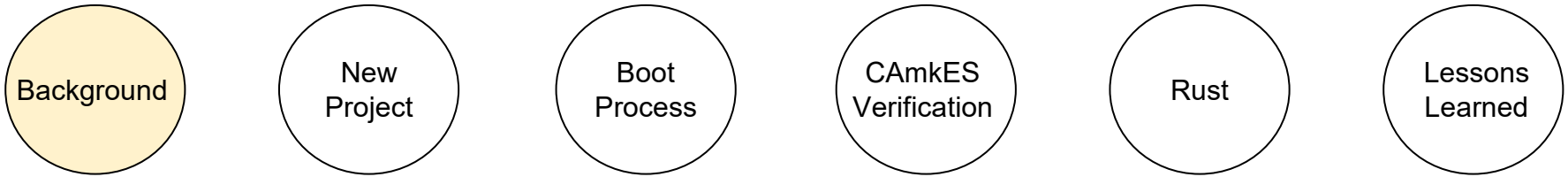
Formally Stepping Into The Unverified World

Sandy Frost
Advanced Research In Cyber Systems
Los Alamos National Laboratory

Sept. 19, 2023

LA-UR-23-30553

Overview


Background
New Project
Boot Process
CAmkES Verification
Rust
Lessons Learned

Background: Previous Project

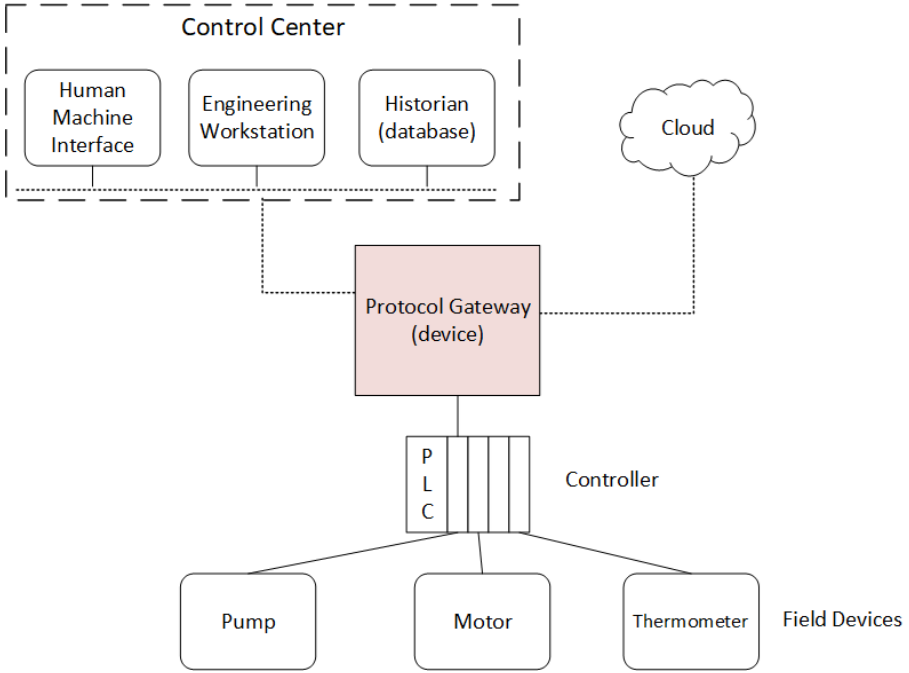
- Identify the benefits and limitations of **provable security approaches**, such as formal verification, that prove the security of protocol gateway converter implementations

- Perform a study that **compares provable security approaches to traditional cyber assessment methods** for protocol gateways, such as pentesting and binary analysis

- Determine a recommended mix of approaches to **achieve maximal security coverage** and **identify remaining security gaps** for new cybersecurity research thrusts

Background: SCADA Protocol Gateway

Protocol gateways convert messages between multiple protocols, such as messages coming from SCADA systems or industrial control centers and field devices (e.g. motors, pumps, actuators, sensors, etc.)



