Class Hierarchy b. COOP Protection in Binaries

B.Sc./M.Sc. thesis problem statement

Ever wanted to do useful static binary analysis for protecting against many types of advanced code reuse attacks? Well, you are just right, if you want to develop a state-of-the-art binary-based tool which can mitigate against the COOP [4, 5, 6, 7] attack by enforcing a precise forward edge control flow integrity (CFI) [1] policy.

Mitigating against the COOP attack at the binary level can be addressed in many ways and with different levels of precision and tradeoffs. We want to mitigate the COOP attack by reconstructing the class hierarchy from a stripped binary—with no RTTI information—as precise as possible.

The goal of this thesis is to develop a state-of-the-art binary based tool which builds on our tool, TypeShield, in order to protect indirect forward edges in COTS binaries against the COOP attack by enforcing a fine-grained CFI policy based on a previously recovered class hierarchy from a binary.

First, we will extend TypeShield in order to be able to perform a sound analysis of the binary with the goal to recuperate the class hierarchy, similar to what Marx [2] is doing we will impose a series of heuristics to narrow down the location of indirect call sites and virtual tables [3]. Second, we will perform several static analysis (e.g., forward, backward, etc.) in order to gain more certainty about the overall class hierarchy structure. Third, we will recuperate a directed class hierarchy (e.g., class: A → class B) which will provide more information for the derived checks as other similar tools. Finally, the tool will be evaluated using web browsers and the SPEC CPU 2006 benchmark and compared with other state-of-the-art tools.

Requirements

Good C programming skills, disassembly knowledge (X86-64 bit) is beneficial

Contact

Paul Muntean, M.Sc.
E-Mail: paul@sec.in.tum.de,
Tel.: (089) 289-18566,
Tender date: December 14, 2016,
Beginning: now
**Work Plan**

1. Develop knowledge of state-of-the-art class recovery tools from binaries:
   (a) Read provided references and find related work on this topic.
   (b) Test some available tools.
   (c) Write a state-of-the-art survey (max 5 pages, A4 pages), which presents and compares the investigated techniques and tools.

2. Perform a security analysis of the tools found at the previous step w.r.t. COOP.
   (a) Identify alternative binary or software protection techniques against the COOP attack.
   (b) Assess the attack surface reduction in a binary by applying the found tools.

3. Implement the COOP mitigation technique(s) which you identified and you think would make sense.
   (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 tool and document design decisions.
   (c) Note that we provide a runnable tool which can be used for extension.

4. Evaluation of own implementation and possibly existing tools (case-study):
   (a) Measure effectiveness of existing tools against the same programs used in step 2.
   (b) Measure performance, effectiveness, true positives and false negatives of the analyzed software w.r.t.: the SPEC 2006 benchmark, a series of server applications (e.g., Nginx, vftpd, lighttpd, etc.) and web browsers (e.g., Chrome, Firefox, IE) and Linux kernel by hardening this software against the COOP attack.
   (c) Compare the results w.r.t. precision, effectiveness w.r.t. other state-of-the-art tools.
   (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 previous mentioned server applications and web browsers.
   (d) Design, rationale for choosing certain technique(s) for implementation.
   (e) Implementation description.
   (f) Evaluation w.r.t. performance, precision, effectiveness, etc.
   (g) Discussion on potential security and performance trade-offs.
   (h) Conclusions and future work.

**Deliverables**

1. Source code of the implementation (i.e., can be implemented as one or multiple DynInst refinement passes or if desired other frameworks can be used e.g., PIN etc.) 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**