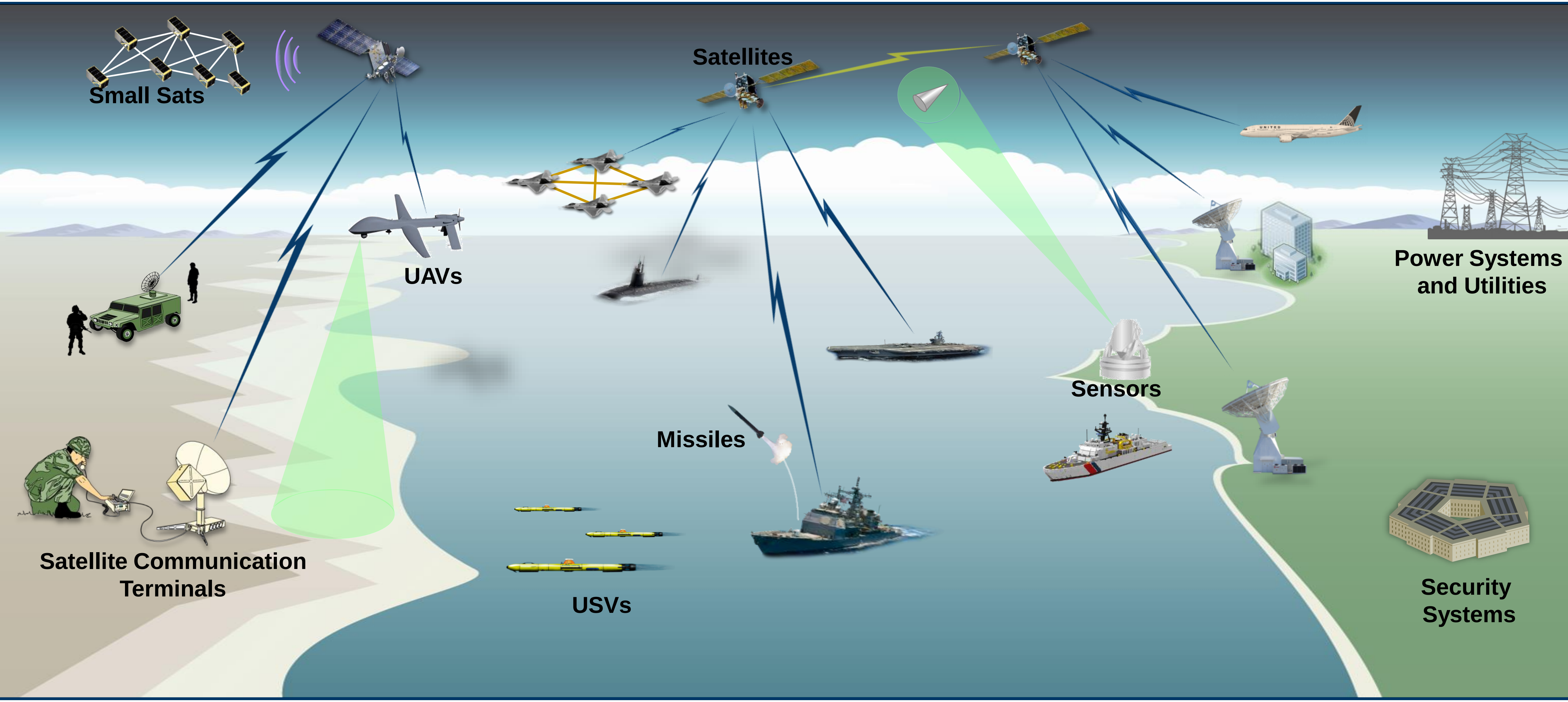
Magnetite:
Rust-Based OS Services for seL4