Protocols of Interest:

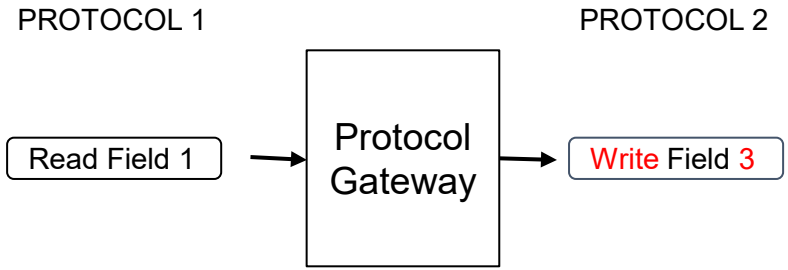
MODBUS, DNP3, BACNET

Motivation

Why choose a protocol gateway?

Motivation

Why choose a protocol gateway?



Software Shortcomings can Introduce Cyber Vulnerabilities

- Malformed messages or improper message translation
- Differences between the specification and implementation
- Memory safety checks are still vulnerable to control flow mainupation (e.g. buffer overflows allowing remote code execution)

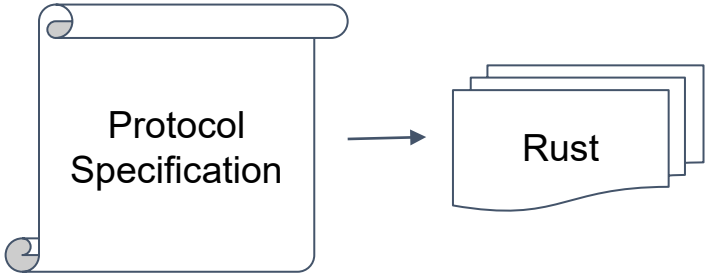
Resources for Vulnerabilities

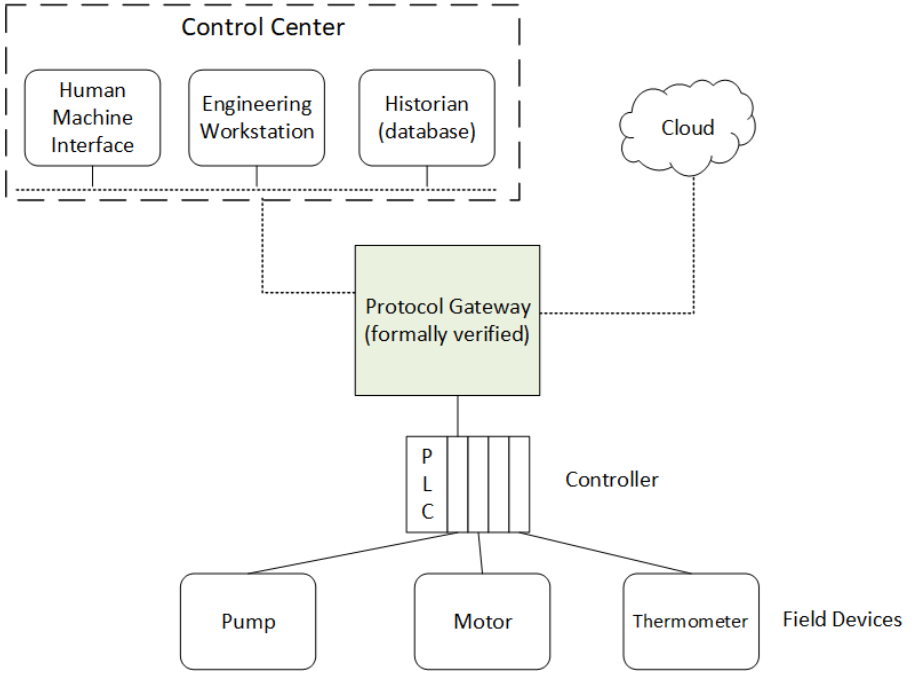
- MITRE Common Vulnerabilities and Exposures (CVE), DHS CISA ICS Advisories, vendor security reports, industry white papers*

*Trend MIcro “Lost in Translation: When Industrial Protocol Translation Goes Wrong”

