Extracting structure from low-level code

PhD proposal

Advisors: Igor Walukiewicz, Olivier Ly, Aymeric Vincent

Software analysis and verification is an active area of research, but at present its applicability is much less spread than that of harware verification. If bug finding is concerned, some spectacular results have already been achieved as demonstrated for example by Coverity Prevent [BBC+10]. Tools for more complex analysis and verification of code exist but do not yet scale up to big examples. Almost all work until now has concentrated on high level programming languages such as C or Java. This subject proposes to concentrate on low-level code, for example: binary code, or a mix of C and assembly code.

Analysing or even understanding low-level code is difficult. First, the control structure is not explicit: not only the control is implemented by jumps, but moreover these jumps can be dynamic. Next, modelling memory access in low-level code is a necessity because we cannot rely on abstract memory access mechanisms like variables, arrays, or structures.

Yet the task of understanding and analysing low level code is important. Let us give two examples. Critical parts of the operating system kernel are often written in a mix of C and assembly language. For example signal delivery or lock management. Correctness of these parts of the code is obviously crucial. With the arrival of poweful mobile devices, almost everybody becomes dependent on the correctness of, several, operating systems; sometimes highly optimised for a particular hardware. The other application is analysis of computer viruses. Here, the goal is not verification but rather analysis and extraction of the structure. This is all-important to speed up an analysis of a virus and an evaluation of its potential threat.

In this thesis, we want to investigate two main aspects of low-level code: (i) extracting a control flow graph, and (ii) reconstructing memory layout. For the first aspect, the main novelty we want to address with repsect to existing works [BGRT05, KV08] is detection of functions and function calls in code not produced by a compiler. The research in the second aspect is less advanced [BR07]. It will be necessary to develop new memory models and provide algorithms constructing model descriptions from low-level code. The subject offers possiblities of both theoretical work, and developement of prototypes for the two areas of applications mentioned above.

The research on these topics in LaBRI has already started in the framework of the ANR-funded project BINCOA.

References

[BBC+10] Al Bessey, Ken Block, Ben Chelf, Andy Chou, Bryan Fulton, Seth Hallem, Charles Henri-Gros, Asya Kamsky, Scott McPeak, and Dawson Engler. A few billion lines of code later: using static analysis to find bugs in the real world. *Commun. ACM*, 53(2):66–75, 2010.

[BGRT05] Gogul Balakrishnan, Radu Gruian, Thomas W. Reps, and Tim Teitelbaum. Codesurfer/x86-a platform for analyzing x86 executables. In *CC*, pages 250–254, 2005.

[BR07] Gogul Balakrishnan and Thomas W. Reps. Divine: Discovering variables in executables. In *VMCAI*, pages 1–28, 2007.

[KV08] Johannes Kinder and Helmut Veith. Jakstab: A static analysis platform for binaries. In *CAV*, pages 423–427, 2008.