Abstract

Runtime data-flow analysis of binary programs for security applications

Ulf Kargén
IDA, Linköping University
ulf.kargen@liu.se
Supervisor: Nahid Shahmehri

With our increasing dependence on IT systems, both in society and in our everyday lives, the threat of software vulnerabilities is an ever-increasing concern. To help combat this threat, many code analysis methods have been suggested to help finding security bugs in software. Most of these techniques require access to the source code of the program, which has mostly restricted their use to in-house testing. In recent years, however, there has been a growing interest in security analysis techniques that can work on off-the-shelf machine code programs. There are many reasons for wanting to perform analysis on program binaries; in some cases bugs are only present in the final executable, such as when program errors have been introduced by compiler bugs or unexpected compiler optimizations. Malware analysts are also often forced to perform their analysis on the machine code of malicious software, since source code is rarely available. Another application of binary analysis is third-party security auditing of closed-source programs, which has become an important part of a defense-in-depth approach to software security for many companies.

Traditionally, analysis of program binaries has been performed mostly through manual reverse-engineering of the machine code. Since most of the high-level semantics of the original source code is lost during compilation, this is an extremely time-consuming and error-prone task. Due to these challenges, many techniques have been suggested to aid in, or partially automate, various kinds of program analysis of binaries. One such popular technique is dynamic taint analysis (DTA). DTA is a form of data-flow analysis, which can be used to track if and when data derived from some specific source is used in some program location that is interesting for the analysis. DTA can, for instance, be used to collect detailed information on how some input data is used in specific parts of a program. Such information is particularly interesting for many types of security analyses, since many security problems stem from user-controllable data being used in an insecure way.

Most existing DTA implementations are designed for some specific type of automated analysis, but DTA could also be used as a powerful aid in manual program comprehension tasks. We have developed a prototype taint analysis tool for aiding in manual program analysis of unmodified x86 binaries under Linux. The tool can be used to uncover details on how a program's input affects its behavior, by creating a report on all instances where data derived from input was used in program-flow altering instructions or as inputs to system calls. By providing detailed information on which bytes of input this data was derived from, the tool can aid in various program comprehension tasks, such as debugging, reverse-engineering or security auditing. One particular application is root-cause analysis of bugs found via fuzz testing, especially when source code is not available for the tested program.

Currently, we are developing a new version of the tool, with the goal of improving its performance and usefulness by utilizing a form of limited dynamic program slicing. This new version is also intended to serve as a platform for future work, e.g. for building tools to automatically detect vulnerabilities in executables.

During the presentation, the basic concepts and terminology of dynamic taint analysis of binaries will be introduced, along with a discussion of some of the challenges in developing a practical taint analysis system. The prototype tool will be presented and an overview of the current work on the new version of the tool will be given. I will conclude with a brief discussion of future work and potential future applications of the taint analysis system.