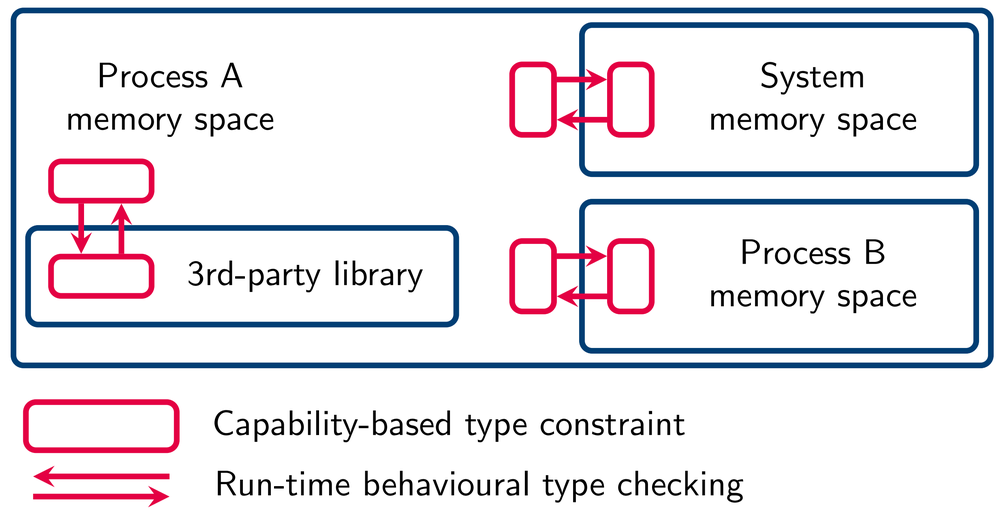
APPCONTROL: ENFORCING APPLICATION BEHAVIOUR THROUGH TYPE-BASED CONSTRAINTS

Wim Vanderbauwhede[1], José Cano[1], Jan de Muijnck-Hughes[1], Laura Voinea[1], Nobuko Yoshida[2], Martin Vassor[2], Klaus McDonald-Maier[3], Xiaojun Zhai[3], Ludovico Poli[3], Michal Borowski[3]

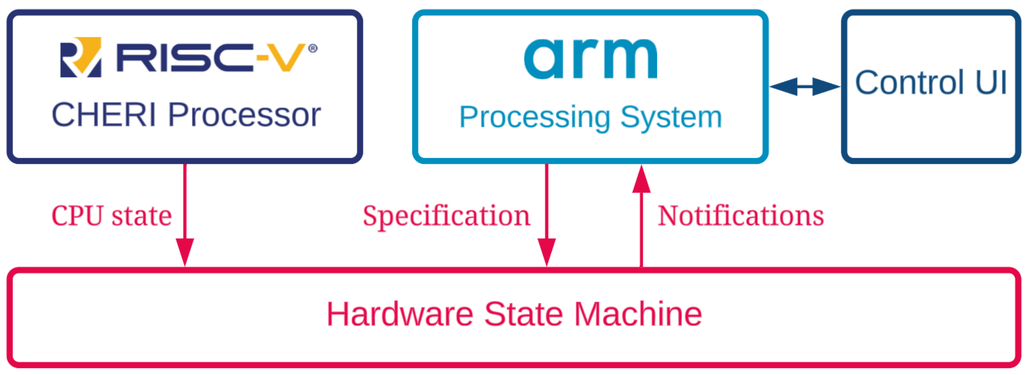
[1]University of Glasgow, [2]University of Oxford, [3]University of Essex

PROJECT OVERVIEW

CHERI capabilities provide fine-grained memory protection, limiting access privileges of third-party applications. However, capabilities say nothing about **program behaviour**. We use **Behavioural Types** to capture the behavioural structure of application interfaces in order to **secure program interaction**.



Behavioural types ensure correctness of behaviour, provided that the specification is correct. Debugging a specification-based system demands the ability to **debug the specification at run-time**.