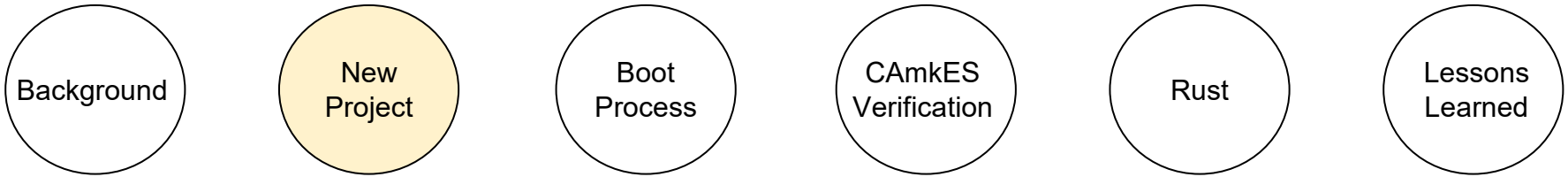
Background: SCADA Protocol Gateway

Protocol gateway formally verified implementation:



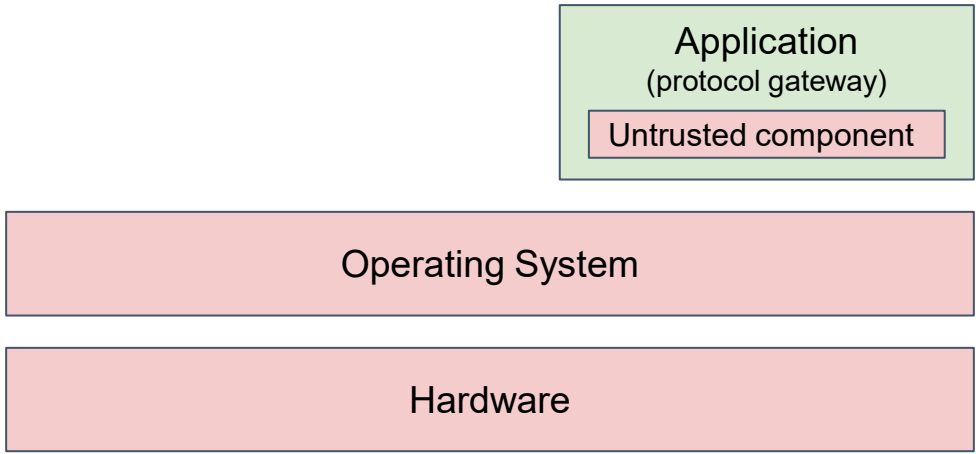


Overview


Background
New Project
Boot Process
CAmkES Verification
Rust
Lessons Learned

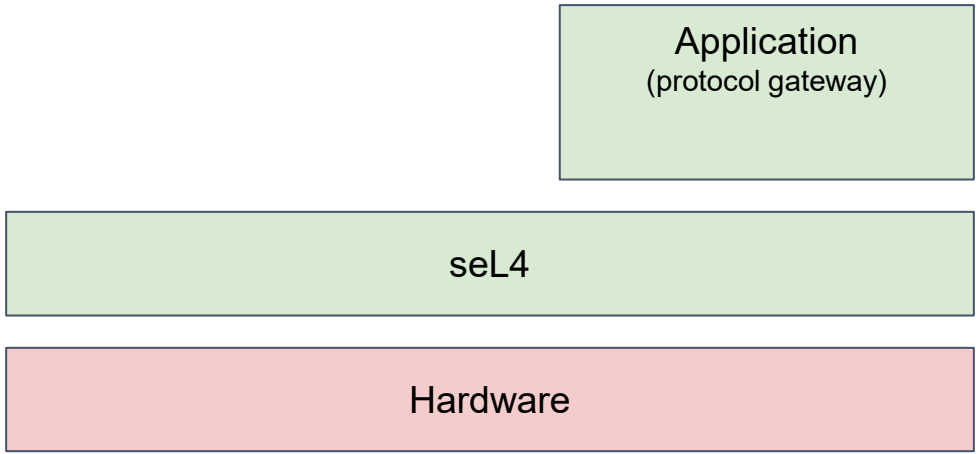
New Project - Expand the Security Envelope

Next step How can we expand the security envelope beyond our formally verified protocol gateway?



