Intelligent Vehicle Security Needs a Verified Operating System

Gernot Heiser
UNSW and seL4 Foundation
gernot@unsw.edu.au
Car Hacking Danger Is Likely Closer Than You Think

A Detroit Free Press report shows there were 150 automotive cybersecurity incidents in 2019 alone.

By Sebastian Blanco  Published: Sep 4, 2021

Vulnerabilities
Car Hacking Is Real. Here's How Manufacturers Can Combat It

Sophisticated cars offer convenience for drivers but opportunities for hackers.

Diego Pena
Head of Content

Hacker Liberates Hyundai Head Unit, Writes Custom Apps

July 18, 2023 by Anja Verrova

© 2023 Gernot Heiser
Traditional Cars Are Exposed

Networking for:
- Entertainment
- Driver information
- Safety (tyre pressure...)
- Maintenance (OTA upgrades)

No security whatsoever on CAN bus!

pwned!
Intelligent Vehicles: Hacker’s Paradise!

Attacks

- Lidar
- Camera
- GPS

"Intelligence"

- Engine Control
- Steering
- Breaks

Highly complex!

Attack by:
- spoofing signals
- exploiting bugs
Who Cares?

- Connected cars are great, until they're not. A recent Detroit Free Press article shows that vehicle hacks are more common and more dangerous that most people realize.
- There were at least 150 automotive cybersecurity incidents in 2019, part of a 94 percent year-over-year increase since 2016, according to a report from Upstream Security.
- Oh, and here's a phrase we're loath to see, even though we're likely to come across it plenty more in the future: ransomware for cars.

CAR AND DRIVER

Published: Sep 4, 2021

of moving vehicles. But as Justin Cappos (the computer science researcher at New York University) told The Times, the potential threats are even worse than anything we've seen yet:

“If there was a war or escalation with a country with strong cyber-capability, I would be very afraid of hacking of vehicles,” Cappos says. “Once in, hackers can send messages to the brakes and shut off the power steering and lock people in the car and do other things that you wouldn’t want to happen.”

Diego Poza
Head of Content

Last Updated On: December 21, 2020
How Can We Protect Intelligent Vehicles?

Requires trustworthy isolation!

Note: Hugely over-simplified!

Sensor → Firewall/packet filter → Runtime Monitor → “Intelligence” → Engine Control, Steering, Breaks

Requires trustworthy isolation!
Foundation for Truly Secure Systems

- Comprehensive formal verification
- Capabilities for fine-grained protection
- World’s fastest microkernel

Present limitations
- initialisation code not verified
- MMU, caches modelled abstractly
- Multicore not yet verified
Capabilities: Fine-Grained Protection

- Enforce *least privilege*
- No communication unless explicitly authorised!
Made For Real-World Use

Autonomous vehicles

Secure communication device
In use in multiple defence forces

Satellites

Laot: Critical infrastructure protection
DARPA HACMS: Incremental Cyber Retrofit

Original Mission Computer

Trusted
Mission Manager
Crypto
Local NW
Ground Stn Link
Linux

Trusted
Mission Manager
Crypto
Local NW
Ground Stn Link
Linux

Trusted
Mission Manager
GS Lk
Mission Mgr
Crypto
GPS
Linux

Camera
Linux
VMM
VMM
DARPA HACMS: Incremental Cyber Retrofit

Original Mission Computer

Trusted
- GS Lk
- Mission Mgr
- Crypto
- GPS
- Camera
- Linux
- VMM

Linux

Local NW

Ground Stn Link

SSIV Keynote – Porto, PT – June'23

© 2023 Gernot Heiser
DARPA HACMS: Incremental Cyber Retrofit

Original Mission Computer

Trusted
- Mission Manager
- Crypto
- Local NW
- Ground Stn Link
- Linux

Cyber-secure Mission Computer

Trusted
- Crypto
- Mission Mngr
- Local NW
- Comms
- GPS

Camera
- Linux
- VMM

[Klein et al, CACM, Oct’18]
World’s Most Secure Drone

We brought a hackable quadcopter with defenses built on our HACMS program to @defcon #AerospaceVillage. As program manager @raymondrichards reports, many attempts to breakthrough were made but none were successful. Formal methods FTW!
seL4 Needs an OS
Microkernel Is Not An OS

