Chapter 47
Combining Static Code Analysis and Machine Learning for Automatic Detection of Security Vulnerabilities in Mobile Apps

Marco Pistoia
IBM Corporation, USA

Omer Tripp
IBM T. J. Watson Research Center, USA

David Lubensky
IBM T. J. Watson Research Center, USA

ABSTRACT

Mobile devices have revolutionized many aspects of our lives. Without realizing it, we often run on them programs that access and transmit private information over the network. Integrity concerns arise when mobile applications use untrusted data as input to security-sensitive computations. Program-analysis tools for integrity and confidentiality enforcement have become a necessity. Static-analysis tools are particularly attractive because they do not require installing and executing the program, and have the potential of never missing any vulnerability. Nevertheless, such tools often have high false-positive rates. In order to reduce the number of false positives, static analysis has to be very precise, but this is in conflict with the analysis' performance and scalability, requiring a more refined model of the application. This chapter proposes Phoenix, a novel solution that combines static analysis with machine learning to identify programs exhibiting suspicious operations. This approach has been widely applied to mobile applications obtaining impressive results.

DOI: 10.4018/978-1-5225-3422-8.ch047

INTRODUCTION

Mobile devices have revolutionized many aspects of our lives. We use smartphones, tablets and wearable devices as portable computers and, often without realizing it, we run various types of security-sensitive programs on them, such as personal and enterprise email and instant-messaging applications, as well as social, banking, insurance and retail programs. These applications access and transmit over the network numerous pieces of private information, including our geographical location, device ID, contacts, calendar events, passwords, and health records, as well as credit-card, social-security, and bank-account numbers. Guaranteeing that no private information is exposed to unauthorized observers is very challenging, given the level of complexity that these applications have reached. Integrity concerns arise when mobile applications take untrusted user data as input to security-sensitive computations. In such cases, in order to avoid integrity violations, it is necessary to verify that the appropriate validation and/or sanitization routines are invoked. Program analysis tools for integrity and confidentiality enforcement have become a necessity, especially for mobile applications, which are updated very often and require high security assurance. Roughly speaking, such tools can be either "dynamic" or "static". Dynamic-analysis tools execute the program and infer security vulnerabilities based on the actual program executions. Such solutions can be quite time consuming because they require installing the program under analysis and then executing it multiple times, with different inputs and different execution choices on each run, in order to maximize the number of execution paths explored by the analysis. Often, however, there is no real guarantee that all the possible paths of execution have been explored. This means that some security vulnerabilities may remain undiscovered by the time the application is released. Conversely, a static-analysis tool does not require installing or executing the program. Rather, a formal model that over-approximates all feasible program executions is built, and the analysis reports results based on that model. Unlike dynamic analysis, static analysis has the potential of never missing any vulnerability, but it may report false positives; that is, it might signal security issues that are never exposed at run-time. Some analysis tools, even at the commercial level, have very high false-positive rates. Numerous user studies have proved that an analysis whose false-positive rate is too high becomes unusable because developers are forced to spend large amounts of time to filter out spurious issues, and when the false-positive rate is excessive, the effort becomes unbearable. Therefore, in order to reduce the number of false positives, a static analysis has to be very precise. Unfortunately, however, precision is in conflict with the analysis' performance and scalability, requiring a more refined model of the application and analysis thereof. A large body of research work has studied how to increase the precision of static-analysis tools without affecting their scalability. At the same time, applications have become increasingly more complex, with the addition of frameworks and dynamically loaded components, thereby making the compromise between precision and scalability even more difficult to reach. This chapter proposes Phoenix, a novel solution that combines static program analysis with machine learning. The idea behind Phoenix is to use relatively scalable static analysis to approximate possible program behaviors, and to then apply machine learning in order to identify programs exhibiting suspicious sequences of operations. This solution has been widely applied to mobile applications obtaining impressive results, with low false-positive and false-negative rates.

Phoenix comprises two components: a novel static analyzer, which performs demand-driven pointer analysis and lends itself to modular reasoning, and a machine-learning engine, which acts on the results of the static analyzer and filters out the majority of the false positives, while retaining almost the totality of the true positives. These two components are described in the remainder of this chapter.