COMP 517: Endocert Proposal

Alexis Le Glaunec (`afl5`)
Ziyang "Zion" Yang (`zy36`)
Senthil Rajasekaran (`sr79`)

October 12, 2021

1 Introduction

1.1 Problem + why it's important

Im et al. [2021] introduced the *endokernel*, an architecture designed to isolate subcomponents within processes by virtualizing access to all system resources. This was mainly accomplished through the use of new syscall monitor, the *nexpoline*, a mechanism nested within the process which screens syscalls based on potential security threats defined by a prespecified policy. The introduction of any new architectures can also introduce new security issues and bugs. Therefore, it is important to provide proof that the architecture works with few undesirable emergent behaviors. Proving this architecture sound will increase trust in the proposed solution, allowing for a wider userbase and further work to be built on top of the endokernel.

1.2 Research context or gap in existing approaches

The endokernel is a promising solution to current problems in the security community. Currently, its implementation stands at 6,400 added lines of code Im et al. [2021], and its correctness has only been established through a finite set of test cases. This means that custom-made attacks are underserved by the current testing-based approach.

1.3 The aim/goal/hypothesis

Compared to a testing-based approach, a more rigorous approach, such as those offered by formal verification tools provide a greater level of confidence in the soundness of the system. Therefore, our goal is to apply these methods to implementations of critical interfaces of the endokernel in order to verify the architecture's robustness.

1.4 The proposed solution: how will you solve it?

We plan on creating a high-level model of the endokernel architecture in a formal verification tool in order to verify key isolation properties. These properties will translate into resource allocation requirements that must hold in order for a system to robustly deal with malicious code within a process.

1.5 How you will evaluate it?

Ideally, we would like to prove total isolation between subprocesses under the endokernel architecture. We will do this by incrementally adding properties on the system abstractions and verifying them. These properties will be scaled on quality by the severity of the threat they represent should they be violated. A robust system will therefore satisfy many properties along with many high quality properties.

1.6 Anticipated contributions

First, we will provide a new model of the endokernel written in a formal verification tool. Then, we will use the tool to prove properties on the model that directly translate into real-world security requirements based on avoiding in-process attacks.

2 Background

This problem is motivated by a long standing collaboration between the fields of systems and verification formed in order to prove real systems correct. This project is motivated by Im et al. [2021], as well as previous works in verification for systems as follow.

1. Mai et al. [2013] (`Dafny + C# Contract`)
   ExpressOS formalizes a microkernel for mobile devices. To verify critical security invariants on their implementation, `BOOGIE`-style tools (Dafny, VCC) are incorporated into the code. It consists in adding Code Contracts to the code: pre-conditions, post-conditions and assertions. Then, a SMT-solver will prove that all contracts are respected. `Ghost variables`, i.e. variables with no semantics for the kernel, are added to ease the proofs. This approach is expected to take significantly less time than full completeness verification like Klein et al. [2009] in `Coq`.

2. Shinde et al. [2020] (`Intel SGX + Coq`)

   BesFS aims to protect application from tampering of a malicious OS. It consists of two parts: (1) a formal specification of the file-system interface that any OS should comply with (2) a formally verified monitor of OS file-system interface and encryptor of application data Using Intel SGX techniques. BesFS runs as a library linked to userland applications. In case of malicious OS behavior (deviant from the spec), BesFS detects it and simply aborts the user application. As mentioned in the paper, formalization is used as a mean to (1) verify the specification of OS API to ensure **safety** (capture well-known attack) and **completeness** (admits normal functionality). (2) verify the implementation of BesFS.

3. Chong et al. [2020] (`SMT solver + C contract`)

   This paper is part of an effort verifying the industrial library aws-common. Both of this and Dafny uses SMT solver, but the language that is checked in this paper is C. Programmers specify precondition, postcondition of functions. Symbolic execution is used to verify the correctness of the functions.