Juliana Furgala
Secure and Resilient Systems and Technologies Group

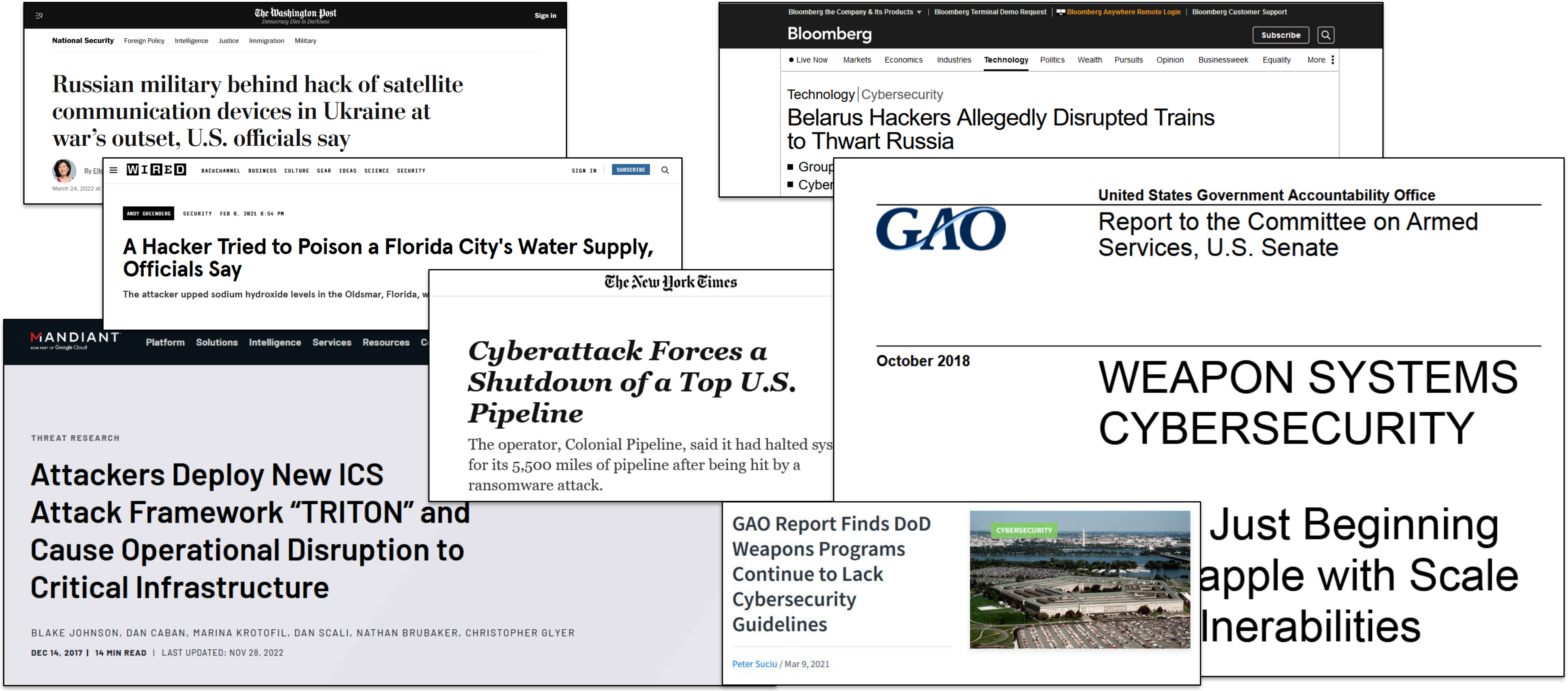
LINCOLN LABORATORY
MASSACHUSETTS INSTITUTE OF TECHNOLOGY

Embedded Systems are Everywhere


Small Sats
Satellites
UAVs
Power Systems and Utilities
Sensors
Missiles
Satellite Communication Terminals
USVs
Security Systems

UAV = Unmanned Aerial Vehicle
USV = Unmanned Submersible Vehicle

Trends in Security of Embedded Systems



The Washington Post
Democracy Dies in Darkness
Sign in

National Security | Foreign Policy | Intelligence | Justice | Immigration | Military

Russian military behind hack of satellite communication devices in Ukraine at war's outset, U.S. officials say

By Elle
March 24, 2022 at

WIRED
BACKCHANNEL BUSINESS CULTURE GEAR IDEAS SCIENCE SECURITY
SIGN IN | SUBSCRIBE

ANDY GREENBERG SECURITY FEB 8, 2021 6:54 PM

A Hacker Tried to Poison a Florida City's Water Supply, Officials Say

The attacker upped sodium hydroxide levels in the Oldsmar, Florida, w

Bloomberg the Company & Its Products | Bloomberg Terminal Demo Request | Bloomberg Anywhere Remote Login | Bloomberg Customer Support

Bloomberg
Subscribe

● Live Now | Markets | Economics | Industries | Technology | Politics | Wealth | Pursuits | Opinion | Businessweek | Equality | More

Technology | Cybersecurity

Belarus Hackers Allegedly Disrupted Trains to Thwart Russia

- Group
- Cyber

MANDIANT
NOW PART OF Google Cloud
Platform Solutions Intelligence Services Resources C

THREAT RESEARCH

Attackers Deploy New ICS Attack Framework "TRITON" and Cause Operational Disruption to Critical Infrastructure

BLAKE JOHNSON, DAN CABAN, MARINA KROTOFIL, DAN SCALI, NATHAN BRUBAKER, CHRISTOPHER GLYER

DEC 14, 2017 | 14 MIN READ | LAST UPDATED: NOV 28, 2022

The New York Times

Cyberattack Forces a Shutdown of a Top U.S. Pipeline

The operator, Colonial Pipeline, said it had halted sys for its 5,500 miles of pipeline after being hit by a ransomware attack.

United States Government Accountability Office

GAO

Report to the Committee on Armed Services, U.S. Senate

October 2018

WEAPON SYSTEMS CYBERSECURITY

Just Beginning apple with Scale lnerabilities

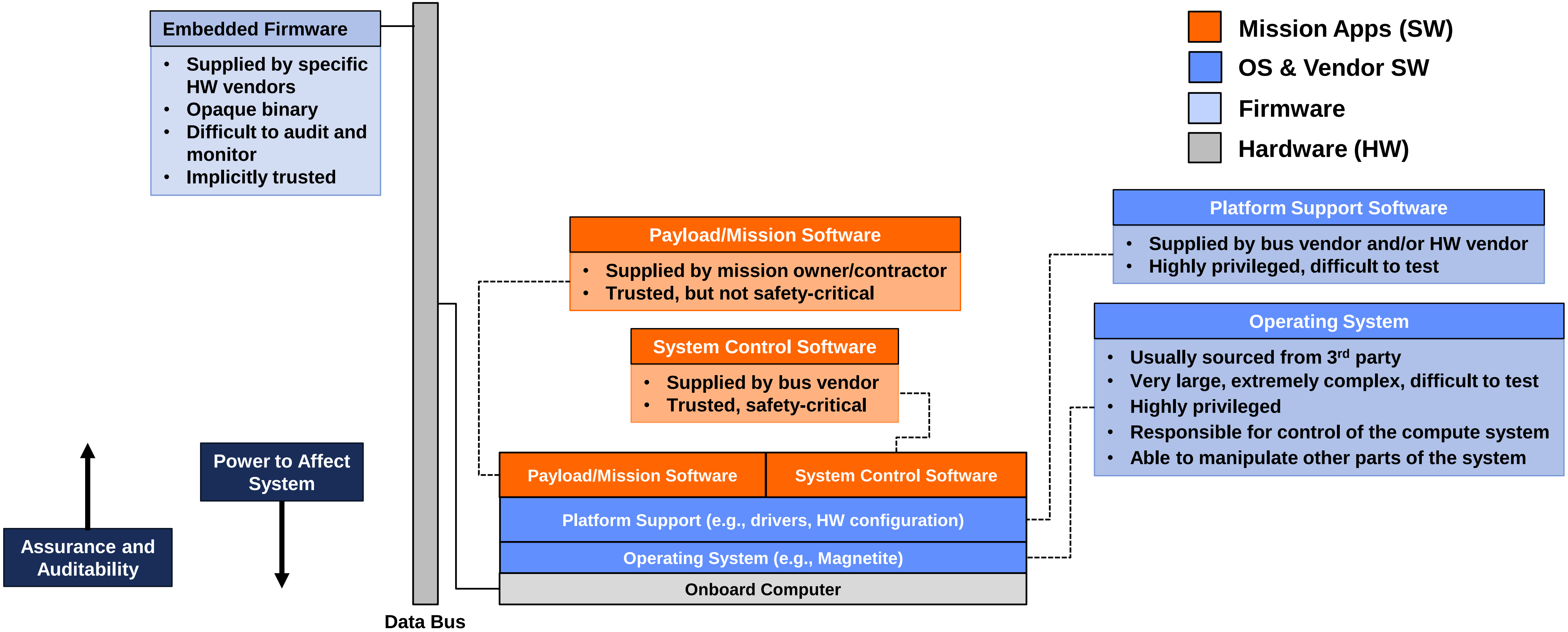
GAO Report Finds DoD Weapons Programs Continue to Lack Cybersecurity Guidelines

