STEVENS
INSTITUTE OF TECHNOLOGY
1870

Ph.D. DISSERTATION DEFENSE

Candidate:	Vidya Lakshmi Rajagopalan
Degree:	Doctor of Philosophy
School/Department:	Charles V. Schaefer, Jr. School of Engineering and Science / Computer Science
Date:	Thursday, December 4th, 2025
Time/Location:	2:00 PM / GN 303
Title:	Automated Generation of System Call Based Sandboxes in Binaries
Chairperson:	Dr. David A. Naumann, Department of Computer Science, Stevens Institute of Technology
Committee Members:	Dr. Georgios Portokalidis (co-advisor), IMDEA Software Institute Dr. Jun Xu, School of Computing, University of Utah Dr. Michael Greenberg, Department of Computer Science, Stevens Institute of Technology

ABSTRACT

Applications typically use only a subset of the system calls provided by an operating system. Removing unused system calls - known as system call debloating - reduces the kernel's attack surface and limits attacker capabilities, thwarting privilege escalation attacks. Recent approaches automate the identification of system calls required by an application and block the rest. Some also enforce phase-specific system call policies, particularly for server applications. However, these methods often require source code access, depend on users to identify phase transitions (e.g., from initialization to serving clients), or fail to account for dynamically loaded libraries.

In the first part of this thesis, we present SysPart, an automatic system call filtering framework for binary-only server programs. SysPart introduces a novel algorithm that combines static and dynamic analysis to identify the serving phases of all server threads. It performs static analysis to determine the system calls required in each serving phase and resolves dynamically loaded libraries in a sound manner. Evaluation on six popular x86-64 Linux servers demonstrates that SysPart accurately identifies serving phases, generates effective system call filters, and mitigates real-world attacks. SysPart outperforms prior binary-only approaches and achieves performance comparable to source-code-based techniques.

Generating precise system call filters requires an accurate and sound call graph of the application and its libraries. However, indirect control flow constructs make this challenging. Conservative methods that overapproximate indirect calls to target all functions or address-taken functions are sound but severely imprecise, diminishing filter effectiveness. In the second part of this thesis, we evaluate the impact of state-of-the-art indirect call refinement techniques on system call filtering. Our analysis reveals that the imprecision during the analysis of libc is one of the key reasons for overapproximating the set of system calls. Therefore, sound and precise refinement of the call graph

of libc can improve code and system call debloating, strengthen Control-Flow Integrity (CFI), and enhance overall system security.

Among the available implementations, glibc is the de facto standard on most major Linux distributions, because it provides performance, high compatibility and a rich and stable API. Moreover, much software assumes glibc behaviour and does not work correctly with other libc versions. However, glibc's large, complex codebase and dependency on GCC—whose pointer analysis capabilities remain limited— makes the application of source-based approaches practically infeasible. Binary analysis methods, in turn, face scalability challenges. To address this, in the final part of this thesis, we define a set of heuristics—derived from a manual analysis of glibc's open-source code—that can be incorporated into automatic binary analysis to enhance the precision of call graph. Our heuristics model key internal mechanisms of glibc and classify its function-pointer usage patterns to resolve the targets of indirect call constructs within glibc. Implemented on top of the Egalito binary analysis framework and SysPart's static call graph generator, our approach strengthens the effectiveness of CFI, code and system call debloating, thereby improving system security by increasing the difficulty of mounting successful exploits.