X.org security

Recap, vulnerabilities, attacks and discussions on the graphic stack's security

Martin Peres & Timothée Ravier

Ph.D. student at LaBRI, System security engineer

September 19 – 21, 2012

Disclaimer

- We are not Linux **graphics** stack developers (yet?);
- We are interested in (desktop/mobile) UI security;
- This presentation is based on our study and is (likely) incomplete (mostly focused on Linux);
- Feel free to interrupt us.

Summary

Confidentiality

Use cases 0 & 1

- The user is shopping online ;
- He/she keys in the credit card number;
- A keylogger was installed on the computer
  or
  A program takes periodical screenshots;
- His/her credit card number got stolen!

Confidentiality & the X-server

User's expectation towards confidentiality

- Applications should never be able to access other applications' input events or output buffers (allow only copy/paste);
- $\Rightarrow$ Apps should not be able to eavesdrop other apps' input events (keyloggers) nor their output buffers;
- $\longmapsto$ This would make e-shopping safer on the system-side.

X11 & X-server

- Grants full-access to whoever can read the magic cookie;
- **Security model**: Applications run by a user should be trusted. Isolation between users only;
- **Problem**: applications cannot be trusted anymore and some apps can be launched behind the user's back;
- $\Rightarrow$ This busts confidentiality!

Integrity

Use case

- The user is visiting his bank's website;
- He/she checks the website address (https + right domain);
- He/she is unaware that he/she is visiting a fake website and that Firefox's address bar has been redrawn by a malware;
- His/her bank information got stolen!

Integrity & the X-server

User's expectation towards integrity

- What is displayed is what the application drew;
- The events sent to the application are never tampered with;
- $\Rightarrow$ Applications should never be able to alter other applications' output buffers or input events;
- $\longmapsto$ Help blocking Phishing-like attacks.

X11 & X-server

- Apps can inject input events (virtual keyboards);
- Apps with DRI 1 can render outside their "window";
- $\Rightarrow$ This busts integrity!

Availability & the X-server

User's expectation towards availability

- Users think their computers do multitasking;
- Thus, one app shouldn't be able to bring the system down;
- $\Rightarrow$ Applications should never be able to deny access to other applications.

X11 & X-server

- Apps can act as screen lockers;
- Virtual keyboards may kill applications they want using XF86ClearGrab (the famous security hole of xserver 1.11);
- $\Rightarrow$ This busts availability!

Current mitigation techniques

XACE $\mapsto$ X11 hardened

- Provides a finer-grained access control in X11;
- Mostly per-feature access control with some clipboard control;
- XSELinux: Deactivated by default in Fedora/RHEL/CentOS as users are unconfined.
- Use of sandbox services (Xephyr) recommended instead;
- $\Rightarrow$ Still too coarse-grained to be fully useful.

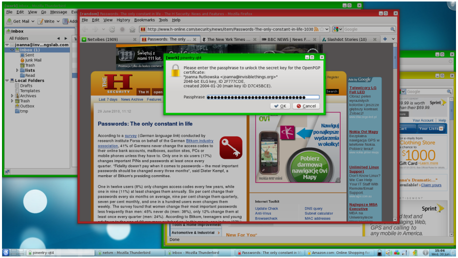
Isolate groups of applications into domains

- QubesOS : Isolation using virtual machines;
- PIGA-OS : Isolation using SELinux + XSELinux + PIGA-SYSTRANS;
- $\Rightarrow$ Force applications to communicate via a controlled system.

QubesOS

QubesOS

- Allows the user to group applications into domains;
- Each domain requires a new Xen Virtual Machine (VM);
- Applications from the VM integrate with the original desktop but are outlined with a specific colour;
- A daemon in dom0 provides a mean of communication between the VMs and does Mandatory Access Control (MAC).



QubesOS

Pros

- Allows defining activities and keep files separated (Taxes, e-shopping, private mails. . .);
- $\Rightarrow$ A compromised domain cannot interfere with other VMs;
- Uses Xen but could also use cgroup/LXC;
- Provides a nicely-integrated GUI to ease setup.