CYBERSECURITY

Peter Suciu / Mar 9, 2021

LINCOLN LABORATORY
MASSACHUSETTS INSTITUTE OF TECHNOLOGY

A Typical Embedded Device



🟧 Mission Apps (SW)
🟦 OS & Vendor SW
▢ Firmware
⬜ Hardware (HW)

Embedded Firmware

- Supplied by specific HW vendors
- Opaque binary
- Difficult to audit and monitor
- Implicitly trusted

Payload/Mission Software

- Supplied by mission owner/contractor
- Trusted, but not safety-critical

System Control Software

- Supplied by bus vendor
- Trusted, safety-critical

Platform Support Software

- Supplied by bus vendor and/or HW vendor
- Highly privileged, difficult to test

Operating System

- Usually sourced from 3rd party
- Very large, extremely complex, difficult to test
- Highly privileged
- Responsible for control of the compute system
- Able to manipulate other parts of the system

Payload/Mission Software
System Control Software
Platform Support (e.g., drivers, HW configuration)
Operating System (e.g., Magnetite)
Onboard Computer
Data Bus
Assurance and Auditability
Power to Affect System

A trustworthy operating system is essential for security

Operating System Challenge: Incorrectness

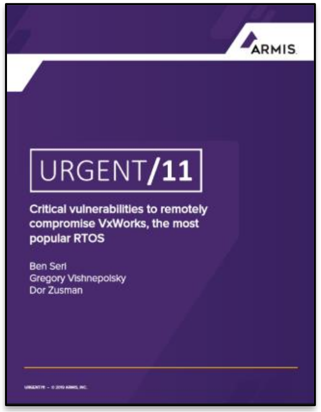
Size and complexity mean a high risk of bugs

- Millions of lines of code
- Hundreds of changes a day
- Huge amounts of functionality

Operating System	Lines of Code	Number of Contributors	Average Daily Commit Rate
FreeRTOS	5,000,000	60	8
RTEMS	2,000,000	200	7
Real Time Linux	26,000,000	20,000	159

Low-level languages means high risk of bugs

- Low-level languages without a runtime required for OS development
- Microsoft reports that 70% of its disclosed vulnerabilities are related to memory safety issues in C/C++
- Similar issues in other operating systems, like VxWorks



ARMIS

URGENT/11

Critical vulnerabilities to remotely compromise VxWorks, the most popular RTOS

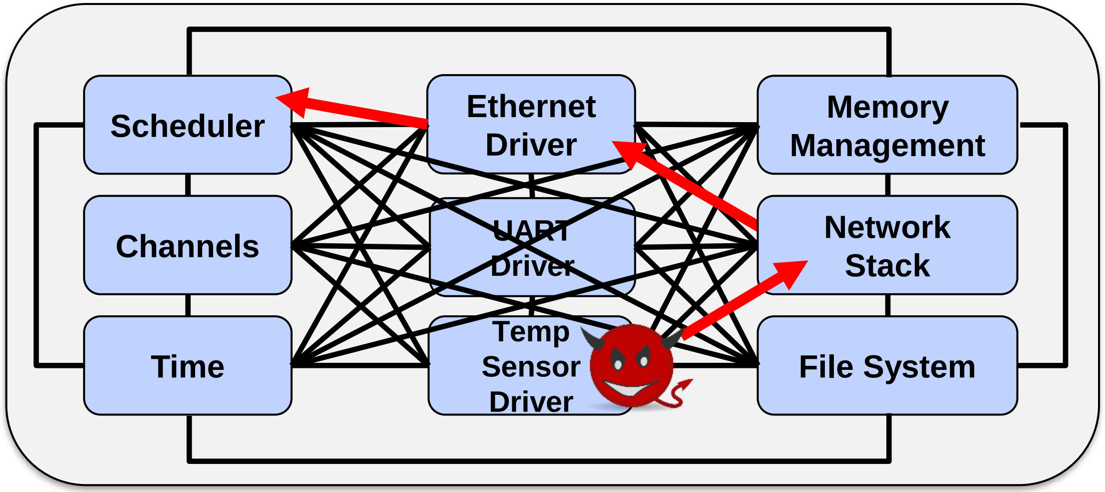
Ben Seri
Gregory Vishnepolsky
Dor Zusman

URGENT/11 – © 2019 ARMIS, INC.

Operating System Challenge: Handling Compromise

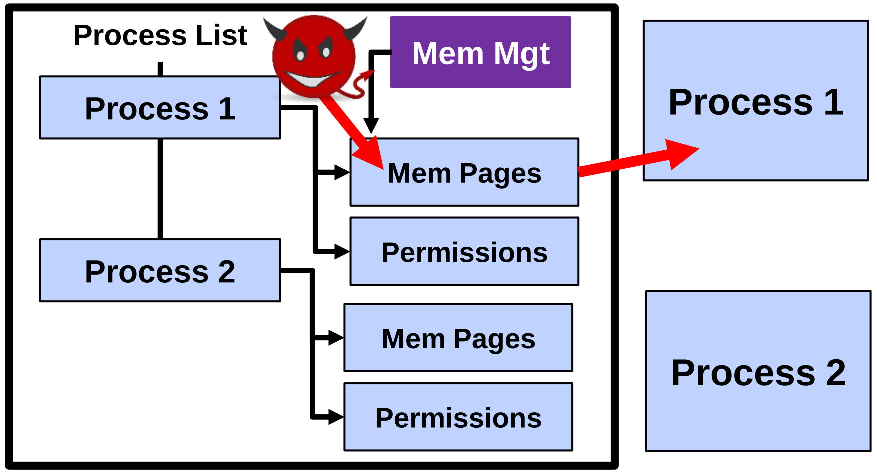
Monolithic design allows easy privilege escalation

