Software Protection by Self-Checking

Master thesis

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

Context

Software programs often run on a platform controlled by an attacker who wishes to compromise the program’s integrity by tampering with the binary (e.g. modifying authentication, disabling license checks). One method to ensure software integrity is via self-checking techniques [1], [2]. These techniques use a network of integrity checks called “code-guards” [1] or “testers” [2], which are able to cross-check one-another.

We assume the attacker has the ability to observe where the software checks reside via software binary decompilation and debugging. Afterwards, the attacker is able to create automated attack software which can disable the self-checking integrity protection of other identical instances of the software.

Goal

The goal of this thesis is to design and implement a mechanism based of software self-checking and software-diversity for C# programs. This mechanism must checks the integrity of both the code running in its own assembly, but also the code running in neighboring assemblies. This prevents attackers from simply extracting a .NET assembly from a software solution and reusing it in their own solution.

Another goal is to perform a case-study on real-world software solution offered by Jungheinrich¹, which has agreed to be the industrial partner for this thesis topic. This case-study will include both a performance evaluation and a security evaluation of the implemented self-checking mechanism when applied to a given software solution.

¹ http://www.jungheinrich.com/
Workplan

1. Write state-of-art survey of several self-checking techniques.
2. Implement a self-checking mechanism for C# programs.
   a. Choose one or more self-checking techniques through a coherent argumentation, i.e. security versus performance, which must be presented in written form.
   b. The design & implementation decisions must be documented in written form.
   c. Self-checks can be implemented through C# reflection
   d. Checks must be automatically and randomly inserted in any C# solution.
3. Case-study on real-world software solution
   a. Performance evaluation of the self-checking mechanism itself including memory and runtime overhead
   b. Performance evaluation of the protected software including memory and runtime overhead
   c. Security evaluation of the protected software measured via static analysis and dynamic analysis techniques.
4. The final thesis document must contain:
   a. Description of the problem and motivation
   b. Description of the theoretical background
   c. Implementation description
   d. Performance evaluation of implementation and protected programs
   e. Security evaluation of protected programs
   f. Conclusions and future work.

Deliverables

- Virtual machine able to run a demo of the implementation, including instructions on how to run the demo.
- The VM should also include the source code of 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