Searching similar functions using the LLVM intermediate representation

Degree programme: Master of Science in Engineering | Specialisation: Information and Communications Technologies
Thesis advisor: Prof. Dr. Endre Bangerter
Expert: Jonas Wagner (Threatray)

The search for similarities in binary codes is an evolving area of active research, with multiple application areas ranging from vulnerability search to malware classification. In this project we propose a scalable solution using similarity preserving-hashes and LLVM intermediate representation to search for similar functions in a large amount of compiled programs.

Julien Farine

Need of similarity search

A new vulnerability has been found in a function of a popular open source cryptographic library. An attacker is now able to exploit programs using the library in order to execute malicious code. Is there a vulnerable program on your system? Answering this question is crucial for the security of your system, but the investigation is not trivial. Indeed, you do not have access to the source code of all the programs installed on the system, moreover the developers of closed source software programs do not necessarily publish the list of libraries they use. The only information accessible is the binaries of installed programs and the vulnerable cryptographic library. In such a scenario, answering to the initial question becomes complex. To overcome such issues, we propose a system able to search for similar functions in a large set of binary programs, allowing to retrieve binaries containing a given function.

Binary codes similarity challenges

One of the major challenges when searching for similar functions between binaries is to be able to handle the effects of compilers on source code. Indeed, the same program, compiled with different compilers, different versions of compilers or different levels of optimisation, produces widely different binary representations. Thus, an exact match between compiled codes will fail to retrieve different binary code representations for the same source code, and a more advanced similar search system is therefore needed.

Using the LLVM intermediate representation

Rather than designing our similarity search framework to work directly with binary programs, we first lifted the binaries into the LLVM intermediate representation (IR). The LLVM IR is a low-level language using a reduced instruction set especially built to support high-level analysis and transformations. Hence, it makes the LLVM IR a valuable tool for function similarity search. The smaller instruction set, compared to the assembly one, gives less possibilities to represent the same operations, which increases similarities between codes. In addition it is easier to extract information from the LLVM IR than from the assembly, and the LLVM tools are well documented and provide access to compiler optimisation passes, which can be useful for reducing the compilers effects.

Feature extraction and performances

Once the binaries have been lifted, we extract several features from functions, all in the form of a MD5 hash. Then, features are added together to create similarity-preserving hashes using the SimHash algorithm. Thus, we can measure similarity between functions by computing the Hamming distance between their SimHashes (the hamming distance is the number of different bits between two hashes). This allows our system to scale given searches in hamming space can be done efficiently on large amount of data.
The SimHash method we use is inspired by an open source project of Thomas Dulien called „Searching statically-linked vulnerable library functions in executable code", which is interesting, as T. Dulien does not use the LLVM IR, it allows us to compare our results and highlight the effect of the LLVM intermediate representation.
With the same set of features as T. Dulien, on the same data, for an irrelevant result rate of 5%, the LLVM IR presents an improvement of true positive rate from 41% in the Dulien's work to 55% in our. This means that the LLVM IR actually increases the similarities.
Moreover we also tested our system in a malware classification use case by searching similar functions in already classified malwares. The system was able to correctly classify over 95% of the tested samples.