Raising the Bar against Automated Attacks using x86 Machine Code Diversity Transformations

Master thesis

Supervisors: Prof. Dr. Alexander Pretschner, Sebastian Banescu
Email: {alexander.pretschner, sebastian.banescu} @ in.tum.de
Phone: +49 89 289 – 17, 314
Starting date: immediately

Context

Software diversity, self-modification, and obfuscation have many applications in software security and attack resilience. For example, software diversity schemes creating individualized binaries can, by scrambling memory and code layout, make attacks based on pattern matching attacks unsuccessful against all installations of one software product. Self-modification can help against runtime analysis, attacks, and binary patching which would, e.g., remove key, authorization, or integrity checks. Obfuscation is commonly applied to protect programs against analysis and reverse engineering. While the aforementioned concepts may not offer perfect security, they can raise the bar for attackers such that reliable attacks become too expensive to pay off.

Goal

The main objective of this thesis is to develop and evaluate the effectiveness of x86 machine code transformations against attacks that try to identify and disable checks (e.g. license checks, code integrity checks, etc.). Transformations may include software diversity, stealth, and obfuscation schemes [1], [2]. The effectiveness of different software transformations against attacks will be investigated via case-studies different versions of popular software applications (e.g. Chrome, Firefox, Thunderbird, OpenSSL, etc.), which have been publicly associated with vulnerabilities. Alternatively or additionally, the effectiveness of the software transformations can also be evaluated by applying them to open source projects freely available on the Internet.

Work-plan

1. Develop understanding of stealth, obfuscation, and software diversity techniques
   a) Write state-of-the-art survey of several techniques
   b) Write security vs. performance analysis of existing techniques
2. Implement modules for machine code level and memory layout transformations:
   a) Implement module(s) for equivalency substitutions (by expanding or reducing existing instructions)
   b) Implement module(s) for data (e.g. constants, addresses) obfuscation
   c) Implement module(s) for control flow variations
3. Evaluate the implemented solution:
   a) Implement and execute a pattern matching attack that is able to detect and patch self-checks when obfuscation is not employed
   b) Execute attack based on taint-analysis described in literature [3]
   c) Discuss the performance impact
   d) Analyze and discuss potential security and performance issues of the implemented solution
4. The final thesis document must contain:
   a) Description of the problem and motivation for the chosen approach
   b) State of the art survey, including analysis of security and performance
c) Rationale for choosing certain technique(s) for implementation

d) Implementation description

e) Performance evaluation of implementation

f) Discussion on potential security and performance threats

g) Conclusions and future work

**Deliverables**

- Source code and compiled version of the implementation.
- Instructions on how to run and use the implementation.
- 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**

