OS Security

Virtualization

Radboud University, Nijmegen, The Netherlands

Winter 2017/2018

A peek into 2018

- January 9, hoorcollege: Q&A (exam preparation)
- January 11, werkcollege: presentation of last homework
- January 16, hoorcollege: guest lecture

A short recap

- Accept that applications behave maliciously
- Limit damage by separation, compartmentalization, or isolation
- Fairly simple approach: `chroot`
  - "Jail" process inside a subdirectory
  - Filesystem content outside the subdirectory is not visible
  - By design: `root` can escape
- More general concept: OS-level virtualization:
  - OS offers multiple user-space instances
  - Instances are called *containers*, *jails*, or *partitions*
  - Each container can see only part of the resources (I/O, CPUs, memory, file system)
- Example of OS virtualization: Linux containers (LXC)
- Main concepts of LXC:
  - namespaces (restrict what a container can see)
  - cgroups (manage resource sharing between containers)

Advantages of OS-level virtualization

- Fairly easy to set up
- Convenient for testing new configurations
- Essentially no overhead or performance loss
- System-level backups:
  - Take "snapshot" of complete container
  - Easy to restore to sane state from snapshot
- Run multiple environments in parallel
- Example: Development Debian system in `chroot` on Android phone

OS-level virtualization for security?

Relationship status: it's complicated

- Already `chroot` provides more separation than nothing
- `chroot` "jails" may be sufficient for stronger separation of non-root processes
- General limitation: `root` can escape
- Similar with **privileged** LXC containers
- Unlimited `root` can escape containers
- Different: **unprivileged** containers (since LXC 1.0)
- Map container user `root` to non-root user outside the container
- **Kernel and drivers are still part of the TCB**

Additional limitations

- Cannot use containers for testing different OS
- Container snapshots exclude the kernel
- Container snapshots exclude memory content

Emulation

- Can emulate a whole computer **in software**
- Install guest unmodified guest OS in the emulator
- Translate all instructions from guest to code running on the host
- Can even translate from one architecture to another
- Examples:
  - QEMU (Quick Emulator), emulates x86, PowerPC, ARM, SPARC,...
  - Virtual PC for MAC, emulates x86 on PowerPC
  - ZSNES emulating Super Nintendo on Windows, Linux,...
- Disadvantage: serious performance penalty

Paravirtualization

- Idea: Multiple **virtual machines** executing code **natively**
- Requires code for the underlying architecture
- Does not incur performance penalty
- Requires **virtual machine manager (VMM)** or **hypervisor**
- VMM needs higher privilege than OS!
- Idea: run VMM in ring 0, guest OS in ring 1
- Guest OS and drivers **need to be modified**
- Ring-0 instructions need to be replaced by **hypercalls**

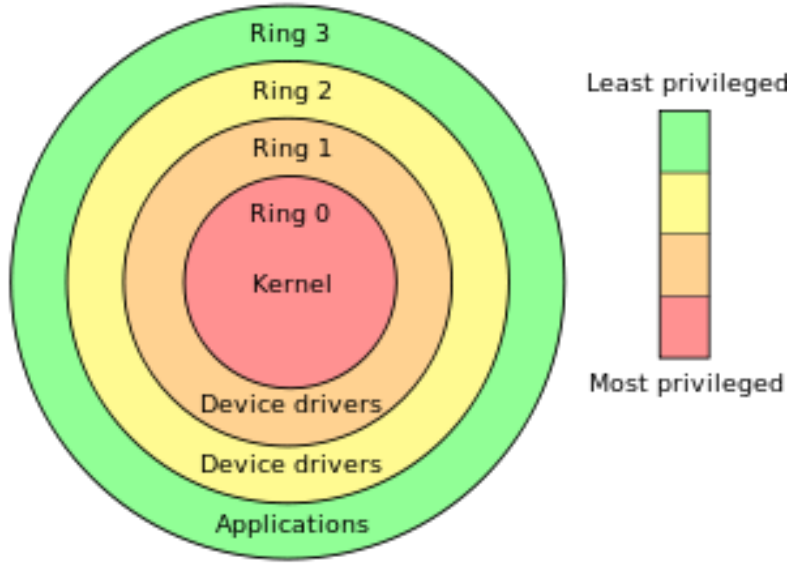


Image source: http://en.wikipedia.org/wiki/Protection_ring

Software full virtualization

- Main disadvantage of paravirtualization: requires modified guest OS
- Question: can we *fully virtualize* machines?
  - Hypervisor (VMM) manages virtual machines
  - Guest OS is *not aware* that it's in a virtual environment
- Concept is old: mid 60s, IBM M44/44X experimental system
- Is it possible on, e.g., x86?

