Type-based security properties assurance in the Rust-based Redox operating system

**Executive summary:** We propose to explore using types in operating system source code as a mean to get assurance on security properties. The use case is the Rust-language-based Redox operating system and its generic service-access framework called schemes. Schemes generalize Unix’ “everything is a file” notion to “everything is a URL”. First it should be shown how the *typestate* design pattern helps getting assurance on isolating clients of a scheme providing access to a cryptographic resource. Second a comparative study should be done on generically implementing resource read- and write- access restriction in the kernel, using traditional attribute-based checks on one side and type-based segregation of handles on the other side.

**Keywords:** Operating systems; Security; Programming languages; Rust; Redox.

**Advisors:**
- Louis Rilling (DGA, team Sushi) louis.rilling@irisa.fr
- Frédéric Tronel (CentraleSupélec, team Sushi) frederic.tronel@centralesuplec.fr

**Team:** Sushi. **Laboratory:** IRISA, Rennes (head: Guillaume Gravier – guig@irisa.fr).

**Required skills:** System programming.

**Appreciated but not mandated skills:** Deep understanding of OSes; Programming in Rust.

**Context and Description**

Operating systems, especially their kernel, are critical software to build applications aiming at providing security properties. A vulnerability in the kernel opens a door for attackers to bypass the application logic, whatever the correctness of the application itself.

Getting assurance on security properties of operating systems services is known to incur high costs. This traditionally relies on careful design and heavy testing, with costs of up to 400% of a non-secure development effort [1]. While formal verification remains a challenge for this class of software, few projects achieved this goal for an up to 1000% cost [2].

We propose to explore a middle way to get assurance, relying on the compiler to check both memory safety and higher-level, logical, security properties that should be encoded in the source code using types. It is expected that this reduces the debugging effort and it could make formal verification easier.

An example of relevant techniques for operating system development is the *typestate* design pattern [3]. This technique gives assurance on the implementation of state machines, by assigning a dedicated type to each state of the state machine and encoding state machine transitions as functions converting from a source state to a target state. Invalid transitions are thus made impossible because the converting functions just do not exist. *Session types* [4] achieve a similar goal for protocols.

Although using types to achieve security has been known for decades, this practice using the programming language of an operating system is hardly explored. Current approaches rely on a separate language and its compiler to write annotations in the original source code and verify that the code satisfies the specified properties [5].

Yet applying typestate analysis on production source code like Linux shows that the security of such development could be improved if types were used to encode logical properties [6]. On the other hand, using the type system of the programming language to ensure functional properties is the topic of ongoing research [7].

The Rust programming language is especially a good candidate as it is strongly- and statically-typed, by design the compiler checks the memory safety of programs, and it is the
basis of several open-source operating system projects, including Redox ¹, which explicitly targets security. Moreover successful Rust-implementations were demonstrated for typestates [8] and session types [9].

One of the mechanisms provided by Redox to achieve security is called schemes. Schemes generalize Unix’ “everything is a file” notion to “everything is a URL”. Using namespace reduction of accessible schemes (the Redox-specific setrens system call), open handles can then be used as capabilities [10].

After a study of the state of the art, the suggested work plan is as follows:

• First it should be shown how the typestate design pattern helps getting assurance on isolating clients of a scheme providing access to a cryptographic resource. To this end a representative userspace scheme daemon will be implemented in Rust for Redox.

• Second a comparative study should be done on generically implementing resource read- and write- access restriction in the kernel, using traditional attribute-based checks on one side and type-based segregation of handles on the other side. During the study the two approaches will be implemented and experimentally compared.

References


¹https://www.redox-os.org/