New Project - Expand the Security Envelope

Next step Interface our formally verified application (protocol converter) with a formally verified microkernel (seL4).



Assumptions

The seL4 Microkernel - An Introduction

- **Hardware behaves as expected.** This should be obvious. The kernel is at the mercy of the underlying hardware, and if the hardware is buggy (or worse, has Trojans), then all bets are off, whether you are running verified seL4 or any unverified OS. Verifying hardware is outside the scope of seL4 (and the competency of Trustworthy Systems); other people are working on that.

The Proof

- **Hardware:** we assume the hardware works correctly. In practice, this means the hardware is assumed not to be tampered with, and working according to specification. It also means, it must be run within its operating conditions.

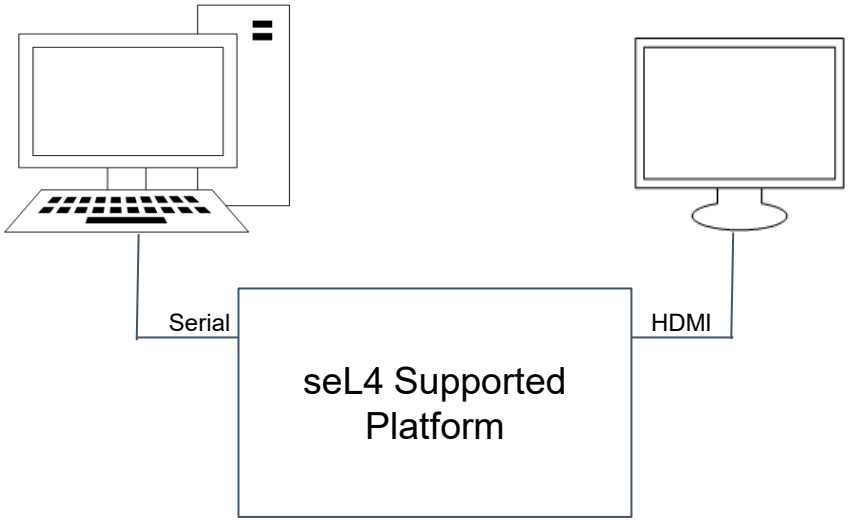
https://sel4.systems/About/seL4-whitepaper.pdf, https://sel4.systems/Info/FAQ/proof.pml,

Questions

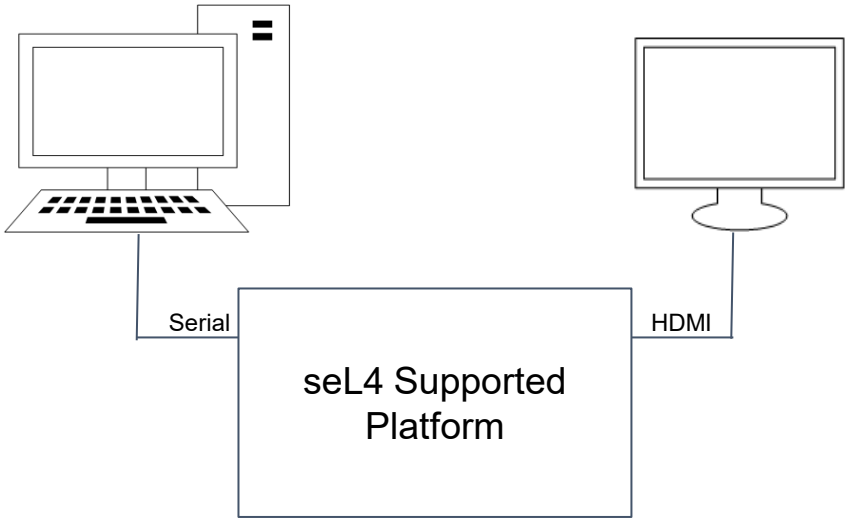
- What are the security impacts of interactions between formally verified and unverified components?
- What is the overhead of developing, deploying and integrating seL4 into existing systems?
- Can the security guarantees be expanded to user space programs without using a virtual machine?
  - Can seL4 run safer languages than C (e.g., Rust, Haskell, Java)?