"In 1998, VMware figured out how to virtualize the x86 platform, once thought to be impossible, and created the market for x86 virtualization. The solution was a combination of binary translation and direct execution on the processor that allowed multiple guest OSes to run in full isolation on the same computer with readily affordable virtualization overhead"

https://www.vmware.com/techpapers/2007/understanding-full-virtualization-paravirtualizat-1008.html

- Run everything natively, except ring-0 instructions
- Translate ring-0 instructions on the fly (similar to emulation)

Hardware full virtualization

- Disadvantage of software full virtualization: complex VMM
- For example: need to emulate MMU access in software ("shadow tables")
- VMM is inherently inside the TCB
- Also this is not free of performance overhead
- Solution: do it in hardware
- Introduce **protection ring -1** for hypervisor
- Implemented on various modern architectures:
  - Intel VT-x (and VT-i on Itanium), introduced 2005
  - AMD-V, introduced 2006
  - ARM Virtualization Extensions (VE)
- Modern implementations also include hardware memory virtualization
- Multiple hypervisors for x86/AMD64 hardware virtualization, e.g., VMWare Workstation, Linux KVM, Xen

Xen

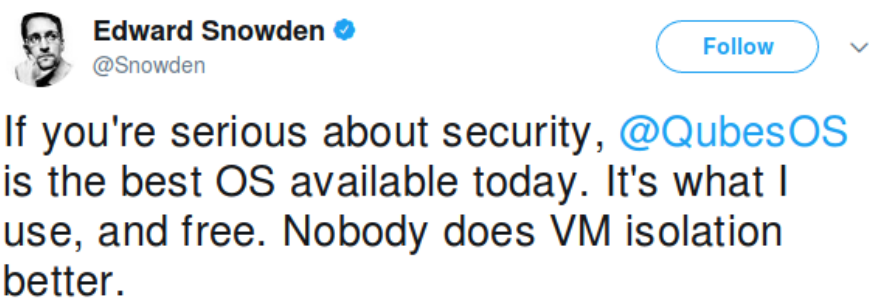
- Xen hypervisor originally used paravirtualization
- Since version 3, Xen is using hardware virtualization
- One privileged VM ("domain") called **dom0**:
  - Initial VM started at boot by the hypervisor
  - Required by the hypervisor
  - Able to create and shut down other VMs
  - Has access to hardware drivers and networking
  - Manages access to shared hardware (e.g., networking) by other VMs
- Unprivileged (all other) VMs are called **domU**
- For security:
  - Run all applications in domU VMs
  - Only admin/management tasks run in dom0
- Malicious applications cannot affect applications in other domU VMS and **cannot affect dom0** (except if they compromise the Xen hypervisor)

I/O Virtualization

- By default, dom0 takes care of networking, USB, Bluetooth
- This is a security nightmare!
- Example 1: BadUSB
  - Reprogram microcontroller in a USB stick
  - Stick will act like a keyboard and "type" malicious commands
- Example 2: BlueBorne (Sep. 2017)
  - Eight vulnerabilities in Bluetooth
  - Affects Android, iOS, Linux, Windows
  - Does not require "pairing" with malicious device
- Example 3: Buffer overflow in Broadcom WiFi driver (CVE-2017-11120)
- Solution: I/O Virtualization
- Assign I/O devices to VMs
- Use input-output memory management unit (IOMMU) for DMA
- Examples: Intel VT-d, AMD-Vi

Qubes OS

- Desktop (mainly laptop) operating system building on Xen
- Developed by Invisible Things Lab (ITL, founded by Joanna Rutkovska)
- Builds on Xen
- Dom0 is running Fedora Linux
- Multiple tools to integrate working with VMs



Dom0, Template VMs, and AppVMs

- Standard for Xen: dom0 is in charge of admin
- For Security: dom0 has no networking
- Updating dom0: download updates in other VM, copy to dom0, verify signatures
- Not perfect security: malicious rpms could exploit bug in GPG verification
- For applications, install software in **template VMs**
- ITL-supported template VMs: Fedora, Debian
- Don't actually *run* template VMs (except for updating)
- Run AppVMs that (typically) mounts the root FS from a template VM read-only
- Only `/home` and `/usr/local` are persistently writeable
- Rebooting an AppVM restores the clean state of the template VM
- Can make changes to AppVMs persistent (but not stealthily so)

