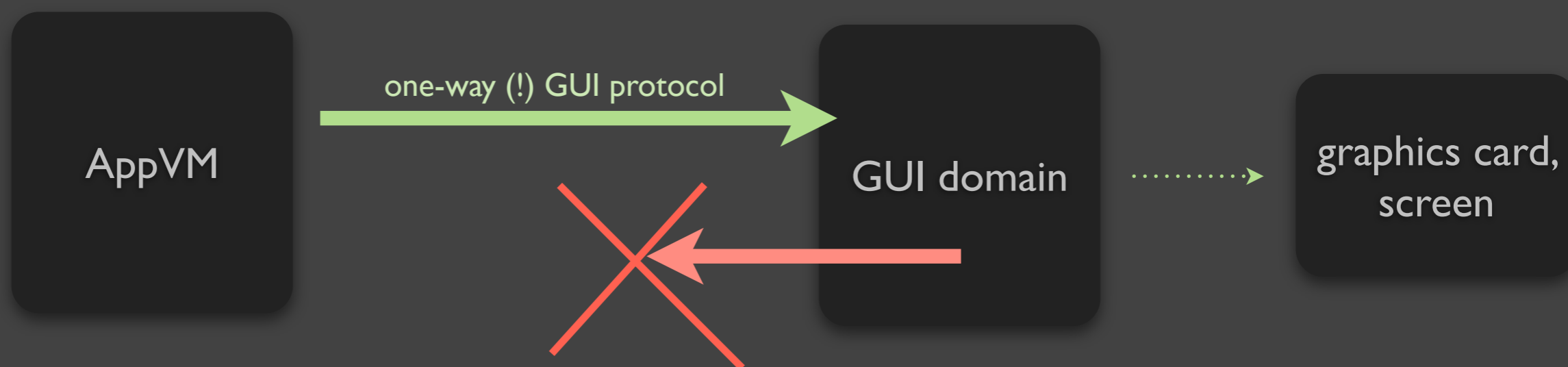
Introducing: **Unpriviliged GUI domain...**

(i.e. the compromise of the GUI domain should be harmless, except for potential DoS)

Wait a sec, GUI domain “sees” all the user’s **secrets unencrypted** (documents, emails, etc), how can we make it unprivileged then?

