CLIP OS: Building a defense-in-depth OS with the Linux kernel and open source software

Timothée Ravier, Nicolas Godinho, Thibaut Sautereau

Agence nationale de la sécurité des systèmes d’information (ANSSI)

Journées nationales du GDR Sécurité

12 juin 2019
About the ANSSI

- *Agence nationale de la sécurité des systèmes d’information*
- French authority in the area of cyberdefence, network and information security
- Provides its expertise and technical assistance to government departments and businesses and plays an enhanced role in supporting operators of vital importance.
CLIP OS?

- Linux distribution developed by the ANSSI
- Initially only available internally
- Now open source, mostly under the LGPL v2.1+
- Code and issue tracker hosted on GitHub\(^1\): 
  - Version 4: available as reference and for upstream patch contribution
  - Version 5: currently developed version, alpha status, beta coming soon

\(^1\) [https://github.com/CLIPOS](https://github.com/CLIPOS)
\(^2\) [https://github.com/CLIPOS-Archive](https://github.com/CLIPOS-Archive)
CLIP OS?

Not yet another Linux distribution

- Not a generic/multi-purpose distribution

Targets three main use cases

- Mobile office workstation
- Remote administration workstation
- IPsec gateway
Hardened OS

- Based on Gentoo Hardened
- Hardened Linux kernel and confined services
- No interactive root account available:
  ⇒ "Unprivileged" admin, audit and update roles
- Automatic updates using A/B partition model (similar to Android 7+)
- Multilevel security:
  - Provide two isolated user environments
  - Controlled interactions between isolated environments
Multilevel from the end user point of view (v4)
Admin panel: devices assignment per level (v4)
Differences with Qubes OS

CLIP OS development began 5 years earlier than Qubes OS

Main goals
- We target non-expert users
- Multilevel security model with two levels
- We favor a defense-in-depth approach

Technical point of view
- Hypervisor (Qubes OS) vs. supervisor isolation (CLIP OS)
- CLIP OS: Limited access rights and capabilities, even for administrators
Security features

Goals

- High resistance to remote or local exploits
- Defense in depth: limit impact of successful exploits
- Limited options for attacker persistence

Challenges

- Mobility / road warrior / remote worker use case
- Multi-level isolation and hardware sharing
General architecture overview

- Cage 1 (Debian)
- Cage 2 (Gentoo)
- Cage 3 (...)

Core (Gentoo Hardened)

Linux kernel

Hardware

Enforced isolation — Controlled interaction
Defense in depth

Concepts
- Minimal attack surface
- Isolation based on containers

Implementation
- All services confined in Linux "containers"

v4
- Additional isolation using Linux-VServer
- Specific Linux Security Module (CLIP-LSM) & capability split

v5
- Linux-VServer like LSM (early development stage)
- Landlock³ (planned)

³See landlock.io
Network level isolation

Low level

App

App

Virtual interface

High level

App

App

Virtual interface

Core

Physical interface

Routing

Routing + IPsec

"Cleartext" traffic

Encrypted traffic
Application hardening and exploit mitigation

Memory-unsafe programming languages (C, C++, etc.)
Root cause of most major vulnerabilities in the last 10+ years\(^4\)

Mitigation
- Built from source with compile-time hardening (Gentoo Hardened)
- v4: PaX (part of grsecurity): strict $W \oplus X$ for memory allocations

Long term solution
- Use only memory safe languages (Rust, OCaml, etc.)
- v4 & planned for v5: PKCS#11 proxy written in OCaml (Caml Crush\(^5\))
- v5: Updater written in Rust (in progress)

\(^4\) https://www.zdnet.com/.../microsoft-70-percent-of-all-security-bugs-are-memory-safety-issues/
\(^5\) https://github.com/caml-pkcs11/caml-crush
# Linux kernel and system hardening

## Goals
- Protect the kernel from itself and from userspace
- Provide good defaults for userspace applications

## Implementation
- Strict kernel build time configuration
- Per hardware curated profiles (modules, firmwares, etc.)
- Paranoid command line (IOMMU, PTI, etc.)
- Strict sysctl defaults (kptr_restrict, ptrace_scope, etc.)

