Binary based protecting against CRAs

**M.Sc. thesis problem statement**

As the threat potential of Code-Reuse Attacks (CRAs) is rising we want to develop a tool that can mitigate one such state-of-the-art attack dubbed Counterfeit Object-Oriented Programming (COOP). This attack is particularly hard to defend against since traditional Control Flow Integrity (CFI) approaches are useless. Based on binary re-writing techniques during program loading we want to harden a binary in such manner that it becomes very hard for an attacker to perform his attack. At least three types of binary checks will be inserted in the binary. First, parameter counting [7] and type inference [5] for caller/callees pairs will be derived based on a precise per function Control Flow Graph (CFG) analysis. The goal of this analysis is to reduce the pairs of caller/callies to which checks have to be applied in order to keep the application performance overhead low. Second, checks for caller/callee pairs will be added in order to avoid calling non void functions were actually void function were expected and vice-versa. Third, an address taken analysis will be derived which will be used to determine the addresses taken during program run-time. Thus, avoiding usage of not loaded addresses in memory. These three approaches are a refinement of previous defensive strategies and will complement existing binary level techniques. For example parameter type inference will help to reduce the set of caller/callee pairs checks thus increasing the performance of our approach by reducing the number of checks that need to be performed. The tool will be implemented as a series of Dyninst passes and will be tested with real COOP attacks for Linux/Windows OSs. Additionally, for completeness reasons we will test the obtained tool with a series of server applications, web browsers and SPEC CPU 2006 benchmark w.r.t. performance.

**Requirements**

Very good C/C++ programming skills, binary reverse engineering and static analysis knowledge, attacker/defender oriented mindset

**Contact**

Paul Muntean, M.Sc. E-Mail: paul@sec.in.tum.de, 
Tel.: (089) 289-18566, 
Tender date: April 27, 2016, 
Beginning: now
Work Plan

1. Develop knowledge of state-of-the-art binary hardening tools against advanced code-reuse attacks:
   (a) Read references 1–6 and find related work on this topic.
   (b) Identify binary hardening tools suitable for Custom of the Shelf (COTS) binary hardening.
   (c) Write state-of-the-art survey, which presents and compares the investigated techniques and tools.

2. Perform a security analysis of the CRAs mentioned in the reference:
   (a) Identify the binary assets/parts (e.g., indirect calls) that need to be protected.
   (b) Identify, evaluate and perform the attacks to extract the assets from the binary.
   (c) Identify countermeasures based on binary hardening. What do the attacks violate? (e.g., number of parameter sent, type, void function called were non void was expected, etc.)

3. Implement the binary hardening strategy which you identified would make sense in order to prevent the CRAs.
   (a) Choose technique(s) described in literature and/or propose a new technique; argument your choice (e.g. security versus cost trade-off) in written form.
   (b) Implement the chosen technique(s) based on the Dyninst framework \(^1\) and document design decisions.

4. Evaluation of own implementation and possibly existing tools (case-study):
   (a) Measure effectiveness of your hardening against the same attacks identified in step 2.
   (b) Measure performance, effectiveness and size impact of the obfuscation on the SPEC 2006 benchmark, a series of server applications (e.g., Nginx, vftpd, lighttpd, etc.) and web browsers binaries (e.g., Chrome, Firefox, IE).
   (c) Measure the performance of the binary hardening transformation/tool itself.
   (d) Analyze and discuss security versus performance trade-offs.

5. 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) Security analysis of the server applications and web browsers
   (d) Rationale for choosing certain technique(s) for implementation
   (e) Implementation description
   (f) Performance evaluation of implementation
   (g) Discussion on potential security and performance trade-offs
   (h) Conclusions and future work.

Deliverables

1. Source code of the implementation (can be implemented as multiple LLVM refinement passes) as well as instructions on how to run the tool.
2. Technical report with comprehensive documentation of the implementation, i.e., design decision, architecture description, API description and usage instructions.

References


\(^1\)http://www.dyninst.org/