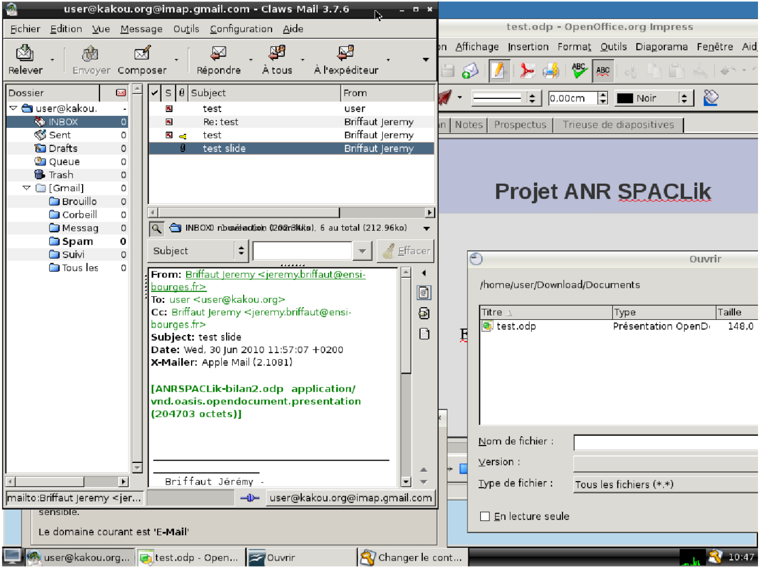
Cons

- Slow and resource heavy;
- Hardware graphic acceleration limited to the number of GPU (with PCI passthrough which requires an IOMMU);
- Limited power management;
- Is Xen able to securely isolate VMs?

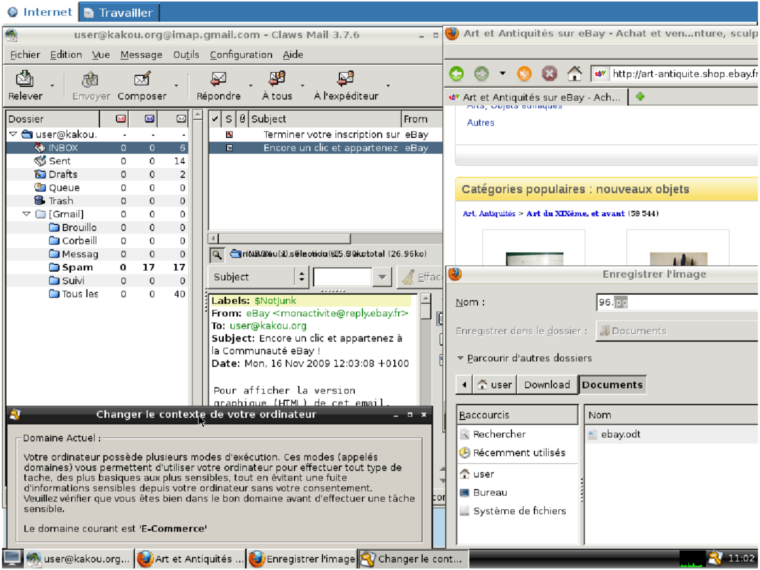
PIGA-OS

- Each applications is put inside a SELinux domain (Type);
- Files, processes, sockets are tagged with a SELinux label;
- A SELinux policy is set for every application and every activity;
- XSELinux is also used to restrict permissions inside the Xserver;
- A daemon (PIGA-SYSTRANS) grants rights as needed and prompts the user if he would like to enter a new domain depending on his/her activity.

PIGA-OS : Example domain Email



PIGA-OS : Example domain E-Shopping



PIGA-OS

Pros

- No need for a virtual machine;
- We can use graphic acceleration for all apps!
- Dynamically adjusts applications' permissions according to the user's activity (if the user agrees with it);
- The model can be re-used if new confinement means appear;
- Power management available.

Cons

- Requires SELinux and a SELinux policy;
- Finer-grained so harder to configure;
- No declassification method provided (yet?).

Summary

Input security & Wayland/Weston

Input confidentiality

- Weston knows where applications are on the screen;
- It decides which applications receive input events (currently selected, under the cursor. . .) $\Rightarrow$ no broadcasting;
- $\Rightarrow$ This defeats keyloggers.

Input integrity

- Weston does not receive input events from applications;
- Input events can not be forged (access to /dev/(u)input restricted to the root user);
- $\Rightarrow$ Virtual keyboards will be discussed later.

Output security & Wayland/Weston

Output buffers confidentiality & integrity

- Weston shares output buffers with applications using the GEM interface to limit buffer copy;
- The GEM handle is a 32bit integer;
  $\Rightarrow$ This can be guessed or easily bruteforced!
- Applications output buffers can be eavesdroped and modified.

Possible solutions

- Add access control to GEM (turn it into GEM2)?
- DMA-BUF for userspace? Access control in DMA-BUF?

Availability & Wayland/Weston

Requirements

- Applications shouldn't crash the compositor;
- Applications shouldn't deny access to other applications.

Vulnerabilities

- Screenlocking;
- Any idea?

Screenlocking

Goals

- Unbypassable screen;
- Ask for a user secret or device to login;
- Enable users to switch or start new sessions.