- Most operating systems are monolithic
- All components can interact with all other components
- No real private data or functionality
- One point of compromise impacts entire operating system



Operating system components must be trusted forever

- Operating system maintains access to memory and permissions of applications
- Able to reach into applications and arbitrarily read/write
- Operating system can always compromise applications



Common Responses to Challenges

- **Patching**
  - **Fixes known bugs!**

Merely mitigates the problem, but cannot solve it

- **Manual and Automated Testing**
  - **Great for verifying functionality and conformance**

People are notoriously bad at creating test cases for malicious behavior

- **Fuzzing and Static Analysis**
  - **Finds lots of bugs, widely applied in practice**

Incomplete, limited analysis ability, bugs are still being discovered

- **Formal Methods**
  - **Provides extremely strong guarantees: “formal proofs”**

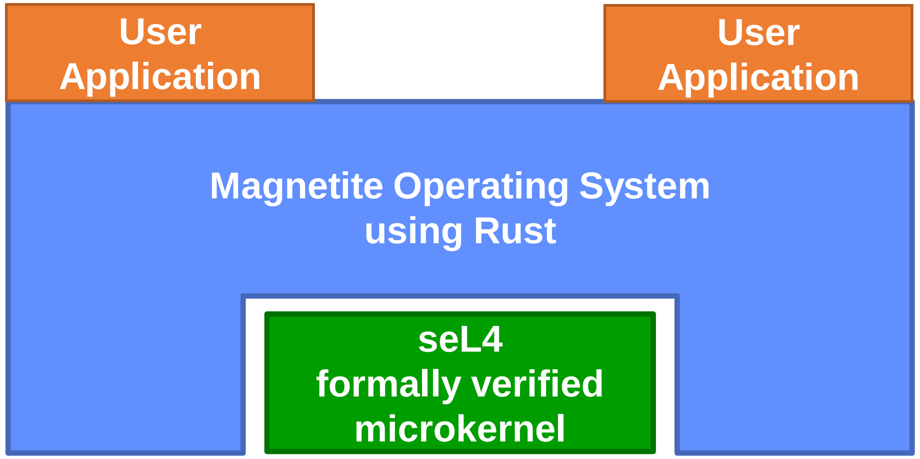
Extremely labor intensive and size limited

- **Microkernels and Compartmentalization**
  - **Reduces privilege escalation in the operating system**

Difficult to retrofit, existing systems are experimental, often poor performance

Magnetite: MIT LL Solution

- A new operating system
- Looks to the field of formal methods for a solid foundation
- Formally verified microkernel (seL4)
  - Provides isolation, scheduling, and resources
  - Careful design and usage to avoid performance impacts
- Leverage Rust’s language-level static analysis to reduce bugs
  - Provides memory safety at the language level
- Architected specifically for security
  - Minimize privilege, separate into components
  - Make it easy to reason about data flow


User Application
User Application
Magnetite Operating System using Rust
seL4 formally verified microkernel

Outline

- Motivation
- ⇨ Technical Foundations
- Magnetite Design and Status
- Applications of Magnetite
- Summary

seL4

- **Formally verified microkernel**

Functional Correctness | **Free From Memory Bugs** | **Binary Correctness** | **Data Integrity** | **Controlled Information Flow**

- **30-person years to verify**
- **~9KLOC**
- **Used by DARPA HACMS and AFRL ARES**

2

Comprehensive Formal Verification of an OS Microkernel

GERWIN KLEIN, JUNE ANDRONICK, KEVIN ELPHINSTONE, TOBY MURRAY, THOMAS SEWELL, RAFAL KOLANSKI, and GERNOT HEISER, NICTA and UNSW, Sydney, Australia

We present an in-depth coverage of the comprehensive machine-checked formal verification of seL4, a general-purpose operating system microkernel.

We discuss the kernel design we used to make its verification tractable. We then describe the functional correctness proof of the kernel's C implementation and we cover further steps that transform this result into a comprehensive formal verification of the kernel: a formally verified IPC fastpath, a proof that the binary code of the kernel correctly implements the C semantics, a proof of correct access-control enforcement, a proof of information-flow noninterference, a sound worst-case execution time analysis of the binary, and an automatic initialiser for user-level systems that connects kernel-level access-control enforcement with reasoning about system behaviour. We summarise these results and show how they integrate to form a coherent overall analysis, backed by machine-checked, end-to-end theorems.

The seL4 microkernel is currently not just the only general-purpose operating system kernel that is fully formally verified to this degree. It is also the only example of formal proof of this scale that is kept current as the requirements, design and implementation of the system evolve over almost a decade. We report on our experience in maintaining this evolving formally verified code base.

Categories and Subject Descriptors: D.2.4 [**Software Engineering**]: Software/Program Verification; D.4.5 [**Operating Systems**]: Reliability—*Verification*

General Terms: Verification; Security; Reliability

Additional Key Words and Phrases: seL4, Isabelle/HOL, operating systems, microkernel, L4

ACM Reference Format:
Gerwin Klein, June Andronick, Kevin Elphinstone, Toby Murray, Thomas Sewell, Rafal Kolanski, and Gernot Heiser. 2014. Comprehensive formal verification of an OS microkernel. *ACM Trans. Comput. Syst.* 32, 1, Article 2 (February 2014), 70 pages.
DOI: http://dx.doi.org/10.1145/2560537

1. INTRODUCTION

This article presents a detailed coverage of the comprehensive formal verification of the seL4 microkernel, from its initial functional correctness proof to more recent results, which extend the assurance argument up to higher-level security properties and down to the binary level of its implementation.

