Implementing Obfuscations as LLVM Passes for Drop-In Obfuscation of Open-Source Projects

Modern research on reverse engineering and binary analysis requires large datasets, e.g. for developing machine learning algorithms, and the corresponding ground truth. Generating these datasets is typically done by collecting open source repositories and compiling them with the needed compiler settings. However, one direction of binary analysis research is the analysis of obfuscated code, i.e. code that has been rewritten with the goal of obscuring its original intention, in order to protect intellectual property or hide malicious intent. Applying obfuscations at scale is not trivial and one project to fill this niche is Obfuscator-LLVM.

Obfuscator-LLVM is an extension of the LLVM compiler, allowing usage as a drop-in replacement for everything that can be compiled using LLVM. It has seen wide adoption in its research field; however, it was released in 2015 and has not seen any significant updates in recent years. There have been projects building on Obfuscator-LLVM, such as Hikari or an Obfuscator-LLVM Clang Plugin, which also have not received significant updates, with the build process of Hikari and Obfuscator-LLVM already being a significant hurdle in using them.

As we rely on being able to generate large datasets and their ground truth automatically for our research, we would like to improve the current situation of Obfuscator-LLVM or obfuscation with the help of LLVM in general. This undertaking consists of the following steps:

1. Identify existing solutions that are more recent than Obfuscator-LLVM and can easily be built and applied on modern projects.
2. Bring Obfuscator-LLVM (or other more recent projects) to current LLVM versions. We should weigh the pros/cons of maintaining an LLVM fork vs. using Clang plugins. The resulting project should ideally be extensible and portable to newer LLVM versions.
3. Implement the “classic” Obfuscator-LLVM obfuscations within our project; verify the correct application of these obfuscations and the correctness of the produced programs.
4. Add newer, stronger obfuscations comparable to the obfuscations provided by the Tigress obfuscator.

We are interested in motivated students who would like to tackle these challenges within a bachelor or master thesis. You should bring the following skills or accustom yourself to the following topics:

- C++ programming
- Compiler construction
- Reverse engineering of ELF binaries in general, e.g. with the help of Ghidra, in order to verify manually (and later automatically) whether obfuscations have been applied

If you are interested in this topic, please reach out to:

- Dr. Corinna Schmitt (UniBW, Examiner at the LMU), Email: corinna.schmitt@unibw.de
- Dr. Georg Merzdovnik (Senior Researcher at SBA Research), Email: gmerzdovnik@sba-research.org
- Dipl.-Ing. Christian Kudera (Researcher at SBA Research), Email: ckudera@sba-research.org
- Dipl.-Ing. Michael Pucher (Researcher at SBA Research), Email: mpucher@sba-research.org