LINUX SECURITY SUMMIT

# Where do Security and Safety Meet?

Elana Copperman, PhD

elana.copperman@mobileye.com

THE LINUX FOUNDATION

# Agenda

- Safety vs Security

  Where they meet / Where they don't meet
- Engineering foundations for security / safety
  - Memory protection features
  - Isolation techniques and FFI (Freedom from Interference)
  - Timing and execution
  - ebpf and profiling
  - Safety extensions to Linux drivers
- Practical considerations and ELISA

# WhoamI?

- System Safety Architect, Mobileye (part of Intel)
- Supports design of safety features in Mobileye products, including system boot; drivers; and Linux infrastructure.
- Before working at Mobileye, worked as a Security Architect for Cisco-II (formerly NDS) and more recently as a security consultant for major European automotive concerns on behalf of various Israeli start-ups.
- Research interests focus on software engineering methodologies and security engineering.

# Safety vs Security

- Functional safety

*The objective of functional safety is freedom from unacceptable risk of physical injury or damage to the health of people either directly or indirectly … by the proper implementation of one or more automatic protection functions (often called safety functions). A safety system … consists of one or more safety functions.*

- Liability → Certification:
  - ISO61508 standard, General standard for electrically-based safety systems
  - ISO13849 standard, Safety-related parts of control systems

# diff

| Security | | Safety |
|---|---|---|
| Malicious intent | vs | Failures (systematic/software, transient/hardware) |
| Block attack (hacker) | vs | Ensure freedom from unacceptable risk |
| Vulnerabilities (weakness which can be exploited by an attacker) | vs | Faults (abnormal conditions which can cause failure) |
| Crypto (mathematical evidence) | vs | Mean Time Between Failures, Failure in Time (statistical) |
| Pentesting, fuzzing | vs | Failure Analysis |
| Open-source + proprietary tools | vs | Safety certification bodies/mistrust open-source |
| ISO/SAE21434 Road vehicles – Cybersecurity engineering | ??? | ISO26262 Functional Safety for Road Vehicles |

# Engineering foundations

- Focus on building safety / security in to the system
- *Investigate how to derive safety mechanisms based on security engineering*
- Identify Linux-based security features which are relevant for safety
- Example,  Freedom From Interference:

  *Absence of cascading failures between two or more elements that could lead to the violation of a safety requirement (ISO26262:1)*

  Translates to:

  *Linux process, access control, reduced privilege execution, container, hypervisor*

# Where do security and safety meet?

- Memory protection features
- “Freedom From Interference”
- Isolation techniques
- Timing and execution (multi-threaded systems)
- System profiling using ebpf-based tools
- Fault handling

*Safety ≠ Security, but they can meet in **code***

# Memory protection features

- Kernel configurations for safety, derived from configs commonly set for security – *draft*
- Map onto ISO26262, as well as security CWEs
- Breakdown into different memory types (e.g., heap/ stack)
- Layman's description, implementation guidelines, runtime/ performance impact
- Identify configs which are potentially relevant for safety

# Where security meets safety

- Disable CONFIG_DEVKMEM
- Enable CONFIG_FORTIFY_SOURCE
- Disable CONFIG_PROC_KCORE
- Enable CONFIG_STRICT_KERNEL_RWX
- Enable CONFIG_THREAD_INFO_IN_TASK

*No safety claims are made, integrator is responsible to map onto the safety claims for a specific use case.*

# Where security does *not* meet safety

- Disable CONFIG_DEVMEM (*refactor code*)
- Enable CONFIG_ELF_CORE (*traceability*)
  Enable CONFIG_STACK_TRACER (*traceability*)
  Enable CONFIG_PROC_PAGE_MONITOR (*traceability*)
- Enable CONFIG_HIBERNATION (*safe state on panic*)
  Enable CONFIG_EXEC (*safe state on panic*)

# Freedom From Interference (FFI)

- ISO26262:6, Annex D:
  - Timing and execution: blocking of execution, deadlocks, livelocks, incorrect allocation of execution time, incorrect synchronization between software elements
  - Memory: corruption of content, inconsistent data, stack overflow or underflow, read or write access to memory allocated to another software element
  - Exchange of information: repetition/loss/delay/insertion/incorrect addressing/incorrect sequencing/corruption of information, asymmetric information sent from a sender to multiple receivers, information from a sender received by only a subset of receivers, blocking access to a communication channel

# Isolation techniques

- Common goals for isolation: Limit access to resources by a Linux process → reduced privilege execution, FFI
- Safety architecture to separate memory space allocated to software elements with different levels of safety criticality
- Prudent use of basic Linux features such as namespaces, cgroups, kernel capabilities.
- Well defined configuration (e.g., systemd unit files) which are the basis for safety claims.

# Timing and execution

- Kernel configurations (primarily off-line testing)
  - Enable CONFIG_SOFTLOCKUP_DETECTOR
  - Enable CONFIG_DEBUG_SPINLOCK
  - Enable CONFIG_WQ_WATCHDOG
  - Enable CONFIG_RCU_TORTURE_TEST
- Dynamic analysis for multi-threaded systems
  - Enable CONFIG_KCSAN
  - TSAN – Thread Sanitizer

# ebpf

- ebpf and security – *established*
  - User space vs kernel space
  - ebpf verifier
- ebpf and safety - *TBD*
- Tracing and profiling: perf command, perf-tools, bpftrace, bcc, new stuff, …
- *epbf verifier as a model for safety run-time monitoring*
- xdp, avoiding the network stack

# Fault handling extensions to Linux drivers

- Fault handling:  detection, correction
- Focus on hardware / software interface
- Advanced Error Reporting (AER), PCIe infrastructure
- Capture errors, regardless of root cause (malicious, systematic, transient)
- *Collaboration with hardware vendors → built-in safety mechanisms in drivers, open-source infrastructure*

# Practical considerations

- Less relevant:
  - SELinux policies
  - seccomp
  - Hypervisor

*Challenge:*

*LSaMs = Linux Safety Modules, open-source building blocks*

# ELISA – Enabling Linux in Safety-critical Applications

As defined by the ELISA charter, *"the mission of the Project is to define and maintain a common set of elements, processes and tools that can be incorporated into Linux-based, safety-critical systems amenable to safety certification."*

- Ongoing work in ELISA currently focused on helping companies to demonstrate that a specific Linux-based system meets necessary safety requirements for certification.
- **Invitation to designers, architects, developers and validation experts who produce such systems and wish to contribute.**
- Demonstrate use of features in real systems.
- Propose enhancements / kernel patches to help make those features more amenable for use in safety-critical systems, collaborating with other Work Groups.

elana.copperman@mobileye.com

THE LINUX FOUNDATION

LINUX
SECURITY
SUMMIT