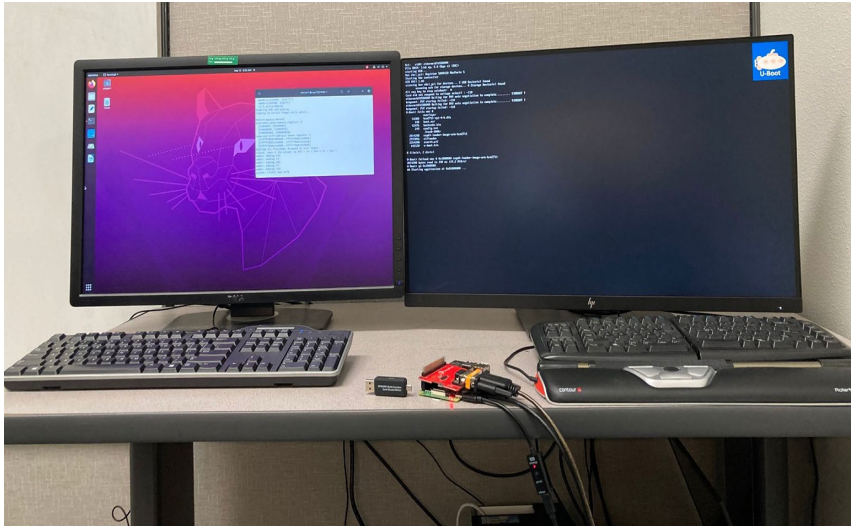
"Software Verification and seL4: Implementing Untrusted Code in a Trusted Environment", B. Lara, June 12, 2023

Objective - Setup Test Environment, Experience



Objective - Setup Test Environment, Experience





Getting Started

- Hardware and Target Supported Platforms

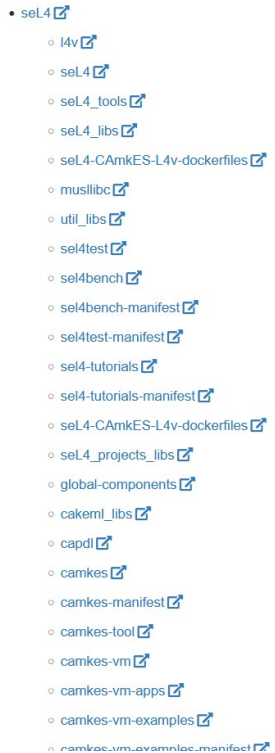
Platform	SOC	Core	Arch	Status
Zynq-7000	Zynq7000	Cortex-A9	ARMv7A	Unverified
Raspberry Pi 3-b	BCM2837	Cortex-A53	ARMv8A	Unverified
Raspberry Pi 4-b	BCM2711	Cortex-A72	ARMv8A	Unverified

Getting Started

There are many repositories (42). Of the most significant are:

- l4v - the seL4 proofs
- seL4 - the seL4 kernel

Repository	Motivation
seL4	Standalone build
sel4test	More guidance
sel4-tutorials	Examples



https://docs.sel4.systems/MaintainedRepositories

Getting Started

- Hardware and Target Supported Platforms

Platform	SOC	Core	Arch	Status
~~Zynq-7000~~	~~Zynq7000~~	~~Cortex-A9~~	~~ARMv7A~~	~~Unverified~~
Raspberry Pi 3-b	BCM2837	Cortex-A53	ARMv8A	Unverified
Raspberry Pi 4-b	BCM2711	Cortex-A72	ARMv8A	Unverified

Getting Started

- Hardware and Target Supported Platforms

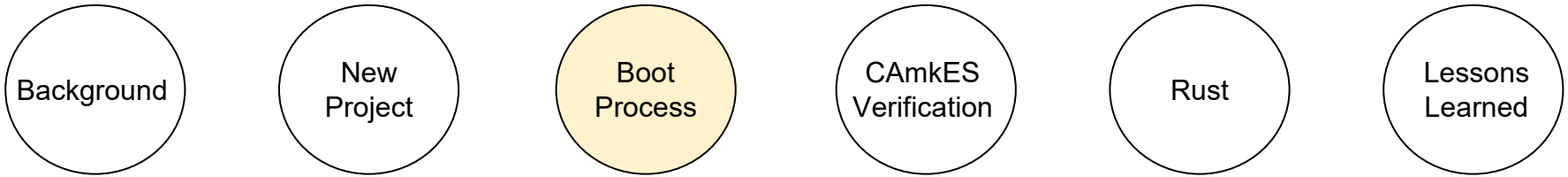
Platform	SOC	Core	Arch	Status
~~Zynq-7000~~	~~Zynq7000~~	~~Cortex-A9~~	~~ARMv7A~~	~~Unverified~~
~~Raspberry Pi 3-b~~	~~BCM2837~~	~~Cortex-A53~~	~~ARMv8A~~	~~Unverified~~
Raspberry Pi 4-b	BCM2711	Cortex-A72	ARMv8A	Unverified