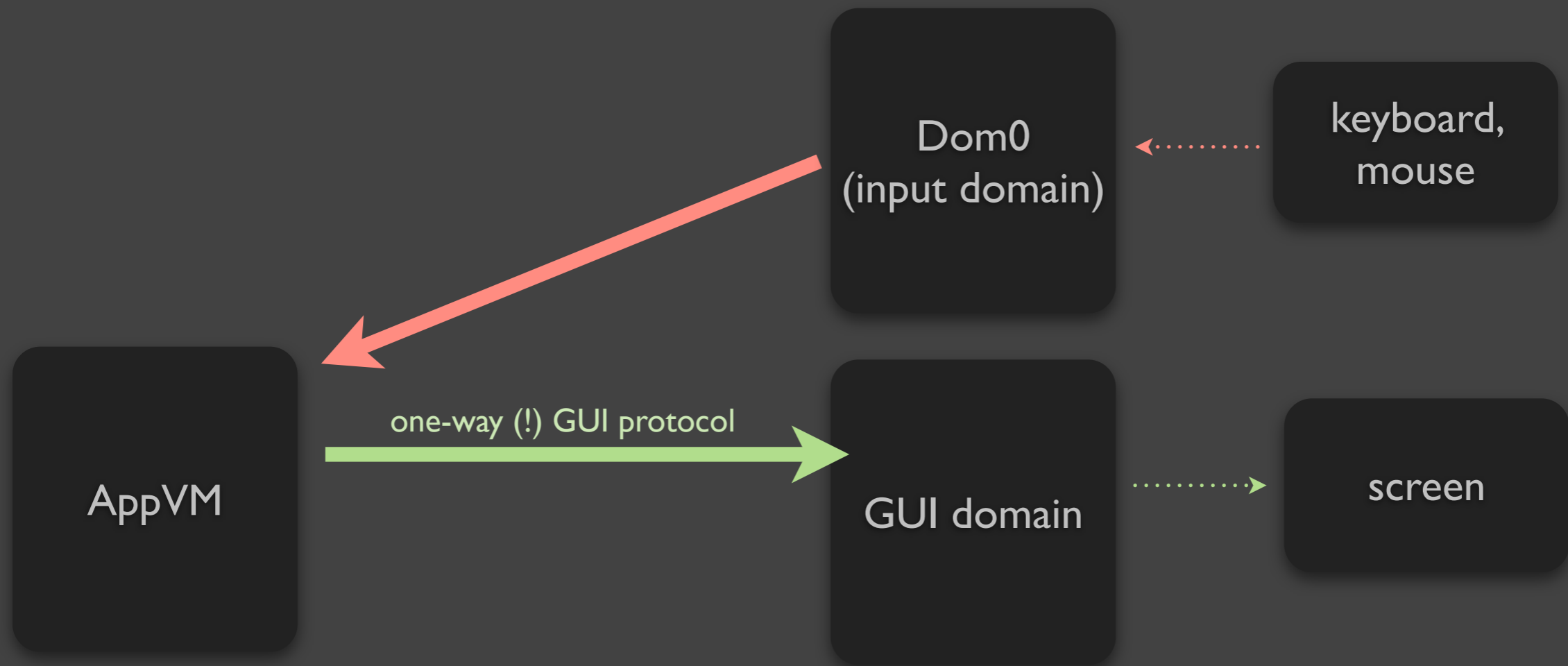
?

We need to restrict(*) any **outbound communication** from GUI domain to other VMs, so that it could not send out the secrets anywhere:



(*)well, at least limit the outgoing bandwidth considerably, as we cannot really eliminate all timing covert channels

