Design Churn as Predictor of Vulnerabilities?

Aram Hovsepyan, iMinds-DistriNet, KU Leuven, Leuven, Belgium
Riccardo Scandariato, iMinds-DistriNet, KU Leuven, Leuven, Belgium
Maximilian Steff, Free University of Bozen, Bolzano, Italy
Wouter Joosen, iMinds-DistriNet, KU Leuven, Leuven, Belgium

ABSTRACT

This paper evaluates a metric suite to predict vulnerable Java classes based on how much the design of an application has changed over time. It refers to this concept as design churn in analogy with code churn. Based on a validation on 10 Android applications, it shows that several design churn metrics are in fact significantly associated with vulnerabilities. When used to build a prediction model, the metrics yield an average precision of 0.71 and an average recall of 0.27.

Keywords: Android Applications, Machine Learning, Security Vulnerability Prediction, Software Metrics

1. INTRODUCTION

Security vulnerabilities are a serious threat to any organization as an exploit can cause severe monetary and reputation damage. It is essential to detect and mitigate software vulnerabilities before the software product is released. Verification and validation activities, such as security testing and code review are effective means in reducing the number of post-release vulnerabilities. However, such quality assurance is not only inexpensive, but it is also best done by engineers specifically trained in software security (McGraw, 2006). Hence, tools and techniques that can help identify components that are more likely to contain vulnerabilities can provide substantial support to the security engineers who can focus their attention and efforts on higher risk components.

One of the possible approaches to predict vulnerable components is to build statistical models using software metrics. Historically, prediction models based on software metrics are known to be very effective in defect prediction (e.g., Basili, 1996; Menzies, 2007; Nagappan, 2005). Since recently, various studies have investigated the effectiveness of vulnerability prediction models based on software metrics. As opposed to defect prediction, vulnerability prediction is much more complicated as vulnerabilities are typically few in number. Nonetheless, various studies have demonstrated the effectiveness of vulnerability prediction models based on software metrics.
prediction models based on mutually complementary set of software metrics. A number of works has investigated the predictive power of implementation-level code measures, such as size and complexity (Shin, 2011; Chowdhury, 2011). Design-level measures, such as coupling, dependencies between components, were observed to be efficient especially in terms of recall (Zimmermann, 2010; Shin, 2011). The afore-mentioned measures are static in the sense that they consider a software system at a specific point in time. Recent works have shifted their focus towards evolutionary measures, such as code churn, which is a measure of the amount of code changed within a software unit over time. The evolutionary measures could provide an even higher performance than static measures (Shin, 2011). However, there is a clear lack of research with respect to the use of evolutionary design-level measures in the domain of vulnerability prediction.

This article focuses on evolutionary design-level measures. The contribution of this paper is an exploratory study of whether the changes to the dependency structure of a software system could be used as predictors of vulnerable software components. We consider a software system as a graph where nodes represent classes and directed edges represent dependencies between the classes. We then compare the changes to the dependency graphs across different versions of a software system. We refer to this metric as design churn in analogy with code churn. Our previous work has shown that design churn metrics are excellent predictors of defects (Steff, 2011). We now apply design churn metrics to the domain of security and perform a validation on ten Android applications. In this paper, we show a statistically significant association between design churn and security vulnerabilities. We also build a vulnerability prediction model based on design churn that provides good performance in terms of precision (0.71 on average), but low recall values (0.27 on average).

The rest of the article is organized as follows. In section 2, we present the overall research methodology including the application selection, the goals of our investigation and the exact experiment setup. In sections 3 and 4, we present and discuss the results and describe the most important threats to validity. In section 5, we describe the related work. Finally, section 6 presents the conclusions and provides an overview of the future work.

2. RESEARCH METHODOLOGY

In the context of our study, we experiment on ten Android applications, where each application is represented by a set of Java classes. Each class is then regarded as vulnerable if there is at least one vulnerability in the code, as reported by a static code analyzer. Otherwise the class is considered as clean.

The overall goal of this study is twofold. First, we investigate whether there is an association between design churn metrics and vulnerable classes. Second, we build a vulnerability prediction model based on design churn metrics and evaluate its predictive power. To encourage the replication of this study, all the experimental materials are available online, including the data we have used, description of the techniques we have used to extract the presented metrics, etc. (KULeuven, 2013).

2.1. Design Churn (Independent Variable)

Each application release can be represented as a directed graph where classes represent the nodes, while dependencies between these classes represent the edges. We define design churn as changes to these graphs across different releases of an application. Before identifying and measuring structural change, first we have to define and extract the structure itself from Java source code. Note that in the context of this work we do not consider the AndroidManifest.xml file although it could contain essential information relevant to vulnerabilities. We partially compile the Java source code and we extract the dependencies from the resulting byte-code. Dagenais and Hendren (Dagenais, 2008) presented PPA, a tool for compiling and
An Analysis of the Internal Consistency of the New Accounting Standard for Virtual Currencies in Generally Accepted Japanese Accounting Principles: A Virtual Currency User Perspective

Building Secure Software Using XP
[www.igi-global.com/chapter/building-secure-software-using/72204?camid=4v1a](www.igi-global.com/chapter/building-secure-software-using/72204?camid=4v1a)

Design Patterns and Design Principles for Internal Domain-Specific Languages
[www.igi-global.com/chapter/design-patterns-design-principles-internal/71820?camid=4v1a](www.igi-global.com/chapter/design-patterns-design-principles-internal/71820?camid=4v1a)