## Additional changes
- v4: grsecurity
- v5: STACKLEAK (now upstream), linux-hardened, Lockdown
No arbitrary code execution: $W \oplus X$

Goal

Defense in depth and difficulty for an attacker to persist post compromise.

Implementation

- User partitions always mounted as RW and noexec
- Multiple partitions to allow RO + exec and RW + noexec mounts

v4

- System partitions mounted as RW + exec to apply updates during boot
- Then remounted as RO + exec once boot is completed

v5

- Stricter split between system and configuration partitions
- RO and exec: system executables, configuration and data
- RW and noexec: runtime configuration, logs, user and application data
Read-write filesystem (e.g. /home/user)

- prog.py

Read-only filesystem (e.g. /usr/bin)

- Python interpreter
- prog.py

Linux kernel

open(prog.py, O_MAYEXEC)

-v4 & planned for v5-

Kernel support currently in progress upstream⁶

---

Updates

Goals

◮ Unattended, automatic and in the background updates
◮ User-controlled rollback at boot time

Implementation

◮ Signed packages (v4) & images (v5) transmitted over HTTPS over IPsec
◮ v4: Installed at boot time for the core / runtime for GUI environments
◮ v5: Installed in background and effective on reboot
Full boot chain integrity guarantee (v5)

Goal

Guarantee full system integrity even in the event of a system compromise

- Will only boot if the system’s integrity can be cryptographically verified
- Based on UEFI Secure Boot feature:
  - Signed bootloader, initramfs, Linux kernel and its command line
  - Read-only system partition (Squashfs) protected by DM-Verity (with forward error correction)
  - Custom keys (i.e. not signed by Microsoft, requires enrollment in hardware)
Password-less encrypted partitions (v5)

Implementation

- Automatic secret sealing & unsealing with a TPM 2.0
- Based on boot chain integrity measurements
Project status (v5)

- First alpha release in September 2018
- Now close to beta release
- Current use-case: server & virtualization (no graphical user interface)
Roadmap: 5.0 Beta

Completed

- "Unprivileged" admin, audit and update roles
- SSH server (for audit, admin and debug)

In progress

- Client for automatic updates
- Confined IPsec client
- Basic network (DHCP, static IP) and firewall (static rules) support
Roadmap: 5.0 stable

Planned

- Confined user environments (GUI)
- Multilevel support (Linux-VServer like LSM)
- Automated installation using PXE
- etc.

ANSSI CLIP OS: Building a defense-in-depth OS
Remaining challenges

Hardware sharing

- Workarounds available for audio, video, smartcards
- Partial solution for USB devices
- Safe access to filesystems on USB devices?
- Safe USB devices? (see WooKey project\(^7\))

Application confinement

- Flatpak (planned for v5)

---

\(^7\)https://github.com/wookey-project
Conclusion

**Pragmatic approach**
- Defense in depth instead of single strong barrier
- Properly configured system: safe by default

**Built to be reusable for multiple use cases**
- May need some adaptation work for integration into an IT infrastructure

**Open source project**
- Sources: [https://github.com/CLIPOS](https://github.com/CLIPOS)
- Bugs: [https://github.com/CLIPOS/bugs](https://github.com/CLIPOS/bugs)
- Documentation: [https://docs.clip-os.org](https://docs.clip-os.org)
- Code review: [https://review.clip-os.org](https://review.clip-os.org)
Thanks!

✉️ clipos@ssi.gouv.fr
🌐 Website: clip-os.org
🌐 Docs: docs.clip-os.org
🌐 Sources: github.com/CLIPOS
🌐 Bugs: github.com/CLIPOS/bugs