Standalone and HVMs

- Three main advantages of template-derived VMs:
  - No need to keep duplicate copies of root file system
  - Easy restore to clean state via reboot
  - Centralized updates
- Disadvantages:
  - We don't want untrusted software in our template VM
  - We are restricted to Linux on AppVMs
- Solution:
  - Standalone VMs that have their own root-file-system copy
  - Hardware VMs (also standalone) to support, e.g., Windows
  - Use Qubes Windows tools for integration with other VMs

Separation by security domains

- Create different AppVMs for different tasks or roles
- Example:
  - AppVM for e-mail
  - AppVM for online shopping
  - AppVM for banking
  - AppVM for Skype
  - AppVM for general web browsing
  - AppVM for research
  - AppVM for teaching
  - ...
- VMs have colors, typically used to indicate "trust level"
- Each of those AppVMs has their own home directory
- Compromising the Skype VM does not give an attacker access to e-mail, banking TANs, online-shop logins, exams, ...
- That's even with full root access to the Skype AppVM!
- Consequence: No root password by default

Messaging and data transfer

- Full separation is too strong, wouldn't allow typical workflow:
  - Attach pdf of a research paper to e-mail
  - Download a picture for lecture slides
  - Send exam questions in an (encrypted!) e-mail to colleagues
- Solution: secure file copy between VMs
  - `qvm-copy-to-vm VMNAME FILENAME`
  - Copies to target VM into `/home/user/QubesIncoming/SOURCEVM`
  - Won't overwrite existing files
  - VMs other than source or destination of the copy cannot "steal" the file
  - Feedback (confirmation dialog) presented to user by dom0
- Qubes inter-VM-copy uses Xen shared memory
- Can copy-paste via Qubes clipboard
  - Copy from local clipboard of source VM by CTRL-Shift-C,
  - Paste into local clipboard of destination VM by CTRL-Shift-V
  - Pasting from global clipboard will clear contents

DispVMs

- Common scenario: you receive a pdf by e-mail
- Question: Where/how do you open this pdf?
- Considerations:
  - The pdf is highly untrustworthy, shouldn't open in mail VM
  - What should happen if we click on the pdf?
- Solution: `qvm-open-in-dvm`
- Create lightweight **disposable VM** (DispVM), copy pdf in there, open
- DispVM is destroyed after you close the pdf
- Careful when opening and editing an office document
- You save the document, close LibreOffice and all changes are lost!

GUI separation

- Why would you go through the hassle of using Qubes?
- Maybe the best answer: **GUI separation**
- In Linux, under X, every program can read and write mouse and keyboard input to all other programs!
- Wait, what?!

"If you have two GUI applications, e.g. an OpenOffice Word Processor, and a stupid Tetris game, both of which granted access to your screen (your X server), then there is no isolation between those two apps. Even if they run as different user accounts! Even if they are somehow sandboxed by SELinux or whatever! None, zero, null, nil!" "The X server architecture, designed long time ago by some happy hippies who just thought all the ~~people~~ apps are good and non-malicious, simply allows any GUI application to control any other one. No bugs, no exploits, no tricks, are required. This is all by design. One application can sniff or inject keystrokes to another one, can take snapshots of the screen occupied by windows belonging to another one, etc."

—Joanna Rutkovska

https://blog.invisiblethings.org/2011/04/23/linux-security-circus-on-gui-isolation.html

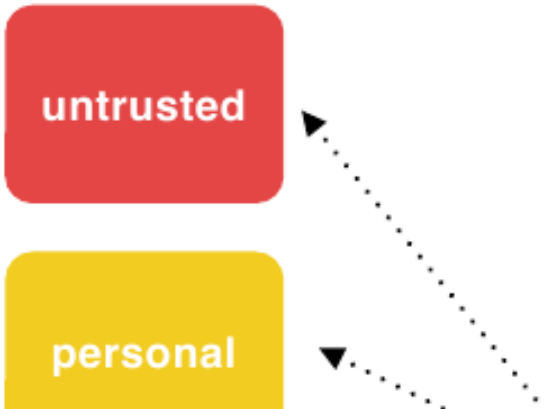
Easy demo

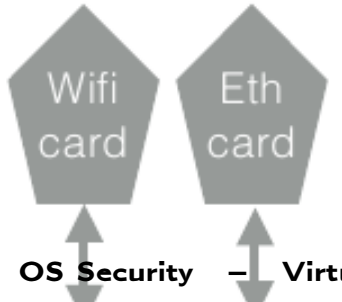
Full screen

- You may have noticed the yellow border around the slides
- Qubes by default does not support fullscreen mode for AppVMs
- Reason: Fullscreen software can trick user
- Examples:
  - Pretend that screenlock is active, obtain user's password
  - Pretend to crash and fake other application's input (e.g., browser with Facebook login page)
- Qubes' colored frame *always* makes clear in what VM you are
- Can enable full-screen through AppVM configuration in dom0