seL4 Documentation

- Reviewed of published documentation about seL4 and compiled our findings
- Audited and documented the source code of seL4
- Performed binary analysis of compiled binaries of seL4 and documented our assessment
- Delivered a 50+ page document of our seL4 assessment

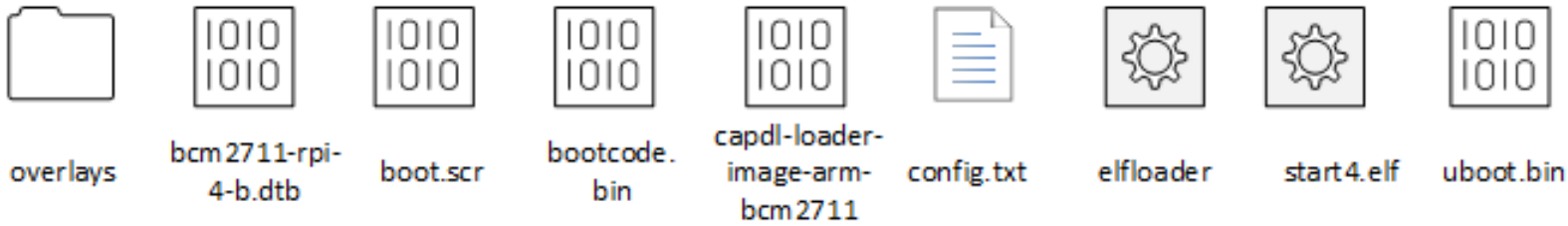
"Software Verification and seL4: Implementing Untrusted Code in a Trusted Environment", B. Lara, June 12, 2023

Overview


Background
New Project
Boot Process
CAmkES Verification
Rust
Lessons Learned

Booting on seL4 Hardware

- Stage 1 is an on board ROM bootloader. This loads bootcode.bin.
- Bootcode.bin inits the GPU and loads the next stage boot loader, start4.elf.
- Start4.elf then loads our u-boot image. This is when the seL4 boot process begins and the Raspberry Pi generic boot process ends.
- U-boot loads in Elfloader, which loads both images into memory
- The kernel takes over from Elfloader and initializes the initial thread
- The initial thread runs, schedules userland programs, and yields control.



"seL4 Notes", B. Lara/LANL, "Software Verification and seL4: Implementing Untrusted Code in a Trusted Environment", B. Lara, June 12, 2023

Assumptions

The Proof

- **Boot code:** the proof currently is about the operation of the kernel after it has been loaded correctly into memory and brought into a consistent, minimal initial state. This leaves out about 1,200 lines of the code base that a kernel programmer would usually consider to be part of the kernel.

https://sel4.systems/Info/FAQ/proof.pml

Supported Platform - Raspberry Pi 4 Model B

There is a website that provides guidance

- Serial connection wiring
- U-Boot commands
  - Need a special version to disable caching
- SD card setup
  - The instructions say to compile U-boot and transfer the uboot.env file to the sdcard, this file is actually called .config and is not required.
- Getting seL4 onto the Raspberry Pi 4
  - The instructions say to load the kernel image at address 0x10000000, but this address is reserved on the Raspberry Pi