OUR APPROACH

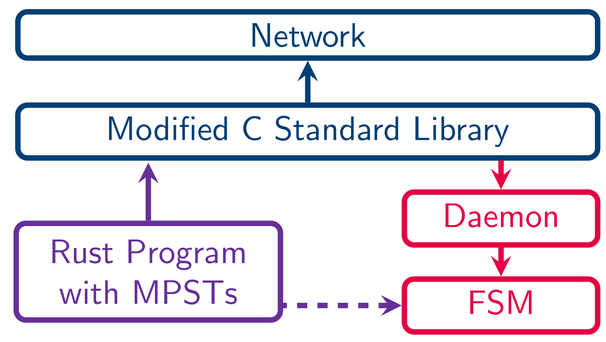
- Investigate the interplay of CHERI capabilities and behavioural types in capability-native languages (e.g. Rust, C) using experimental language Capable. Capable is a barebones imperative language written in Idris2 as a intrinsically-scoped/typed EDSL.
- Develop **Behavioural APIs** enabling use of CHERI Capabilities.
- Develop **multiparty session type (MPST)** theories and tools to ensure capability-based behavioural properties in capability native languages.
- Develop a framework to enable the monitoring and **debugging** of capability-supporting code.

ACKNOWLEDGEMENTS

The authors thank the UKRI Digital Security by Design (DSbD) Programme for funding the AppControl project through grant EP/V000462/1 and Morello-Hat through grant EP/X015955/1.

RUST API

The Rust API will connect MPSTs with CHERI's memory control capabilities. We identify socket layer system calls to be the most important use case for monitoring adherence to the specification. As Rust's POSIX socket library is built on top of the C standard library, we can address this by modifying the libC to communicate these calls to a listening daemon. The daemon can maintain a finite-state machine (FSM) determined by the specification.



MPST IN RUST

- Developed Rumpsteak, a library for asynchronous MPST in Rust
- Support protocol optimisation with message reordering
- Protocols generated from 3rd party library **statically type-checked** for behavioural correctness, ensuring **deadlock-freedom**.
- We are currently adapting Rumpsteak for the Rust API.
- `https://github.com/zakcutner/rumpsteak`

A ring protocol in Rumpsteak:

```
1 type Source = Receive<T, Ready, Select<T, SourceChoice>>;
2 enum SourceChoice { Value(Value, Source),
3                     Stop(Stop, End) }
4 type Sink = Send<S, Ready, Branch<S, SinkChoice>>;
5 enum SinkChoice { Value(Value, Sink),
6                   Stop(Stop, End) }
```

DEBUGGER AND DIAGNOSTICS TOOLS

- Prototypes of non-intrusive, on-chip debug system for CHERI-RISC-V processors.
- Extraction of program metrics from the custom implementation of CHERI Flute based SoC.
- Touch screen control interface for dynamic configuration and overview of the monitored system.
- ZC706 board implementation of PYNQ wrapper for CHERI-RISC-V Flute processor, utilizing Continuous Monitoring System for baremetal programs.
- Performance comparison of conventional and novel (e.g. autoencoder-forest) abnormal behaviour detection methods.

PROJECT WEB SITE AND REPOSITORIES

`https://dsbd-appcontrol.github.io/`

`https://github.com/DSbD-AppControl`

BONUS

MORELLO-HAT: HIGH-LEVEL API AND TOOLING

Wim Vanderbauwhede[1], José Cano[1], Cristian Urlea[1], Nobuko Yoshida[2], Martin Vassor[2], Klaus McDonald-Maier[3], Xiaojun Zhai[3], Ludovico Poli[3], Michal Borowski[3]

The **Morello-HAT** project intends to create a common API that can be used by compiler developers as well as programmers of higher-level languages, to allow them to leverage Morello's HW capabilities to improve memory security and type safety, spatial as well as temporal, of their language and programs.

The project consists of three work packages, one to develop the API using C++, Rust, Go and Dart, one to demonstrate the usability and effectiveness of the API on a series of example applications by ML-based detection of vulnerabilities and assessment of the effectiveness of mitigation through the use of the API and one to use HW capabilities to enhance the debug infrastructure.