Modularisation: Separate components
- operating-system services
- device drivers
- applications

Microkernel enforces isolation – bullet-proof
- kernel code reduced to minimum
- mediates hardware resources

OS

Resource Mgmt | File System | Networking | Device Driver | Trusted Component | Untrusted Component

Virtual Machine
- Linux App
- Linux

User Mode

Kernel Mode

Hardware
Build a performant OS from Scratch?

Yes – if we strictly observe some fundamental principles: KISS

• Fine-grained modularity, strong separation of concerns
• Least privilege
• *Radical Simplicity™*: provide only the features needed
• Swappable, *use-case specific policy* (rather than universal policy)

Also limit scope:
• Cyberphysical systems
• IoT systems

“Universal” policies are complex, always have pathological cases
OS Framework: The seL4 Core Platform

- Simple, single-threaded event-driven
- Minimal abstractions
- Thin wrapper of seL4
- Encourage “correct” use of seL4 primitives
- Aimed at IoT/cyberphysical

Protection Domain 1:
- init(...)
- notified(...)
- protected(...)

Protection Domain 2:
- init(...)
- notified(...)

Communication Cannel:
- notify(...) -> Protected Procedure Call

Memory Region

May be a virtual machine
Apply KISS Principles to OS

- Large number of extremely simple components
- Zero-copy, lock-free, asynchronous shared-memory communication
- Broadcasts handled separately
- IP stack is client library
- Copy where needed for security only
- Multiplexers share device between clients
- HW-specific interface
- HW-independent interface

Example: Networking

Device driver only translates HW-specific to HW-independent interface
Legacy Drivers?

Can use Linux drivers wrapped into individual driver VM
Can This Work?
Comparison to Linux

**Linux:**
- NW driver: 4k lines
- NW system total: 1M lines

**KISS design:**
- NW driver: 700 lines
- MUX: 400 lines
- Copier: 200 lines
- IP stack: much simpler, client library
- Shared NW system total: < 2,000 lines

Written by second-year student!

Performance?
Evaluation Setup

2 context switches per packet

10 context switches per packet
Achieved Performance

- Gigabit Ethernet
- Single core

Simplicity wins!

Bigger is better

Smaller is better

Core take-away: We can build a performant OS this way!
Can We Verify It?
seL4CP Verification

Conditions apply

CapDL spec

SDF Spec

Translation-validation (push-button proof)

Compiler/Linker

system.elf

init.o

PD1.c PD2.c libsel4cp.c

sel4 spec

sel4CP spec

Push-button proof

PD0 PD1
seL4CP Verification: libsel4cp

C parser (Isabelle)

SimplExport (Isabelle)

seL4CP C implementation

Implementation formalisation

Control-flow graph

seL4CP Spec

seL4 Abstract Spec Projection

Tool

SMT Solver
s4L4CP Verification in Context

**Linux:**
- NW driver: 4k lines
- NW system total: 1M lines

**KISS design:**
- NW driver: 700 lines
- MUX: 400 lines
- Copier: 200 lines
- IP stack: much simpler, client library
- shared NW system total: < 2,000 lines

**seL4CP:**
- libsel4cp: 280 lines
seL4CP Verification
Stepping Back: sDDF and CP Verification

**sDDF demonstrates:**
- A highly modular design is possible and can perform well!
- Design enables building OS from scratch
- Simplicity wins – KISS!

**seL4CP verification demonstrates:**
- Small, simple modules can be verified using push-button techniques!
- A KISS-based design should be verifiable

**Plan:**
- **2023**: OS with networking & file system
- **2024**: verified core OS components
Security is no excuse for bad performance!