The target of our verification, the kernel, is the most critical part of a system, which is our motivation for starting system verification with this component. The customary definition of a kernel is the software that executes in the privileged mode of the hard-

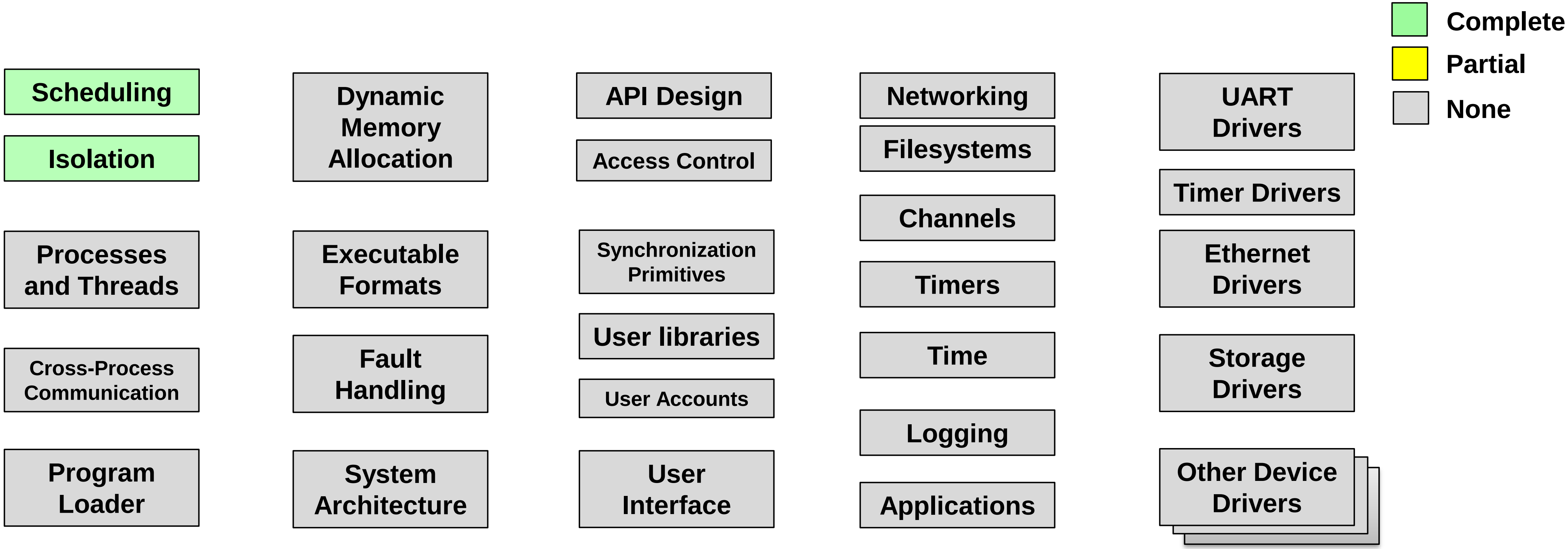
NICTA is funded by the Australian Government as represented by the Department of Broadband, Communications and the Digital Economy and the Australian Research Council through the ICT Centre of Excellence program.
Authors' address: NICTA, Level 4, 223 Anzac Pde, Sydney NSW 2052, Australia; Correspondence email: gernot@nicta.com.au.
Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies show this notice on the first page or initial screen of a display along with the full citation. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers, to redistribute to lists, or to use any component of this work in other works requires prior specific permission and/or a fee. Permissions may be requested from Publications Dept., ACM, Inc., 2 Penn Plaza, Suite 701, New York, NY 10121-0701 USA, fax +1 (212) 869-0481, or permissions@acm.org.
© 2014 ACM 0734-2071/2014/02-ART2 $15.00
DOI: http://dx.doi.org/10.1145/2560537

ACM Transactions on Computer Systems, Vol. 32, No. 1, Article 2, Publication date: February 2014.

Klein, Gerwin, et al. "Comprehensive formal verification of an OS microkernel", 2014

KLOC = thousand lines of code
HACMS = High Assurance Cyber Military Systems
ARES = Agile and Resilient Embedded Systems

seL4’s Role in an Full-Fledged OS



🟩 Complete

🟨 Partial

⬜ None

- 🟩 Scheduling
- 🟩 Isolation
- ⬜ Processes and Threads
- ⬜ Cross-Process Communication
- ⬜ Program Loader

- ⬜ Dynamic Memory Allocation
- ⬜ Executable Formats
- ⬜ Fault Handling
- ⬜ System Architecture

- ⬜ API Design
- ⬜ Access Control
- ⬜ Synchronization Primitives
- ⬜ User libraries
- ⬜ User Accounts
- ⬜ User Interface

- ⬜ Networking
- ⬜ Filesystems
- ⬜ Channels
- ⬜ Timers
- ⬜ Time
- ⬜ Logging
- ⬜ Applications

- ⬜ UART Drivers
- ⬜ Timer Drivers
- ⬜ Ethernet Drivers
- ⬜ Storage Drivers
- ⬜ Other Device Drivers

The design of operating system features is crucially important to system and application security

Rust

- **Programming language originally developed by Mozilla**
- **First new systems language in many years**
- **Now sponsored by an independent foundation and used by Mozilla, Amazon, Google, Microsoft, etc.**
- **Relies heavily on static analysis**
- **Features:**