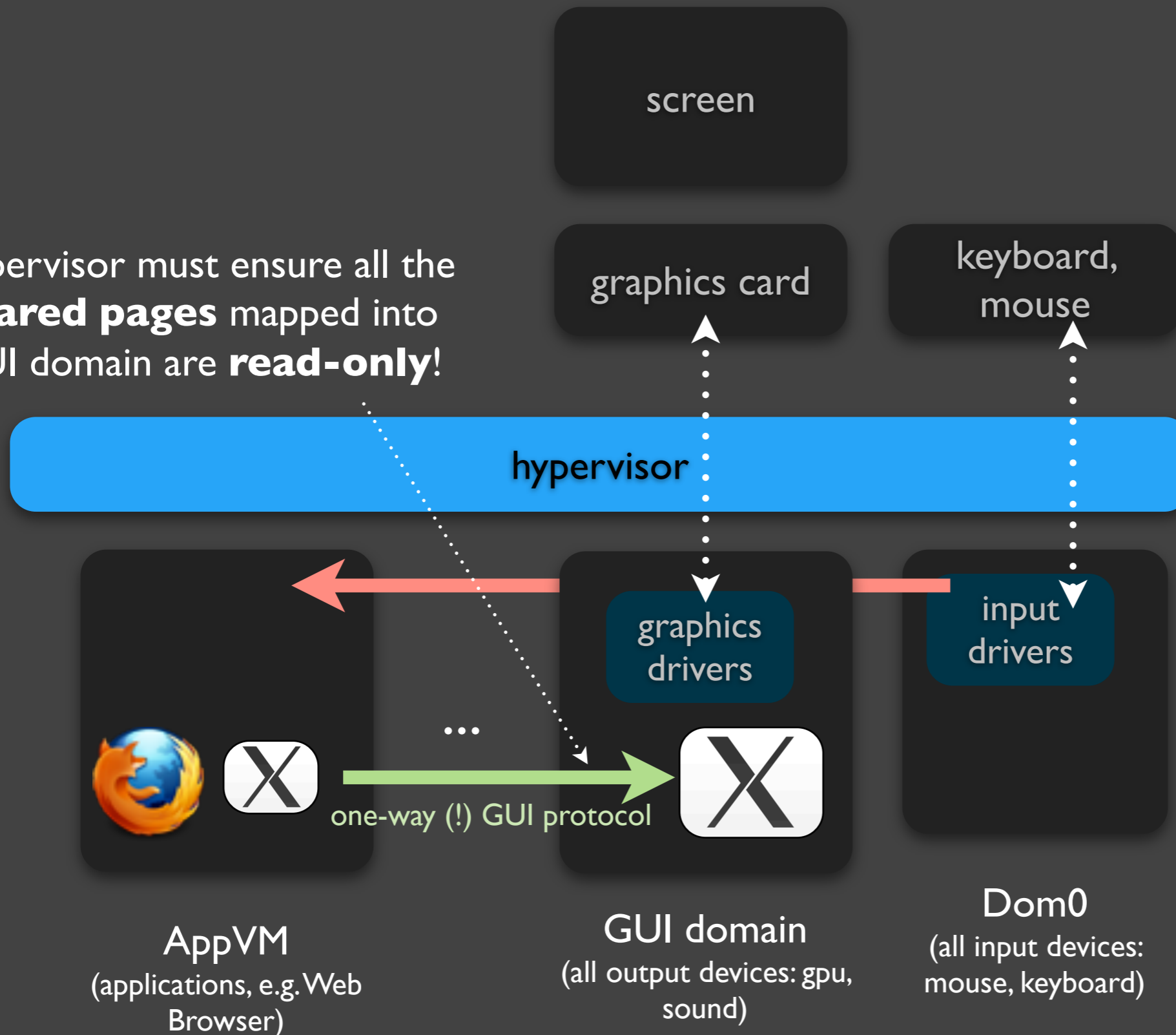
GUI domain should only be used for **output**, but **not for input** (keyboard, mouse) to make sure it cannot manipulate the state of the system in any way:



Putting it all together...

Qubes **Split I/O Model** (planned for Qubes 2.0)

hypervisor must ensure all the **shared pages** mapped into GUI domain are **read-only!**