Recomendations

- Control which applications are able to lock the screen;
- Make sure it uses PAM so we can extend loggin methods.

How do we allow exceptions?

Visual keyboards

- They need to send input events to the compositor;
- Could be included into the compositor.

Screenshot applications

- They need access to the global buffer;
- They can easily break confidentiality.

Global shortcuts

- Media players use global shortcuts to interact with the user;
- They should register key combos to the compositor in order to receive those events;
- Where is the limit (keyloggers, user configured shortcuts)?

Proposal: a MAC framework

Mandatory Access Control

- Control enforced by the system (mostly the kernel);
- Based on a policy (no unprivileged user control).

Suggestions

- Should be implemented as a library to unify access control on every wayland compositors;
- Should define which applications are allowed to take screenshots/act as virtual keyboards/copy & paste/drag & drop/register global shortcuts...
- Generic model, could look like or be polkit;
- Integrating SELinux to use policy mecanisms.

Rootless Weston ?

Incentives

- Compositors have access to everything;
- They will only get bigger as the feature list grows;
- They will have vulnerabilities.

What's blocking us?

- Input management;
- Output buffer management.

Possible solution

- Separate the privileged code from the functional one;
- Use UNIX sockets to forward file descriptors (drm + input).

Summary

1. Expected security properties & X-server
2. About Wayland/Weston
3. Hardware/Driver security

Why should we care about the driver/hardware security?

Requirements

A driver/hw should not allow privilege escalation and should isolate GPU users:

- User ID;
- Confidentiality: read access to other buffers;
- Integrity: write access to other buffers.

Current status

- Good access control to the RAM and VRAM from the CPU;
- The GPU may provide read-write-access to the whole VRAM/Host RAM range to UNIX users through the use of Shaders/GPGPU/copy-engines (TEGRA 2);
- The nVidia driver allows users to access the GPU's registers.

Driver/Hardware security : Current solutions

Expose a secure API to the userland

Goal: Users shouldn't be able to interfere with other GPU users

- The kernel should expose a sane API that isolate GPU users;
- This API should be the only way for a user to access the GPU;
- $\mapsto$ no regs should be accessible from the userspace!

Restrict GPU's RAM access rights

Goal: Deny access to the GPU to the kernel's internal structures or other programs' data.

- VGA window: The GPU can access the first 1.5MB of RAM;
- AGP aperture: Allow GPU access to a fixed part of the RAM;
- IOMMU: Programmable MMU for devices to grant RAM access as needeed where needed.

Driver/Hardware security : Current solutions

Isolate users in a separate VM

Goal: Restrict a GPU user to its own data by abstraction the memory address space

- Most secure solution;
- Increase context-switching delay (problem with DRI2 and Qt5)
- Currently used by: Nouveau (geforce 8+);
- Could also be used by: AMD (Southern Island+), Intel (Sandy Bridge+), . . .

Driver/Hardware security : Current solutions

Isolate users through Command Submission validation

Goal: Restrict a GPU user to its own data by checking the commands issued by the user

- Lower context-switching delay;
- Higher CPU usage in kernel space;
- Currently used by: Radeon, Intel;
- Can be used by: any driver on any card.

Driver/Hardware security : Possible solutions

Zero buffer content at allocation time

Goal: Restrict a GPU user to its own data by zeroing buffers at allocation time

- Increase confidentiality;
- Prettier output;
- High-performance hit on memory-intensive applications;
- Solution: Zero un-used buffers when idle?

Limits to per-GPU-user isolation

- Driver/Hardware can provide isolation between GPU users;
- Compositors have access to applications' output buffers;
- $\mapsto$ The compositor and its plugins should also be secured.

Compositor $\leftrightarrow$ plugins Interface

- Plugins shouldn't have access to buffers (when possible);
- Plugins shouldn't have access to inputs (when possible);
- We should make it hard for plugins to access output buffers;
- Buffers should be located at random addresses: Address-Space Layout Randomization (ASLR) in the driver?
- Applications generating a pagefault should be killed.

Conclusion

Goals

- Make it possible to implement activities and provide secure isolation between them (like QubesOS/PIGA-OS);
- Allow the user to decide what he wants (per-application isolation vs performance?);
- Be ready for GPGPU shared clusters and the soon-to-come WebGL applications.

Current state

- No confidentiality/integrity between applications run by the same user:
  $\mapsto$The Linux graphics stack make it possible to spy on users.

Needed work

- Increase isolation between GPU users.

Thank you for listening!

Martin Peres: martin.peres@labri.fr

Timothée Ravier: timothee.romain.ravier@gmail.com