Lehrstuhl für Sicherheit in der Informatik
Prof. Dr. Claudia Eckert

# Source code and backward Edge-based Protection Against Advanced Code Reuse Attacks

## M.Sc. thesis problem statement

As the, *1)* threat potential of advanced Code-Reuse Attacks (CRAs) is rising [5], *2)* memory corruptions can not be completely ruled out from programs developed in unsafe languages (i.e., C, C++, etc.), and *3)* the Clang+LLVM compiler does not support shadow stacks enforcement such as Intel CET [13], we want to develop a tool that can mitigate such state-of-the-art attacks.

Based on source code recompilation techniques we want to harden the application binary and add the lowest possible performance penalty such that it becomes very hard (or even impossible) for the attacker to perform her attack.

Previous work (e.g., Bounov et al. [4]) focused on securing forward control flow edges (e.g., indirect calls). In this work we want to secure backward edges (i.e., function returns) such that the call/return calling convention is enforced. Range checks should be added before each function return instruction contained in a vulnerable program binary in such a way that only valid addresses can be targeted.

The new developed security technique will be based on prior work in which we precisely secured indirect forward edges. Based on this work we want to implement a novel technique which secures backward edges (i.e., similar to what shadow call stacks do).

The tool will be based on one or more LLVM instrumentation passes and will be tested with real CRAs for Linux/Windows OSs. Additionally, 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

Strong C/C++ programming skills, LLVM pass knowledge is beneficial

## Contact

Paul Muntean, M.Sc. E-Mail: paul@sec.in.tum.de, Tel.: (089) 289-18566, Tender date: September 29, 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, 2, 3, 4, 6, 7, 8, 9, 10, 11, 12] 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 technique which you identified and you think would make sense in order to prevent against CRAs. Basically any backward edge (e.g., return instructions) based 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.
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, vfptd, 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.
3. Final thesis report written in conformance with TUM guidelines.

[1] http://llvm.org/

# References


[1] M. Abadi et al. Control-Flow Integrity. In CCS, 2005.

[2] C. Tice et al. Enforcing Forward-Edge Control-Flow Integrity in GCC & LLVM, In: Usenix Security Symposium, 2014. (vtv tool)

[3] I. Haller et al. ShrinkWrap: VTable Protection without Loose Ends, In: ACSAC 2015.

[4] D. Bounov et al. Protecting C++ Dynamic Dispatch Through VTable Interleaving, In: NDSS 2016.

[5] Oliveira et al. An Information Flow-based Taxonomy to Understand the Nature of Software Vulnerabilities, In: IFIP SEC 2016.

[6] D. Jang, et al. SafeDispatch: Securing C++ virtual calls from memory corruption attacks, In: NDSS 2014.

[7] C. Zhang, et al. Vtint: Protecting virtual function tables' integrity, In: NDSS 2015.

[8] C. Zhang, et al. VTrust: Regaining Trust on Virtual Calls, In: NDSS 2016.

[9] F. Schuster et al. Counterfeit Object-oriented Programming: On the Difficulty of Preventing Code Reuse Attacks in C++ Applications, In S&P, 2015

[10] S. Crane et al. It's a TRaP: Table Randomization and Protection against FunctionReuse Attacks. In CCS, 2015.

[11] I. Evans et al. Control Jujutsu: On the Weaknesses of Fine-Grained Control Flow Integrity. In CCS, 2015.

[12] B. Lan, et al. Loop-Oriented Programming: A New Code Reuse Attack to Bypass Modern Defenses, In: IEEE Trustcom 2015.

[13] Intel CET `https://blogs.intel.com/evangelists/2016/06/09/intel-release-new-technology-specifications-protect-rop-attacks/` and `http://www.theregister.co.uk/2016/06/10/intel_control_flow_enforcement/`, Jun. 2016.