Benefits

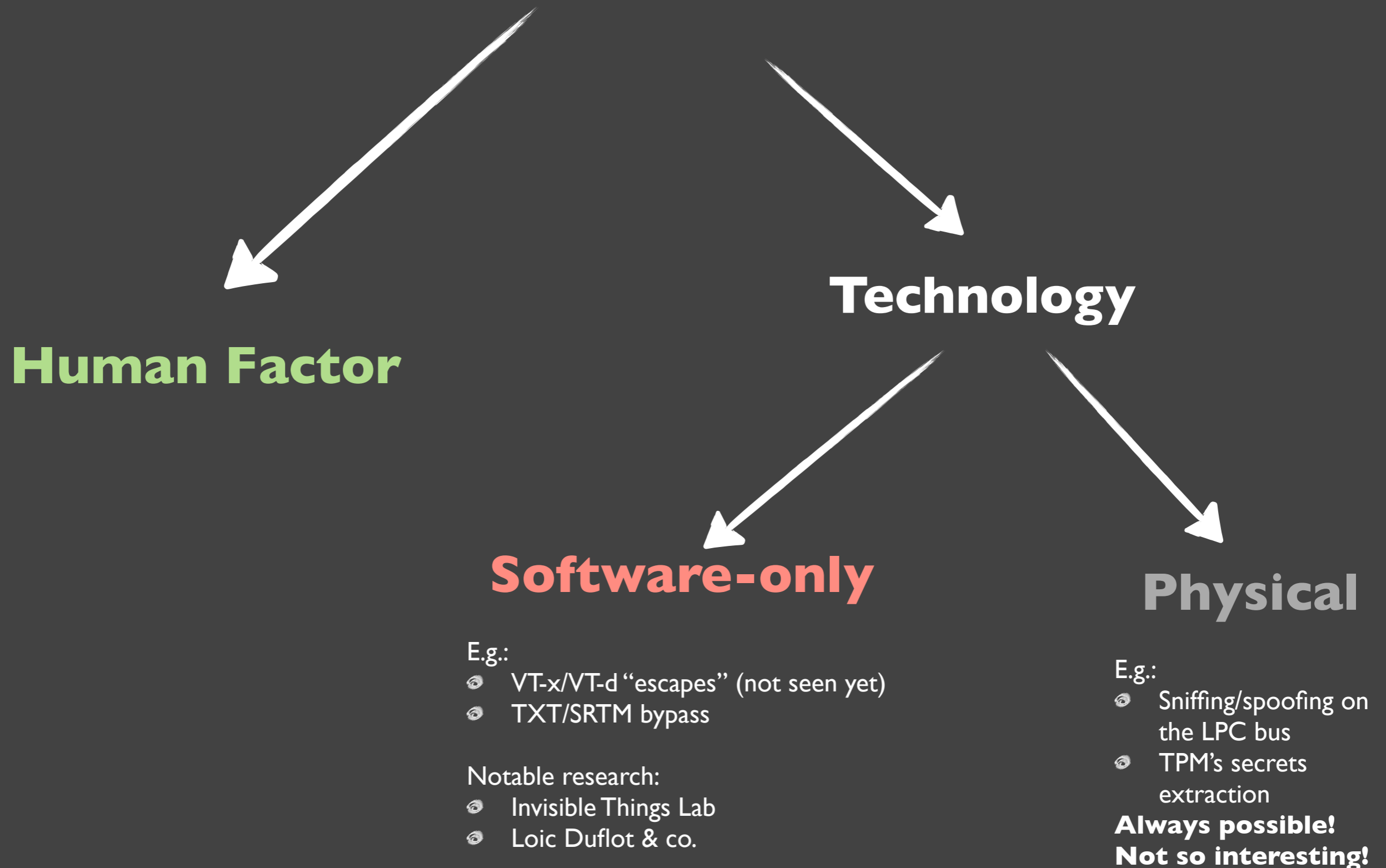
- ✓ GUI domain becomes security non-critical!
- ✓ We can put lots of untrusted (unsigned!) software there:
 - 👁 3rd party, untrusted GPU drivers (Hi Nvidia!)
 - 👁 Complex, buggy, X server (Hi X.org!)
 - 👁 Complex, bloated, buggy, untrusted GUI environments (Hi KDE!)
- ✓ We can expose complex graphics protocols to other VMs, such as 2D/3D h/w acceleration
(No problem if somebody exploits a bug in the protocol/backend!)

Problems?

- The GUI protocol must be uni-directional -- is it possible to create an OpenGL-ish version that would be one-way only?
- Something we're missing...?

So, is Trusted Computing the silver bullet?

Attacks on Trusted Computing



Human Factor vs. Technology?

The Human Factor

- **Which domain/VM** am I using for a given task?
 - Random browsing in “banking” VM?
 - Banking in “random browsing” VM?
- How do I **partition my digital life** into domains/VMs?
 - work, personal, shopping, banking, ...?
 - disposable VMs? (per-document VMs)
- Inter VM **copy-and-paste** and **file sharing**:
 - can a clipboard copied from “random” VM exploit my editor in “work” VM?

Check out **Qubes OS** for implementation of the ideas presented here:

<http://qubes-os.org>

We're currently looking for an
investor in Qubes Pro!

<http://invisiblethingslab.com>

ITL

INVISIBLE THINGS LAB

INVISIBLE THINGS LAB