Thoughts on Client Systems Security

Joanna Rutkowska
Invisible Things Lab

SSTIC 2011, Rennes, France, June 2011
Why client systems security is important?
If your client device (laptop, tablet, phone) is compromised...
... all the security is lost!
Client systems are your **eyes** and **fingertips**

- The client OS can see what you see on the screen (decrypted)
- The client OS can pretend to be you
Approaches to building secure (client) systems
Approaches to secure systems

- Safe Languages
  - Thin microkernels & hypervisors
  - Common-sense verification
  - Networking
  - Storage
  - Drivers virtualization (IOMMU)
    - GUI?

- Security by Isolation
  - Small TCBI
  - Untrusted subsystems
    - Examples
      - Xen Client
      - Qubes
      - Google Chrome
Security by Isolation:
Goals
Isolation between two apps...
Isolation between two apps...
Isolation between two apps and the OS...
What we isolate

- Address space isolation
- FS isolation
- Networking isolation
- GUI isolation
GUI-level isolation
Lack of GUI isolation on many Windowing Systems...
App1

Inject keystrokes

Take screenshots

Inject keystrokes

App2

Sniff keystrokes
Fat GUI APIs that are likely to be buggy (and exploitable)
GUI API (Xlib, OpenGL, ...)

GPU

App1

App2
Work email

Tetris
Bank Browser

Personal Browser
We don't want two apps to be able to interact with each other via X/OpenGl/GPU!

(Xorg people still don't get it, after 20+ years...)
Anyway…
Let's imagine we implemented strong isolation...
We still must allow the user to bypass it sometimes!
Data flows between domains

Clipboard

File sharing
Down-transfers vs. Up-transfers
Trust level

App2

App1

"Down Transfer"
"Traditional" school of thought:

Never allow **down-transfers**!

Even between two *cooperating* domains!
Rationale: never allow to move more sensitive data (e.g. Embassy cables) to less trusted domain (e.g. The Internet)
OS should never allow for this flow!
This requires elimination/drastic reduction of all potential *cooperative covert channels* between the apps/domains!
I seriously doubt this is possible on modern x86 hardware...
Covert channels via CPU cache
Covert channels via GUI/GPU
Covert channels via networking
Covert channels via other subsystems
?
"Qubes" school of thought:

Avoid up-transfers!
Rationale: an up-transfer can potentially compromise a buggy app in the destination domains (untrusted input processing)
Some up-transfers are difficult to avoid...
Copying a link found on the Internet, and emailing it to a colleague at work
Copying a cool cartoon found on the Internet into work confidential report/presentation
Solution: use trusted converters, e.g. for all JPEGs?
Trusted converters
Written in safe languages

- Perhaps we can ensure the converter won’t be exploited...
- Concurrency safety
- Memory safety
- ...What if the bug in the dest app can be triggered by _legal_ file format?

- Need to handle each file type separately
  - JPEG
  - PDF
  - ...
  - It’s just a string of bytes...
  - Might be pasted into many different apps...
  - Each app might interpret it differently
  - E.g. An URL string

- How to handle clipboard?
Another types of problems related to file sharing is **FS Metadata parsing**
Two air-gapped systems

Machine 1

Machine 2
Two air-gapped systems

Machine 1

Machine 2

Copying data using USB stick
Two air-gapped systems

Machine 1

Exploit

Machine 2

The sticks partition table turned out to be malformed...
In Qubes we copy files between domains using shared memory and simple cpio-like tool (this cpio-like tool is the security critical code)
Limitations of Security by Isolation approach
Security by Isolation doesn't protect your apps from being compromised!
Work email

Random Web browsing
Exploiting hypothetical bug in my email client's OpenSSL
Exploiting hypothetical bug in my email client's OpenSSL

MTIM

Mail server

Work email

Random Web browsing
My recent adventure in a hotel in Paris ;)}
Solution: decompose the app! (More security by isolation!)
Capsicum is working on such app-level decompositions (will definitely use in Qubes when ready)
Another approach: safe languages

(so, where can I get thunderbird-like app written in C#?)
Security by Isolation: Useful technologies
Technologies for **address space isolation**