https://docs.sel4.systems/Hardware/Rpi4.html

Automated Deployment

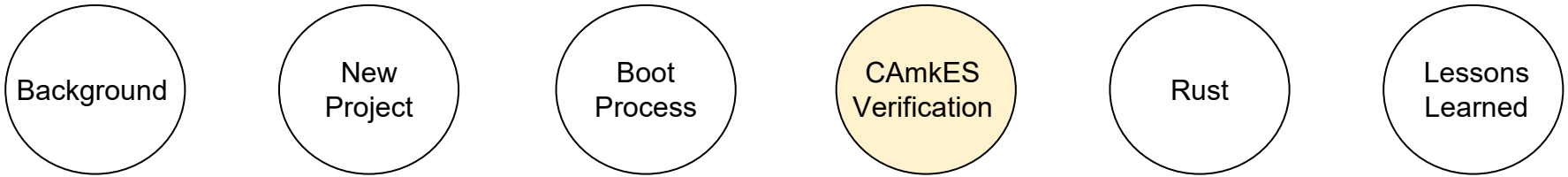
Created a build script that compiles seL4 and all of its dependencies

- apt installs <all dependencies>
- Builds the bootloader: U-boot
- Patches the kernel with a required patch
- Compiles the kernel and an example HelloWorld program
- Creates an sdcard image containing the bootloader, compiled kernel, and configuration files

The copyright pending hardware independent seL4 build system will be available ~November 2023 on the Los Alamos National Laboratory GitHub.

"Software Verification and seL4: Implementing Untrusted Code in a Trusted Environment", B. Lara, June 12, 2023

Overview


Background
New Project
Boot Process
CAmkES Verification
Rust
Lessons Learned

seL4 CAmkES Component Framework

- Framework aimed at embedded and cyber-physical systems, which typically have a static architecture
  - Defined set of components that don't change once the system is booted up.
- The CAmkES system is specified in a formal architecture description language (ADL), which describes the components, their interfaces and the connectors that link them up.
- The promise of CAmkES is that what is specified in the ADL, is a faithful representation of possible interactions.
- The promise depends on enforcement by seL4 and the ADL representation must be mapped onto the seL4 low-level objects.

https://docs.sel4.systems/projects/camkes/manual.html

Assumptions

The seL4 Microkernel - An Introduction (Rev. 1.2 of 2020-06-10)

- Note: At the time of writing, the proofs about CAmkES and CapDL are not yet complete, but completion should not be far off.

The L4.verified Proofs

- Camkes: an initial formalisation of the CAmkES component platform on seL4. Work in progress.

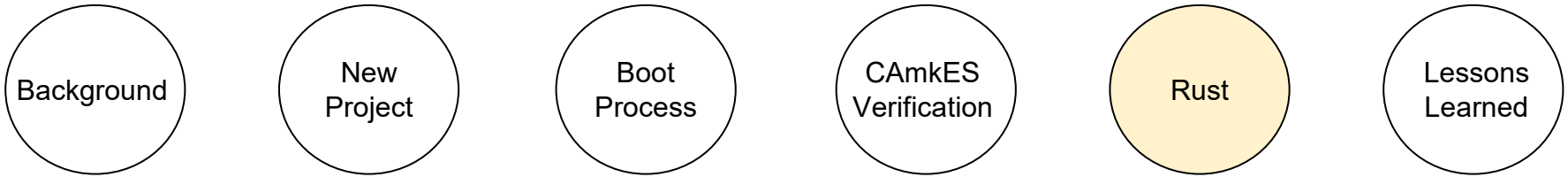
adder.camkes

```
/*
 * Copyright 2017, Data61, CSIRO (ABN 41 687 119 230)
 *
 * SPDX-License-Identifier: BSD-2-Clause
 */

import <std_connector.camkes>;

import "components/Adder/Adder.camkes";
import "components/Client/Client.camkes";

assembly {
        composition {
        component Adder adder;
        component Client client;

        connection seL4SharedData s(from adder.d, to client.d);
        connection seL4RPCCall p(from client.a, to adder.a);
        }
}
```

```
Bootstrapping kernel
available phys memory regions: 3
  [1000000..3b400000]
  [40000000..fc000000]
  [100000000..200000000]
reserved virt address space regions: 3
  [ffffff8001000000..ffffff800123d000]
  [ffffff800123d000..ffffff800124356b]
  [ffffff8001244000..ffffff800145d000]
Booting all finished, dropped to user space
client: what's the answer to 342 + 74 + 283 + 37 + 534 ?
adder: Adding 342
adder: Adding 74
adder: Adding 283
adder: Adding 37
adder: Adding 534
client: result was 1270
```

https://github.com/seL4/camkes/tree/master/apps/adder

Overview


Background
New Project
Boot Process
CAmkES Verification
Rust
Lessons Learned

Extending the Verification

- The protocol converter application was written in Rust
  - Chosen for safety guarantees built into the programming language design
    - Example: Rust prevents entire classes of memory vulnerabilities by design (e.g. buffer overflows, use after free, data races).

