Measuring COOP Attack Surface Reduction

M.Sc. Thesis Problem Statement
Nowadays control-flow hijacking attacks represents the highest software-based security threat [16]. We want to develop a tool that can measure the exact attack surface reduction w.r.t. the attack, Counterfeit Object-Oriented Programming (COOP) [8]. This attack is particularly hard to defend against since traditional Control Flow Integrity (CFI) [1] approaches and hardware based shadow stacks [17] are useless.

In this research we want to measure the real attack surface reduction after a COOP mitigation tool [9] was applied to a program binary by rewriting the binary. The goal of this research is first, to determine how much the attack surface was reduced (available gadgets [18] (assembly code chunks)) quantitatively after binary hardening and second, which gadgets are still available for each indirect call site (qualitatively) before and after hardening.

First, the tool [9] (code available) used for binary hardening will be modified such that it counts the locations in binary code where the tool inserts the checks. Second, we will use a LLVM pass [20] (code available) to detect all available COOP gadgets in the source of an open source application (the same programs as before) by recompiling those with the new pass in place. Third, the source will be compiled with LLVM and DWARF [19] information such that binary code can be easily mapped to source code lines. This information is useful for the previous step. Four, a series of open source gadget finding tools [13, 15, 14, 18] will be used. These will be tailored such that these can be used to detect the COOP gadgets in a binary file and compare those to the previously found gadgets in steps 2 and 3. Thus, the overall idea of these steps is to map the hardened binary parts to source code in order to measure quantitatively (in percent) and qualitatively (per call site) the attack surface reduction w.r.t. COOP.

Finally, for completeness reasons we will test our approach with a series of server applications and web browsers (as in [9]) by measuring the attack surface reduction.

Requirements
Good C/C++ and/or Python script programming skills

Contact
Paul Muntean, M.Sc. E-Mail: paul@sec.in.tum.de, Tel.: (089) 289-18566, Tender date: July 26, 2016, Beginning: now
Work Plan

1. Develop knowledge of the state-of-the-art source/binary code hardening techniques against advanced code-reuse attacks:
   (a) Read references [1, 2, 3, 4, 5, 6, 7, 8, 10, 11, 12, 18, 19] and find related work on this topic.
   (b) Identify source code hardening tools suitable for real life software (e.g., web browsers, servers, etc.).
   (c) Write a state-of-the-art survey (max 5, A4 pages), 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, vTables, class hierarchies (normal and virtual)) that need to be protected.
   (b) Identify, evaluate and perform attacks to extract the assets from the compiled binary.
   (c) Identify countermeasures based on binary hardening. What do the mentioned attacks violate?, (hint: CFI). (e.g., number of function parameters, types, void function called were non void was expected, etc.)

3. Implement your approach in the tools which were mentioned in the MA description.
   (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 LLVM framework 1 and document design decisions.
   (c) Note: we provide a LLVM pass on which the tool can be build upon.

4. Evaluation of own implementation by comparing with the tool presented in the MA description (case-study):
   (a) Measure the attack surface reduction presented in the binary hardening tool [9].
   (b) Measure performance, effectiveness and size impact of the hardening 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), by recompiling these open source apps.
   (c) 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 previous mentioned server applications and web browsers
   (d) Rationale for choosing certain technique(s) for implementation
   (e) Implementation description
   (f) Performance evaluation of the implementation
   (g) Discussion on potential security (potentially high security low performance) and performance (potentially high performance low security) trade-offs
   (h) Conclusions and future work.

Deliverables

1. Source code of the implementation as well as instructions on how to run the tool(s).

2. Technical report with comprehensive documentation of the implementation, i.e., design decision, architecture description, API description and usage instructions.


1http://llvm.org/
References


