Context

Man-at-the-end (MATE) attackers are a problem both to users and intellectual property right (IPR) owners. A MATE attack can occur via malware that tampers with running processes, effectively taking control over them and changing their intended control flow. For example, malware could modify the control flow of the Chromium browser [1] and trick a user, for instance, into navigating to a phishing site. On the other end, IPR owners seek for a tamper-free execution of their software. However, a MATE attacker could analyze the binary and remove existing protections like licensing mechanisms or disclose secret information.

In this context, research proposed different software integrity protection primitives (SIPP), e.g., self-checksumming (SC) [2], control-flow integrity (CFI) [1], result-checking (RC) [3] oblivious hashing (OH) [4] and obfuscation [5], to mitigate such attacks. Each SIPP normally receives a set of protection parameters (PParam) enabling users to tune them in according to their needs.

Despite the existence of such mechanisms perpetrators may still be able to bypass applied software protections. There exist generic algorithms that can defeat some SIPPs (when used exclusively) [6, 7]. Therefore, it is desirable to compose a multitude of SIPPs to thwart a wider range of attacks. Unfortunately, manually composing SIPPs is a daunting task, as it requires a comprehensive reconciliation of potential inter-protection conflicts. Moreover, the order in which protections are applied, and their PParams could severely impact security guarantees and imposed overheads.

Goal

The goal of this thesis is to create a software protection composition framework (SPCF) aiming for more resilience against MATE attacks. In a nutshell, SPCF composes existing (some available on our github page\(^1\)) SIPPs (viz. CFI, SC, OH and RC) while guaranteeing that protected binaries remain intact (i.e. without conflicts). For this purpose, the student will first model the constraints of each protections (in form of pre- and post- conditions). This model enables SPCF to properly compose SIPPs (by ensuring such conditions hold throughout the chaining) in arbitrary orders with given sets of PParams. Finally, the student will study the imposed overhead and security implications of different composition combinations (which are randomly or systematically generated) on a given set of programs.

Working Plan

1. Write a state-of-the-art survey on SIPPs
2. Identify (semi) formal constraints w.r.t. a combination of SIPPs
3. Implement a SIPP composition framework
4. Conduct a case study with real-world software
   (a) Apply SIPPs onto the existing systems using the composition framework
   (b) Evaluate the security of different SIPP combinations using attack defense trees
   (c) Evaluate the performance of the mechanism including memory and runtime overhead analysis
   (d) If time permits, identify best (least penalty in performance with the highest possible security) possible combination of SIPPs using a Genetic Algorithm

Deliverables

- Docker container able to run a demo of the implementation, including instructions on how to run the demo
- The container should also include the source code of the implementation

---

\(^1\)https://github.com/tum-i22/sip-toolchain
• Technical report with comprehensive documentation of the implementation, i.e. design decision, architecture description, API description and usage instructions
• Final thesis report written in conformance with TUM guidelines

References


