Assessing the Attack Surface Reduction in Executables for an Advanced Code Reuse Attack

M.Sc. Thesis Problem Statement

Nowadays control-flow hijacking attacks represents the highest software-based security threat [16]. For this reason we want to develop a tool that can assess the attack surface reduction (Q: Which useful code parts for an attack are still available after a hardening policy was applied to an executable?) w.r.t. the attack dubbed, Counterfeit Object-Oriented Programming (COOP) [8]. This attack is particularly hard to defend against since coarse-grained Control Flow Integrity (CFI) [1] approaches and hardware based shadow stacks, Intel CET [17], are useless. For this reason we want to assess what is protected and what not in an executable w.r.t. the COOP attack.

The goal of this research is twofold: first, we want to determine how much the attack surface was reduced after applying a hardening tool (quantitatively) and second, which COOP gadgets are still available (can be reached) for each indirect call site (qualitatively) before and after hardening (i.e., has caller overestimation and callee underestimation).

First, a provided hardening tool—code available—for binary hardening will be modified such that it logs the locations in a binary where the tool inserts the CFI checks. This helps to determine protected from unprotected code regions. Second (optional), we will use a LLVM pass [20] (code available) to detect all available COOP gadgets in the source code of the same programs as before. Third (optional), the source code will be compiled with DWARF [19] information such that the binary code can be easily mapped to source code lines. Four, a gadget detection tool will be selected from [13, 15, 14, 18] and modified in order to find only COOP gadgets in a binary. The usefulness of COOP gadgets will be assessed w.r.t. indirect call sites, etc. The locations of the gadgets (e.g., start and end address) will be logged with the goal to distinguish between protected and unprotected code parts by first applying a hardening policy. However, the overall goal of these steps is to be able to map the hardened binary parts to source code by using direct and indirect program control flow information in order to assess quantitatively (in percent) and qualitatively (per call site) the attack surface reduction w.r.t. COOP.

Finally, for evaluation means we will test our approach with given: server applications, SPEC2006 test programs, and web browsers.
**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: November 1, 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 where 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 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, vpfd, 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.


References