Networking

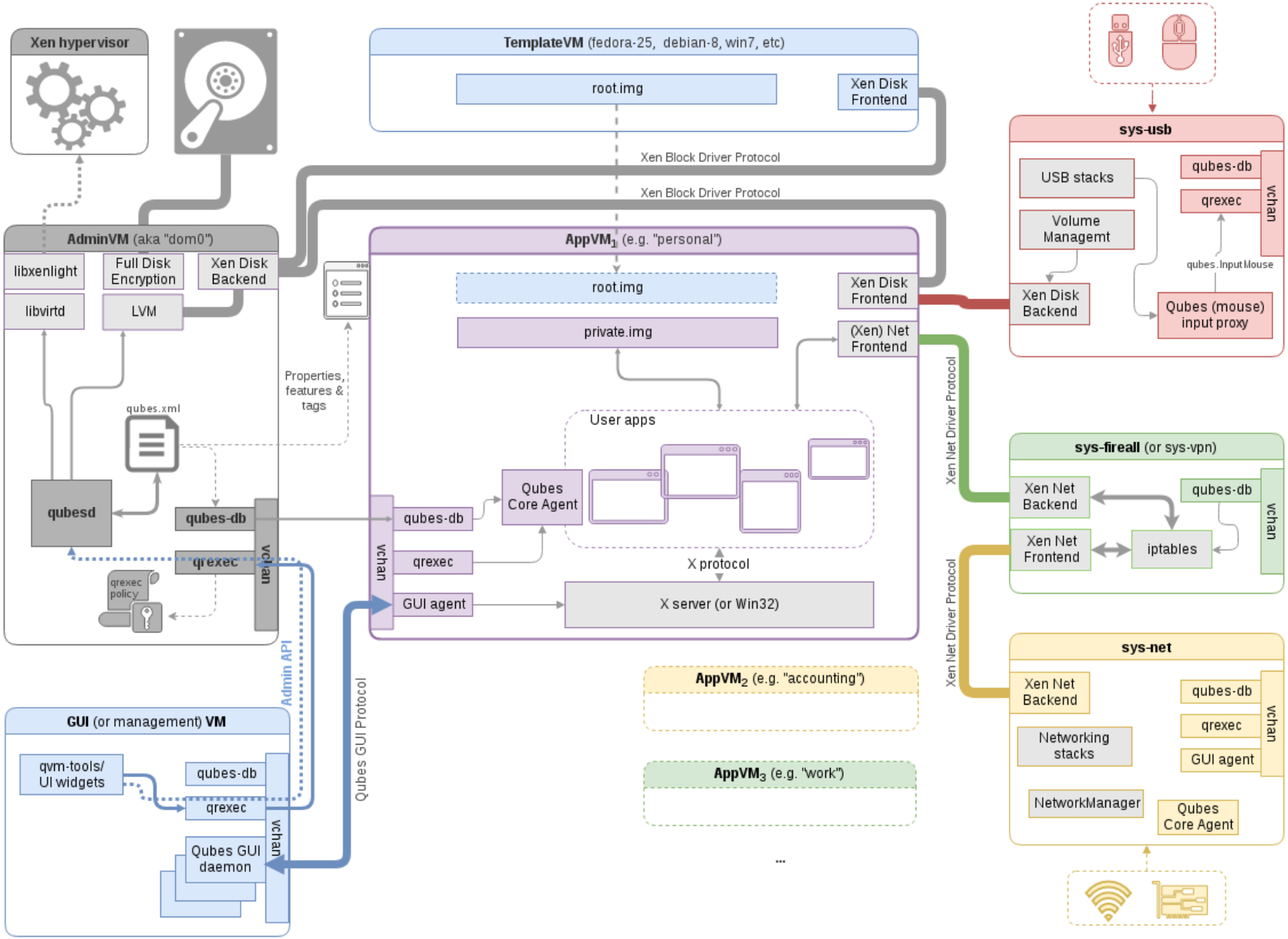
- NIC drivers and network stack are naturally exposed
- Need to keep them out of dom0
- Use I/O virtualization to map network devices to `sys-net` VM
- `sys-net` does not contain any sensitive data, except for WiFi passwords
- AppVMs connect via `sys-firewall` through `sys-net`
- Firewall rules for AppVMs are stored in dom0, provided to and enforced by `sys-firewall`
- Ideally limit network access of trusted AppVMs
- Examples:
  - `mail` VM only has access to SMTP and IMAP server
  - `research` VM only has access to some git servers
  - `shopping` VM only allows HTTPS