- Rust
  - Uses a borrow checking and lifetime scheme to guarantee memory safety
  - Does not protect against things like:

    if (a < b) vs if (a <= b)

"Memory-Safety Challenge Considered Solved? An In-Depth Study with All Rust CVEs" Hui Xu, Zhuangbin Chen, Mingshen Sun, Yangfan Zhou, Michael Lyu, "Software Verification and seL4: Implementing Untrusted Code in a Trusted Environment", B. Lara, June 12, 2023

seL4 and Rust

- seL4 Rust webpage
  - “The rust support that this page talks about is no longer supported”.

- Other projects
  - Some involved writing the CapDL initialization step.

seL4 and Rust

- Both Rust and C compile to assembly binaries
  - These binaries are the same.
- Issue
  - The standard library contains operating system specific system calls.
- Plan
  - Write Rust without any use of the standard library, by using the assembly.

"Software Verification and seL4: Implementing Untrusted Code in a Trusted Environment", B. Lara, June 12, 2023

The Embedded Rust Book

- A no_std Rust Environment
  - In bare metal environments, no code has been loaded before your program (does not load the standard library).
  - It then depends on the hardware, your crates and program to run.

- A little Rust with your C
  - #[no_mangle]
    - The Rust compiler normally mangles symbol names differently than native code linkers expect.
    - Any function that is exported by Rust needs to be told not to be mangled by the compiler.

"Software Verification and seL4: Implementing Untrusted Code in a Trusted Environment", B. Lara, June 12, 2023

Compile and Link

- Was able to compile this Rust code with the following command
  - cargo build –target aarch64-unknown-none

- Was able to link Rust by specifying in CMake that the Rust binary was a static library.
  - This links the Rust assembly into our C assembly.

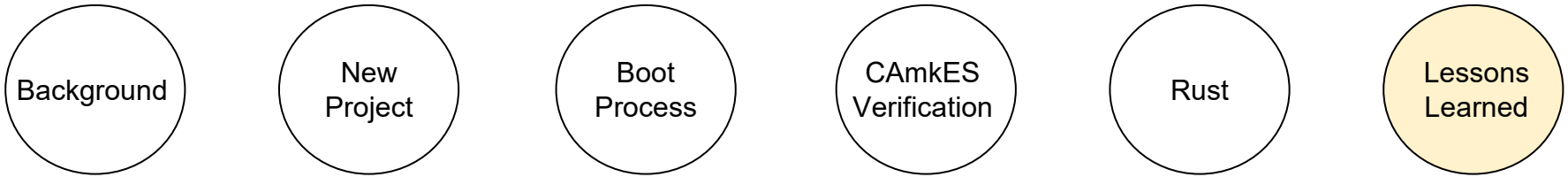
"Software Verification and seL4: Implementing Untrusted Code in a Trusted Environment", B. Lara, June 12, 2023

Assumptions

The Proof

- **Assembly**: the seL4 kernel, like all operating system kernels, contains some assembly code, about 340 lines of ARM assembly in our case, and a similar amount for the RISCV64 version. For seL4, this concerns mainly entry to and exit from the kernel, as well as direct hardware accesses. For the proof, we assume this code is correct.

Overview


Background
New Project
Boot Process
CAmkES Verification
Rust
Lessons Learned

Lessons Learned

- What are the security impacts of interactions between formally verified and unverified components?
- What is the overhead of developing, deploying and integrating seL4 into existing systems?
- Can the security guarantees be extended to user space programs without using a VM?
  - Can seL4 run safer languages than C (e.g., Rust, Haskell, Java)?