- **MMU**
- **Virtualization** (VT-x/AMD-v, EPT/NPT)
<table>
<thead>
<tr>
<th>MMU</th>
<th>VT-x/EPT</th>
</tr>
</thead>
<tbody>
<tr>
<td>User mode (ring 3)</td>
<td>Guest mode (non-root)</td>
</tr>
<tr>
<td>Kernel mode (ring 0)</td>
<td>Hypervisor (root mode)</td>
</tr>
<tr>
<td>Page Tables</td>
<td>Extended Page Tables (EPT)</td>
</tr>
<tr>
<td>Exceptions (#GP, #PF, ...)</td>
<td>VM exits</td>
</tr>
</tbody>
</table>
### Differences

<table>
<thead>
<tr>
<th>MMU</th>
<th>VT-x/EPT</th>
</tr>
</thead>
<tbody>
<tr>
<td>User mode and kernel mode often share the same address space (e.g. 3/1GB split on 32bit Linux)</td>
<td>Guest and the hypervisor never share the same address space</td>
</tr>
<tr>
<td>SIPI interrupts kernel execution</td>
<td>SIPI is blocked in VMX</td>
</tr>
</tbody>
</table>

SMEP somehow eliminates this difference

Interrupt Remapping makes this irrelevant anyway
So, why bother using virtualization?

Why not just use the good old MMU for address space isolation?
For **compatibility** with OSes that are not para-virtualizable

- Windows
- Mac OSX

Linux is PV aware and we can virtualize it using MMU under Xen (Run it as ring3, no need for VT-x)
But why would we want to virtualize the OS in the first place?

A virtualized buggy, messy OS is still... a buggy, messy OS!
Because we want to use the OS as an API provider!
Everything and the kitchen sink!
But those (legacy) apps expect a POSIX API, they don't know how to talk to the backends.
So we must virtualize the whole OS to provide API for legacy apps...
TCB (microkernel/hypervisor)

- NICs
- SATA, USB
- GPU, keyboard

Frontends
- App1
- App2
- App3

Apps see POSIX APIs

Domain 1

Domain 2

Backends
- Storage drivers and backends (block, pvusb)
- Networking Drivers & stacks

Untrusted subsystems

Backend
- GUI
But it is not like virtualization (VT-x) provides stronger security than MMU!
IOMMU (VT-d)
IOMMU allows to sandbox drivers and devices, so plays a key role in TCB disaggregation...
IOMMU: catches
For safe language-based OSes (e.g. Singularity and derivatives) IOMMU is needed to restrict devices to accesses to their DMA buffers only to preserve memory safety
Catches:

- MSI attacks
- BDF Spoofing
- Reflashing device firmware?
- Interrupt Remapping (see our latest paper on VT-d escapes)
- PCIe ACS
- DMA-resistant trusted boot
We really need more trusted trusted boots!  
(subject for another presentation)
No secure client systems without IOMMU and trusted boot!
Security by Isolation: Challenges
How to partition my digital life into security **domains**?
Do we actually need domains? Perhaps we can just isolate each app from each other app?

We need OSes to provide legacy APIs to apps.

Would be a waste of memory to have one instance of an OS per each app...
But even if we did isolate (virtualize?) on a per app granularity, still the problem of partitioning doesn't go away...
Unless we get 100% safe languages we would not avoid security by isolation...
Other challenges
GPU multiplexing
USB multiplexing
I'd love to discuss that last two problems!
Qubes OS implements lots of ideas mentioned here
Qubes

Xen as the microkernel
- Small TCB
- Good IOMMU support
- Reasonable ACPI support

Untrusted networking subsystem
- Uses IOMMU
- Separate firewallsing domain
- Support for optional separate VPN domains
- No inter-domain networking
- Support for many isolated NetVMs
  - Each using different NIC
- Host X server never exposed to domains (VMs)
  - Secure clipboard between domains
  - Secure file copy between domains
- Other domains cannot steal the clipboard
- No volume/fs metadata parsing by the receiver
- Each app's window labeled with domain name and color
- Trusted Window Manager handles the labeling
- Tray icons integration!

Safe GUI
- Centralized updates for all domains
- Save on disk space
- Template root fs are shared as read-only

Shared templates (rootfs)

Untrusted storage subsystem
- Requires TXT
- Planned for Qubes 2.0
Qubes is not a microkernel....

... It's everything else!
Qubes-OS.org
Thanks!