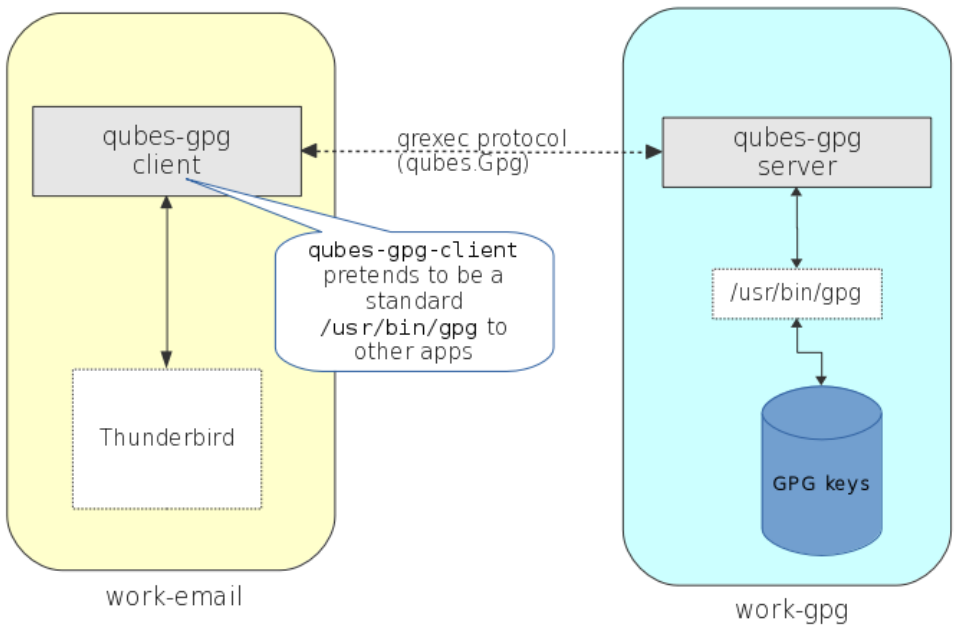
USB

- Would you plug a USB stick somebody gives you into your laptop?
- Generally a large security risk; used to compromise air-gapped systems in the real world!
  https://www.youtube.com/watch?v=ZI5fvU5QKwQ
- Qubes optionally creates `sys-usb` VM
- USB VM has control over all USB controllers
- BadUSB attack would compromise only untrusted VM
- Disadvantages:
  - Slightly more effort to copy files to USB stick
  - No USB input devices in other VMS


Xen hypervisor
TemplateVM (fedora-25, debian-8, win7, etc)
root.img
Xen Disk Frontend
Xen Block Driver Protocol
Xen Block Driver Protocol
sys-usb
USB stacks
Volume Managemt
qubes-db
qrexec
vchan
qubes.InputMouse
Xen Disk Backend
Qubes (mouse) input proxy
AdminVM (aka "dom0")
libxenlight
Full Disk Encryption
Xen Disk Backend
libvirtd
LVM
AppVM1 (e.g. "personal")
root.img
private.img
Xen Disk Frontend
(Xen) Net Frontend
Properties, features & tags
qubes.xml
User apps
qubesd
qubes-db
qrexec
vchan
qrexec policy
Qubes Core Agent
X protocol
GUI agent
X server (or Win32)
Xen Net Driver Protocol
sys-firewall (or sys-vpn)
Xen Net Backend
Xen Net Frontend
iptables
qubes-db
vchan
Xen Net Driver Protocol
sys-net
Xen Net Backend
Networking stacks
NetworkManager
qubes-db
qrexec
GUI agent
Qubes Core Agent
vchan
AppVM2 (e.g. "accounting")
AppVM3 (e.g. "work")
...
Admin API
Qubes GUI Protocol
GUI (or management) VM
qvm-tools/ UI widgets
qubes-db
qrexec
Qubes GUI daemon
vchan