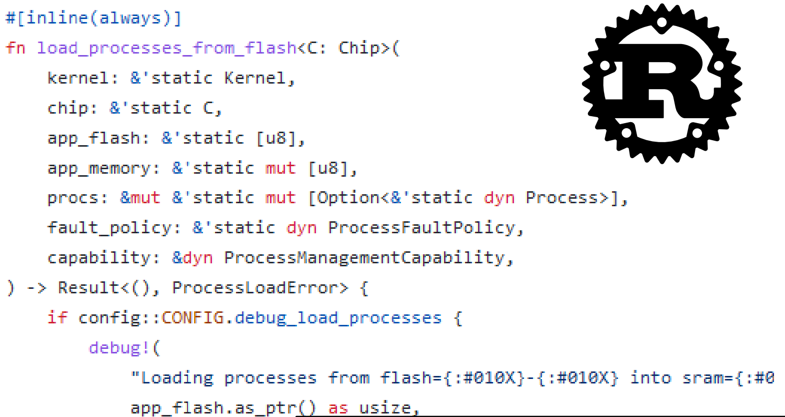
Memory Safety as Default

Bare Metal Support

Modern Package Management

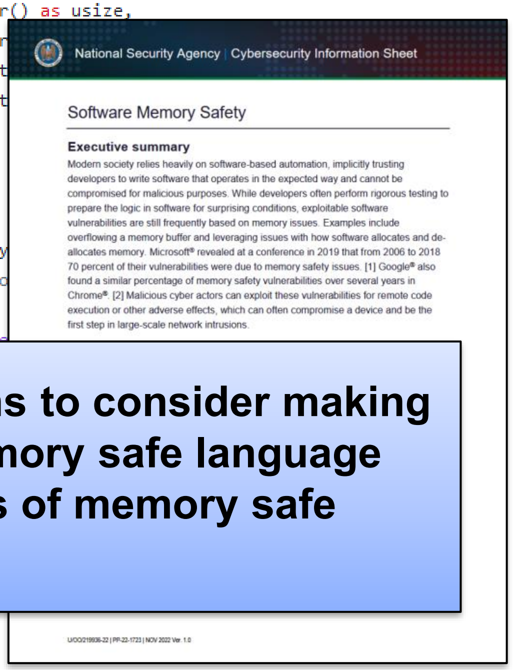
Interaction with Other Languages

C C++ Python



```
#[inline(always)]
fn load_processes_from_flash<C: Chip>(
    kernel: &'static Kernel,
    chip: &'static C,
    app_flash: &'static [u8],
    app_memory: &'static mut [u8],
    procs: &mut &'static mut [Option<&'static dyn Process>],
    fault_policy: &'static dyn ProcessFaultPolicy,
    capability: &dyn ProcessManagementCapability,
) -> Result<(), ProcessLoadError> {
    if config::CONFIG.debug_load_processes {
        debug!(
            "Loading processes from flash={:#010X}-{:#010X} into sram={:#0
            app_flash.as_ptr() as usize,
            app_flash.as_ptr
            app_memory.as_pt
            app_memory.as_pt
        );
    }

    let mut remaining_flash
    let mut remaining_memory
    // Try to discover up to
    let mut index = 0;
    let num_procs = procs.le
```



National Security Agency | Cybersecurity Information Sheet

Software Memory Safety

Executive summary

Modern society relies heavily on software-based automation, implicitly trusting developers to write software that operates in the expected way and cannot be compromised for malicious purposes. While developers often perform rigorous testing to prepare the logic in software for surprising conditions, exploitable software vulnerabilities are still frequently based on memory issues. Examples include overflowing a memory buffer and leveraging issues with how software allocates and de-allocates memory. Microsoft® revealed at a conference in 2019 that from 2006 to 2018 70 percent of their vulnerabilities were due to memory safety issues. [1] Google® also found a similar percentage of memory safety vulnerabilities over several years in Chrome®. [2] Malicious cyber actors can exploit these vulnerabilities for remote code execution or other adverse effects, which can often compromise a device and be the first step in large-scale network intrusions.

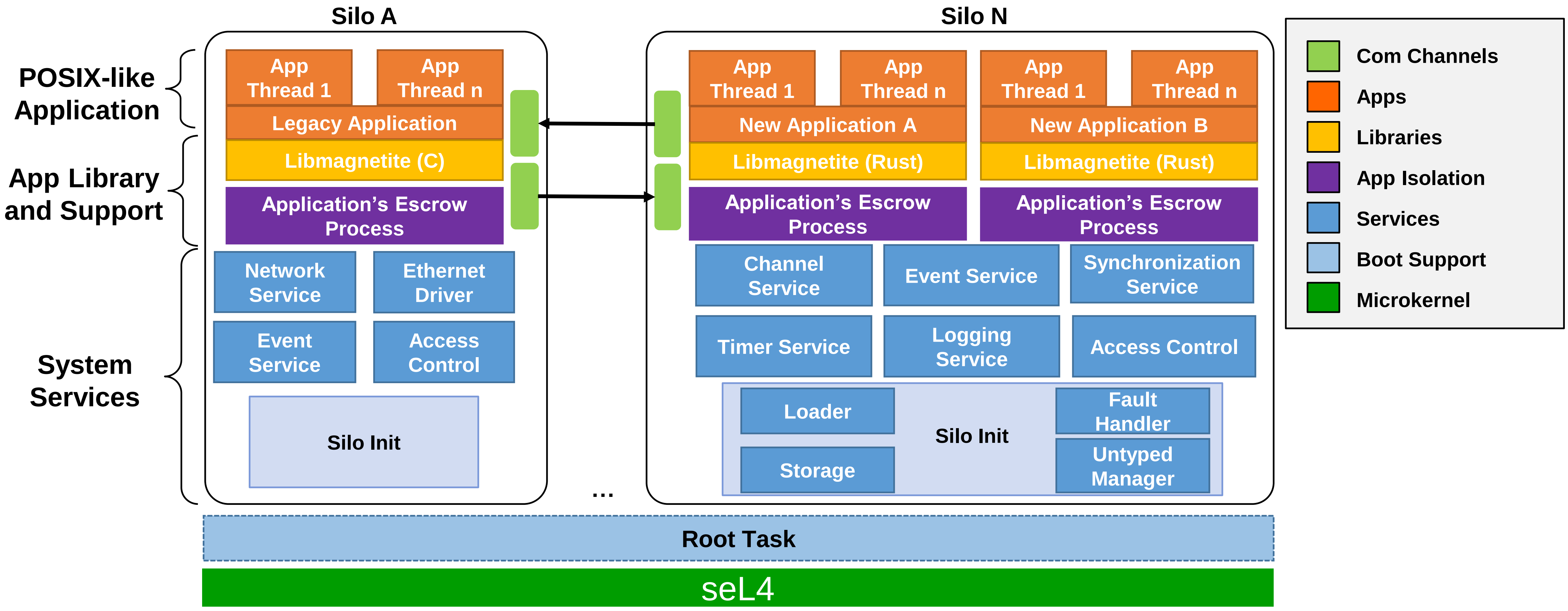
U/OO/219936-22 | PP-22-1723 | NOV 2022 Ver. 1.0

