Toward Efficient Formal Verification of Reference Monitors for Isolated Execution

Ryo Nakashima
The University of Tokyo
Tokyo, Japan

Takahiro Shinagawa
The University of Tokyo
Tokyo, Japan

EXTENDED ABSTRACT

Recent growth in cloud computing increases the importance of leveraging isolated execution, from classical isolation with processes and virtual machines to modern trusted computing, to enhance security. The security of isolated execution relies on system software that serves as trusted computing base (TCB), i.e., OS kernels, hypervisors, and enclave software, which are assumed to work correctly. However, increasing functional requirements for system software often bloat the TCB size, making it difficult to guarantee their correctness.

Formal verification is an effective method to improve security by verifying that a system satisfies certain properties. However, formal verification is a time-consuming task that requires significant effort by experts, making it difficult to apply to large-scale software. In addition, formal verification only guarantees certain properties that are explicitly verified, making it difficult to mitigate a variety of attacks based on arbitrary code execution. Therefore, to deal with diversified attacks with limited effort, verification targets and properties need to be carefully selected.

Several studies take the approach of simplifying the code to be verified. Sridhar et al. [2] target a security-relevant component called inlined reference monitors and use model checking to prove conformance to security policy. However, they only verify conformance under specific conditions and do not consider various attacks. Komodo [1] presents an approach to efficiently prove the entire isolation among TEE enclaves and the OS by verifying a simple software manager for the enclaves. However, while many of the management functions are not directly related to security, the correctness of these functions must be verified together in order to prove the overall isolation.

Our approach to efficient formal verification is to focus on security-critical components while assuming a wide variety of attacks. The security-critical components we focus on are reference monitors, which are access control components that operate at the interface between isolated execution environments and are embedded in or add-in to the TCB system software. Since reference monitors directly handle input from attackers and are often the target of attacks, their formal verification is a cost-effective way to improve security.

As a first step toward formal verification of general reference monitors, we target the time-of-check-to-time-of-use (TOCTTOU) problem in monitoring system calls. TOCTTOU is a timing attack in which the object to be accessed is replaced by another object after the reference monitor has checked the access rights to the object but before the subject actually accesses the object. The challenge in verifying against TOCTTOU attacks is the diversity of attacks; there are many combinations of interleaving system calls that can cause race conditions, and reference monitors must be verified against all such combinations.

To address this challenge, we introduce a two-step exhaustive verification with automatic verification code generation and parallel verification. Based on a predefined access model for defining sequences of system call operations and arguments that can lead to race conditions, we exhaustively generate code for monitored subjects and attackers that can cause TOCTTOU. We then perform model checking against reference monitors in parallel for each generated code. This two-step verification allows to verify that reference monitors can correctly enforce a given security policy without causing TOCTTOU even if an attacker issues arbitrary system calls at arbitrary timing and in arbitrary order.

We develop our framework on top of Linux. To explore interleaving for concurrent file access, we implement a mock file system that emulates the minimum functionality required for verification. The internal data structures of files are provided by Loom, the model checker used in this study, to identify relevant interleaving. All code is implemented in Rust to take advantage of memory safety and reduce verification effort.

We evaluated our framework in a case study: we verified the conformance to a file access policy under the conditions of one access pattern (call to open), one vulnerability (symlink race), and one attack pattern (re-linking of target files). As a result, we confirmed that the verification succeeded for a reference monitor with a symlink race countermeasure, but failed for one without the countermeasure.

REFERENCES

[1] Andrew Ferraiuolo et al. Komodo: Using verification to disentangle secure-enclave hardware from software. In *Proc. 26th Symposium on Operating Systems Principles*, pages 287–305, 2017.

[2] Meera Sridhar et al. Model-checking in-lined reference monitors. In *Proc. International Workshop on Verification, Model Checking, and Abstract Interpretation*, pages 312–327, 2010.