Qubes and GPG

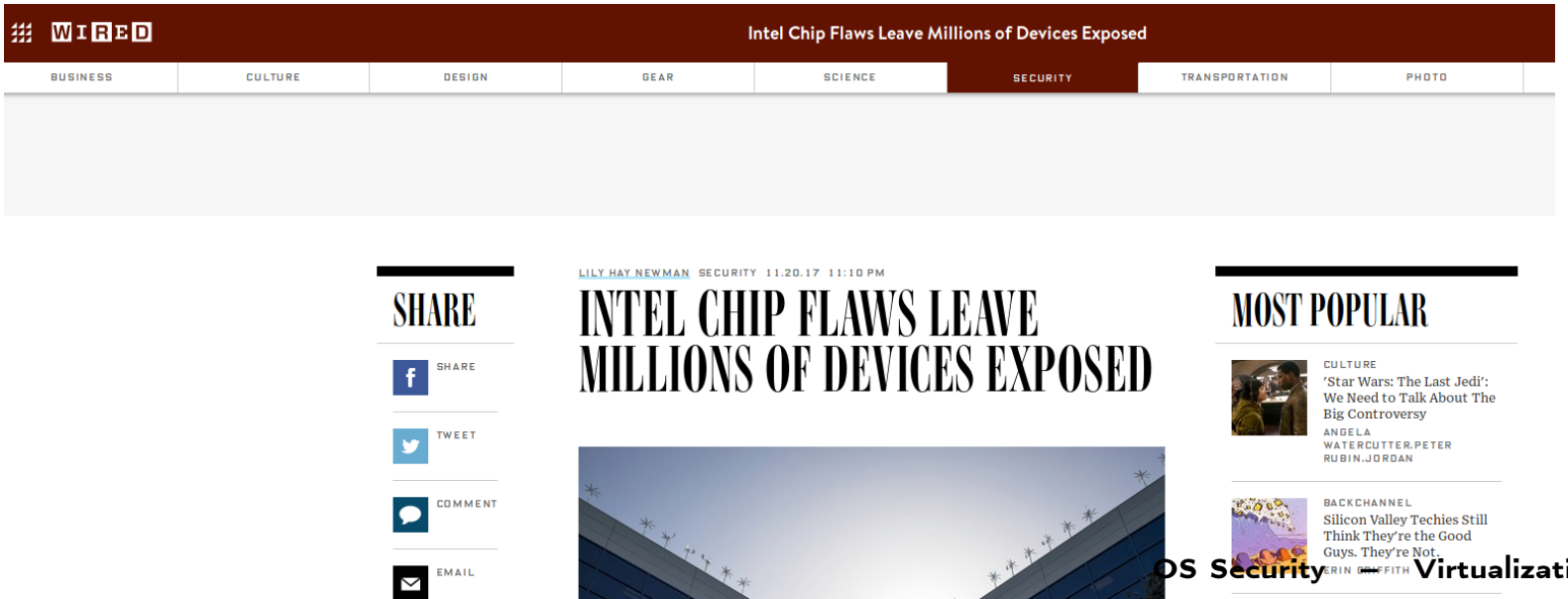
- `mail` VM needs access to GPG keys
- How trusted is your `mail` VM?
- Wait, *does* the `mail` VM need access to GPG **keys**?
- It doesn't, it needs access to **decryption** and **signing**
- GPG can be used with a smartcard, keys never leave the card
- Idea: simulate a smartcard in a trusted VM (no network access)



https://www.qubes-os.org/doc/split-gpg/

Limitations of Qubes

- Aiming at single-user laptop usage scenario
- Security by isolation creates small overhead in workflow
- Not (yet) the OS I would recommend my mother to use
- Relies on security of Xen
- Has been vulnerable via Xen vulnerabilities in the past
- Relies on security of various components in dom0
- Relies on security of more privileged components than hypervisor:
  - Intel SMM ("System Management Mode"), protection ring -2
  - Intel ME ("Management Engine"), protection ring -3
  - BIOS
  - Hardware



General design principles for secure systems

- Move critical, exposed components outside of TCB
  - Network stack
  - USB
- Shrink TCB to necessary minimum
- Ideally: formally verify TCB
- Separate security domains (with corresponding data)
- Limit damage done by malicious non-TCB software


Prettige
Kerstdagen!