“NSA advises organizations to consider making a strategic shift… to a memory safe language when possible… Examples of memory safe language include… Rust”

Outline

- Motivation
- Technical Foundations
- Magnetite Design and Status
- Applications of Magnetite
- Summary

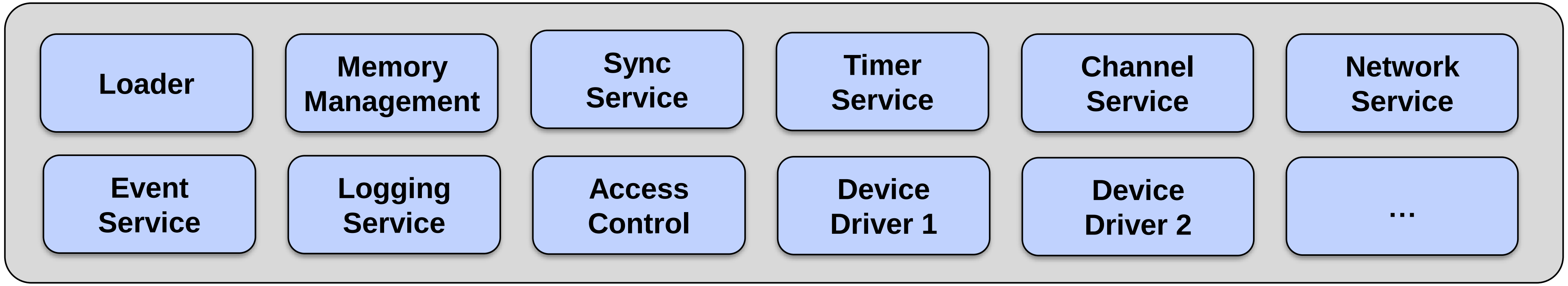
Magnetite Design



Feature: Separate Services

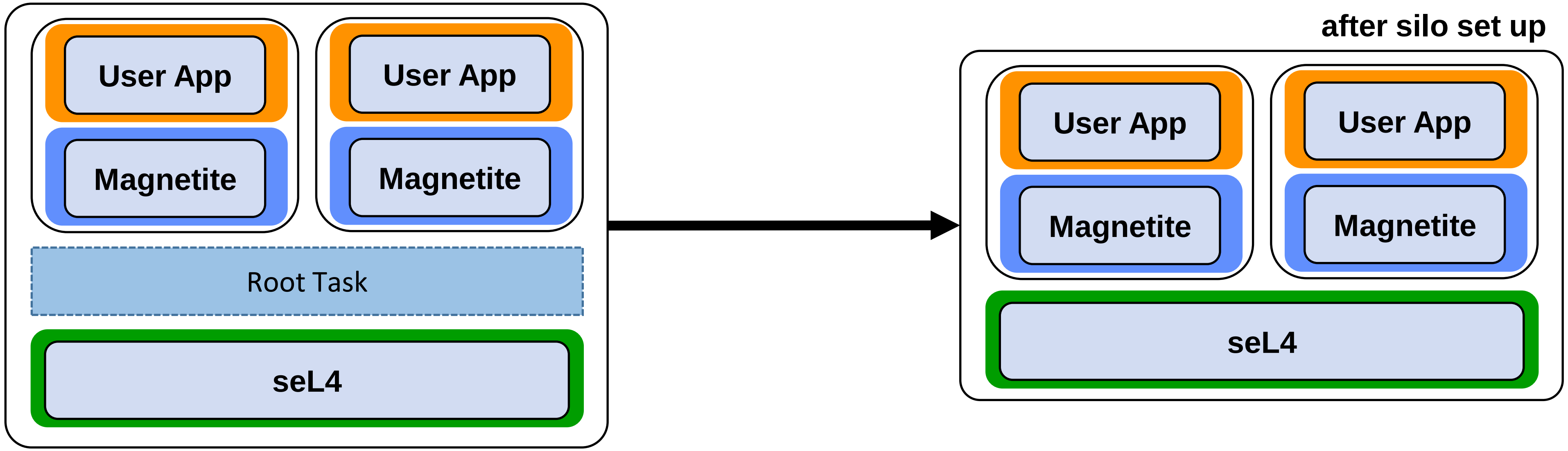
Magnetite's functionality is separated into multiple processes

- Usually considered a Good Thing for security
  - Reduced privilege escalation and compromise of unrelated functionality
- Challenges
  - Tends to result in complex communication patterns, overhead
  - Increase in message parsing, which is bug prone
- Solutions
  - Separate each type of functionality into its own service
  - Use auto-generated parsing code

