Source code based protection 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 such state-of-the-art attacks (e.g., the 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 source code recompilation techniques we want to harden the application binary in such a 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 (e.g., number of parameters and types, void function were non void was expected and vice-versa, etc). First, parameter counting and type for caller/callees pairs will be derived based on a precise per function Control Flow Graph (CFG) analysis. 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, checks based on the virtual and non virtual class hierarchy (obtained for free from LLVM compiler framework) can be derived and enforced. Additionally, the approach [4] based on class interleaving can be improved with differentiating between virtual and normal class hierarchy or complemented with checks which are a mix of the interleaved class hierarchy and function parameter types, number and return values (void or non-void). These three approaches will be a refinement of previous defensive source code based strategies against CRAs and will complement existing source code level techniques. For example parameter type hardening 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 one or more LLVM passes and will be tested with real CRAs for Linux/Windows OSs. Additionally, for completeness reasons we will test the tool (by recompiling) with a series of server applications, web browsers and SPEC CPU 2006 benchmark w.r.t. performance.

Requirements

Very good C/C++ prog. skills, LLVM passes knowledge, attacker/defender oriented mindset

Contact

Paul Muntean, M.Sc. E-Mail: paul@sec.in.tum.de, Tel.: (089) 289-18566, Tender date: September 9, 2016, Beginning: now
Work Plan

1. Develop knowledge of state-of-the-art source code hardening tools against advanced code-reuse attacks:
   (a) Read references [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11] 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 the binary hardening strategy which you identified and you think would make sense in order to prevent against CRAs. Basically any forward edge (e.g., indirect call) attack is a potential candidate for your tool.
   (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 and the interleaved class hierarchy tool [4] on which further development can be performed.

4. Evaluation of own implementation and possibly existing tools (case-study):
   (a) Measure effectiveness of your hardening tool against the same attacks identified in step 2.
   (b) Measure performance, effectiveness and size impact of the hardening on the SPEC 2006 benchmark, a series of server applications (e.g., Nginx, vfp, lighttpd, etc.) and web browsers binaries (e.g., Chrome, Firefox, IE), by recompiling these open source apps.
   (c) Measure the performance of the hardened binary and of the 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 previous mentioned 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 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.

¹http://llvm.org/
References