3 Methods and Plan

At first, we plan on formalizing the main abstractions used for the endokernel architecture, which are the process abstraction, the resource virtualization (memory, signals, IO) and the syscall monitor. Once these are implemented in a formal verification tool we will define our key properties. These properties will be based on the goals of the endokernel design that need to hold for the design to be sound. Finally, we will incrementally verify those properties on the object abstractions and then the whole endokernel system.

4 Milestones

1. **10/17:** Define on paper the main abstractions of the endokernel architecture.
2. **10/24:** Code the abstractions in a verification tool language and begin defining properties to verify on paper.
3. **10/26:** Midterm meetings - Demonstrate the model and the relevant abstractions.
4. **11/1:** Manually prove properties for the endokernel abstractions on paper.
5. **11/15:** Prove these properties using a verification tool.
6. **11/30:** Prepare a presentation
7. **12/3:** Finalize the final report

Bibliography

B. Beckert and R. Hahnle. Reasoning and verification: State of the art and current trends. *IEEE Intelligent Systems*, 29(01):20–29, jan 2014. ISSN 1941-1294. doi: 10.1109/MIS.2014.3.

Nathan Chong, Byron Cook, Konstantinos Kallas, Kareem Khazem, Felipe R Monteiro, Daniel Schwartz-Narbonne, Serdar Tasiran, Michael Tautschnig, and Mark R Tuttle. Code-level model checking in the software development workflow. In *2020 IEEE/ACM 42nd International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, pages 11–20. IEEE, 2020.

Bumjin Im, Fangfei Yang, Chia-Che Tsai, Michael LeMay, Anjo Vahldiek-Oberwagner, and Nathan Dautenhahn. The endokernel: Fast, secure, and programmable subprocess virtualization. *CoRR*, abs/2108.03705, 2021.

Gerwin Klein, Kevin Elphinstone, Gernot Heiser, June Andronick, David Cock, Philip Derrin, Dhammika Elkaduwe, Kai Engelhardt, Rafal Kolanski, Michael Norrish, Thomas Sewell, Harvey Tuch, and Simon Winwood. Sel4: Formal verification of an os kernel. In *Proceedings of the ACM SIGOPS 22nd Symposium on Operating Systems Principles*, SOSP '09, page 207–220, New York, NY, USA, 2009. Association for Computing Machinery. ISBN 9781605587523. doi: 10.1145/1629575.1629596. URL `https://doi.org/10.1145/1629575.1629596`.

K. Rustan M. Leino. Dafny: An automatic program verifier for functional correctness. In *Proceedings of the 16th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning*, LPAR'10, page 348–370, Berlin, Heidelberg, 2010. Springer-Verlag. ISBN 3642175104.

Haohui Mai, Edgar Pek, Hui Xue, Samuel Talmadge King, and Parthasarathy Madhusudan. Verifying security invariants in expressos. *SIGPLAN Not.*, 48(4):293–304, March 2013. doi: 10.1145/2499368.2451148. URL `https://doi.org/10.1145/2499368.2451148`.

Luke Nelson, Helgi Sigurbjarnarson, Kaiyuan Zhang, Dylan Johnson, James Bornholt, Emina Torlak, and Xi Wang. Hyperkernel: Push-button verification of an os kernel. In *Proceedings of the 26th Symposium on Operating Systems Principles*, SOSP '17, page 252–269, New York, NY, USA, 2017. Association for Computing Machinery. ISBN 9781450350853. doi: 10.1145/3132747.3132748. URL `https://doi.org/10.1145/3132747.3132748`.

Shweta Shinde, Shengyi Wang, Pinghai Yuan, Aquinas Hobor, Abhik Roychoudhury, and Prateek Saxena. Besfs: A POSIX filesystem for enclaves with a mechanized safety proof. In *29th USENIX Security Symposium (USENIX Security 20)*, pages 523–540. USENIX Association, August 2020. ISBN 978-1-939133-17-5. URL `https://www.usenix.org/conference/usenixsecurity20/presentation/shinde`.