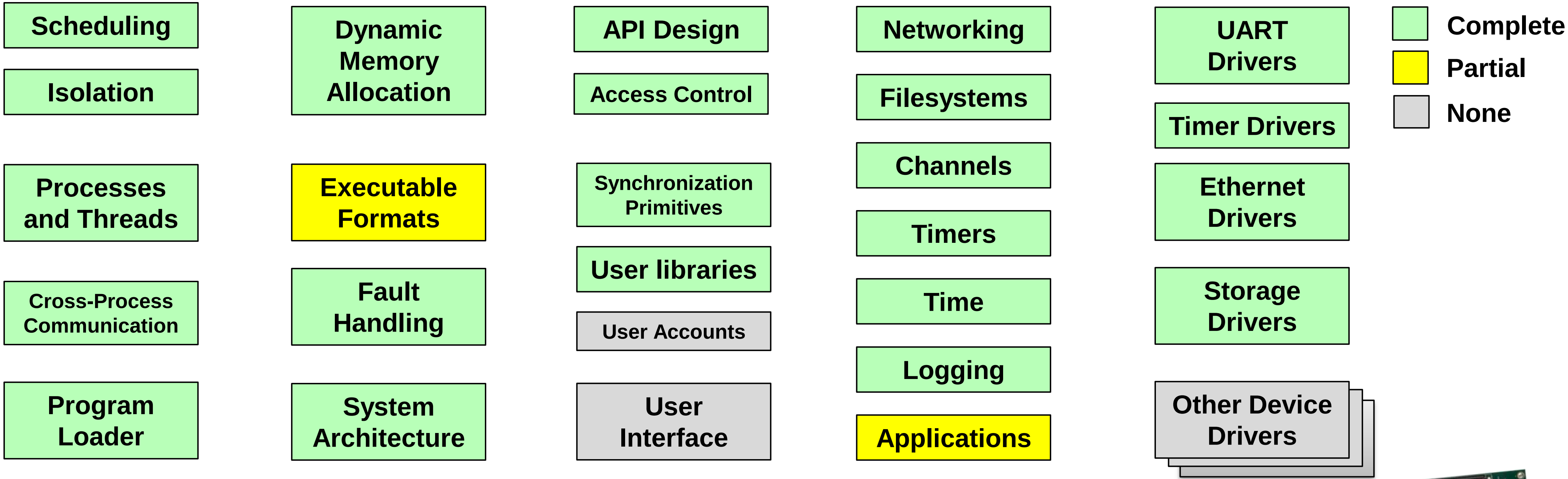


Feature: Bounded Data Flow

- Many missions have requirements on bounding data flow between parts of the system
- Magnetite provides “shared-nothing” silos of functionality, with explicit channels
  - “Shared-nothing” ensures that data cannot accidentally flow between silos
  - Explicit channels allow desired flows, which can be one-sided
  - Silos and channels are immutable after boot



Magnetite’s Current Status



- 🟩 Scheduling
- 🟩 Isolation
- 🟩 Processes and Threads
- 🟩 Cross-Process Communication
- 🟩 Program Loader
- 🟩 Dynamic Memory Allocation
- 🟨 Executable Formats
- 🟩 Fault Handling
- 🟩 System Architecture
- 🟩 API Design
- 🟩 Access Control
- 🟩 Synchronization Primitives
- 🟩 User libraries
- ⬜ User Accounts
- ⬜ User Interface
- 🟩 Networking
- 🟩 Filesystems
- 🟩 Channels
- 🟩 Timers
- 🟩 Time
- 🟩 Logging
- 🟨 Applications
- 🟩 UART Drivers
- 🟩 Timer Drivers
- 🟩 Ethernet Drivers
- 🟩 Storage Drivers
- ⬜ Other Device Drivers

🟩 Complete

🟨 Partial

⬜ None

Supported platforms:

– ARMv7a

– Xilinx ZC702 and TI DM3730

– Other platforms possible (ARMv8, RISC-V)

Operating System	Source Lines of Code (SLoC)
Magnetite	113,062
Real Time Linux	14,964,907

Today Magnetite is a mature system with solid basic OS functionality

Comparing Current Performance of Magnetite Against Common Alternative

Benchmarked on a Xilinx ZC702

Against Linux 5.4 with the realtime patch

	Average Case Performance (CPU Cycles)		Worst Case Performance (CPU Cycles)	
Benchmark	Real Time Linux	Magnetite	Real Time Linux	Magnetite
Locking a Contended Mutex	15,844	15,574	30,570	17,394
Timer Latency (POSIX)	20,666	12,202	33,118	13,907
Timer Latency (timerfd)	6,494	12,202	14,806	13,907
Channel Latency	9,439	18,367	22,671	20,038

Lower is Better

Lower is Better

Magnetite has a clear advantage over Linux for worst case performance, which is critical for embedded systems

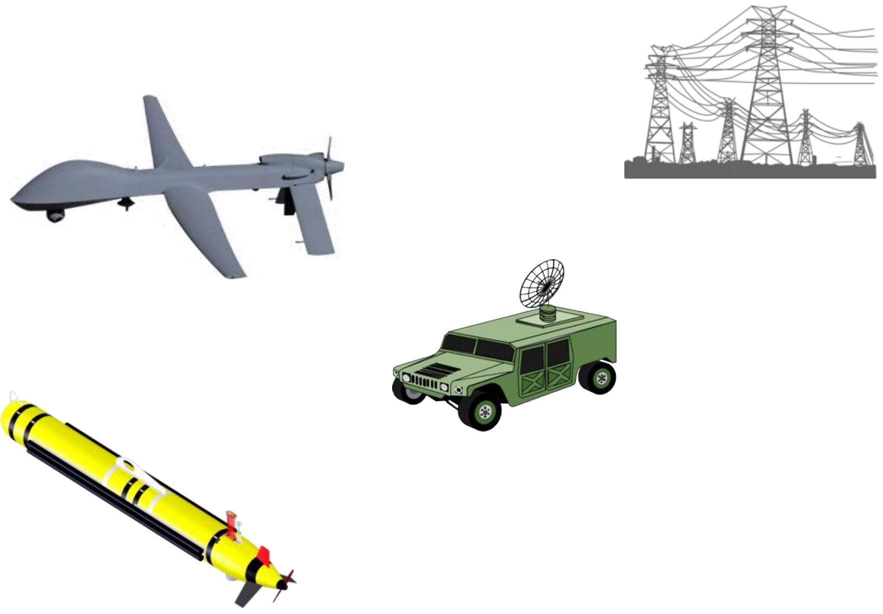
Outline

- Motivation
- Technical Foundations
- Magnetite Design and Status
- ➡ Applications of Magnetite
- Summary

Use Cases

Magnetite is very relevant for high-criticality embedded devices

- **Especially where:**
  - **Strong requirements on information flow exist**
  - **Isolation of components is critical**
  - **Performance is a requirement**
- **Possible Applications:**
  - **UxVs**
  - **Critical infrastructure**
  - **Hypervisors**
  - **Other high-criticality, embedded systems**



Summary

- MIT LL developed a novel operating system called Magnetite
  - Founded on formal methods and static analysis
  - Separates functionality into multiple processes to avoid impact of compromise
  - Enables control of information flow in a system
- Magnetite is a mature system
  - Demonstrates the possibilities of seL4 as the foundation for a secure OS
  - Continuing to mature through further technical development

Contact Information

Juliana Furgala

Juliana.Furgala@ll.mit.edu

MIT Lincoln Laboratory