EVALUATION OF MACHINE LEARNING ALGORITHMS IN THE CATEGORIZATION OF ANDROID API METHODS INTO SOURCES AND SINKS

By

WALBER DE MACEDO RODRIGUES

B.Sc. Dissertation Proposal

Federal University of Pernambuco
secgrad@cin.ufpe.br
www.cin.ufpe.br/~secgrad

RECIFE/2018

Resumo

Um programa computa em dados sensíveis e não sensíveis, esses dados seguem um fluxo específico indo de *data sources* para *data sinks*. O vazamento de dados acontece quando dados sensíveis chegam sem autorização em *sinks*, para prevenir isso, técnicas estáticas e dinâmicas de *Flow Enforcement* garantem que esses dados não cheguem nessas *sinks*. Para isso, esses métodos usam listas, geradas manualmente, de métodos que sejam *sources* sensíveis ou *sinks*, e essa solução é impraticável para grandes APIs como a do Android. Visto isso, uma abordagem usando *machine learning* foi desenvolvida para classificar esses métodos *sources* e *sinks*. O presente trabalho tem como objetivo criar um dataset para avaliar os métodos de classificação mais utilizados e decidir quais os mais apropriados para esse problema.

Abstract

A program computes in either sensitive and non-sensitive data and follows a specific flow, from data sources to data sinks. Data leakage happens when sensitive data is sent to unauthorized data sinks, to prevent that, Dynamic and Static Flow Enforcement techniques ensure that sensitive data reaches those sinks. To prevent data leakage, these methods rely on a list of sensitive data sources and data sinks, this list is hand annotated and is impractical to be made to a huge API such as Android. With that in mind, a machine learning model is used to classify methods into sources and sinks. The present work intends to extend the previous work creating a dataset to evaluate the most used classification algorithms and define which is the most suitable to this problem.

1

Introduction

Every program computes on either sensitive data or non-sensitive data. Sensitive is any data that can be used to identify the user or any private user information, such as photos, International Mobile Equipment Identity (IMEI) nd biometric data. Non-sensitive data is any dynamic information that do not identify the user, oftenly this kind of information if public or shared, such as application source code.

In an application, data follows a specific flow, first it is acquired from data sources and sent to data sinks (MCCABE, 2003). Data sources, in the context of mobile and IoT devices, are defined as method calls that read data from shared resources such as phone calls, screenshots, sensor polling data from ambient, device identification numbers (RASTHOFER; ARZT; BODDEN, 2014). Data sinks are methods calls that have at least one argument, this argument is non-constant data from the source code (RASTHOFER; ARZT; BODDEN, 2014). The data sink can be an interface to the user or system API for communication to other devices or store data (VIET TRIEM TONG; CLARK; MÉ, 2010).

Dynamic Flow Enforcement are techniques that tracks and enforces information flow during the application runtime. These methods relies on Taint Analysis to track possible sensitive data flow to untrusted sinks. Taint Analysis marks every sensitive data gathered from a source and every other variable that inherit any operation from the tainted data, in the end, if any tainted variable is accessed by a sink method, the information has leaked and the analysis gives a detailed path through which the data passed. During the tracking, there are different methods to enforce in runtime that the data will not leak, FERNANDES et al. (2016) uses virtualization to guarantee that the data will only operate in the controlled environment and SUN et al. (2017) declassifying information before it is computed in trusted methods or if reach a trusted API.

Static Flow Enforcement starts by creating abstract models of the application code to provide a simpler representation (MYERS, 1999), using frameworks like Soot (VALLÉE-RAI et al., 2000). Then, this model will be used in control-flow, data-flow and points-to analysis to observe the application control, data sequence and compute static abstractions for variables LI et al. (2017). These methods are implemented and used in DroidSafe GORDON et al. (2015). JFlow MYERS (1999) inserts statically checked and secured code when the application computes on sensitive data.

Both Static and Dynamic Flow Enforcement techniques require information of which methods is a source of sensitive data and which is a data sink. This is used to identify if a sink method is truly leaking sensitive data or not. So, lists containing sources and sinks of sensitive data are hand created, but this solution is impractical considering a huge API like the Android API RASTHOFER; ARZT; BODDEN (2014).

Considering that issue, Rasthofer et al. RASTHOFER; ARZT; BODDEN (2014) proposed to use machine learning to automatically create a categorized list of sources and sinks methods to be used in Flow Enforcement techniques. The list consists in methods classified into Flow Classes and Android Method Categories. The Flow Classes are source of sensitive data, or just source, and sink of data, but also, the method can be neither source or sink. For Android Methods Categories, there are 12 different classes: account, Bluetooth, browser, calendar, contact, database, file, network, NFC, settings, sync, a unique identifier, and no category if the method does not belong to any of the previous.

The authors shortly compared Decision Trees and Naive Bayes with the SVM and choosed to use SVM to create the categorized list of sources and sinks, as SVM showed to be more precise in categorize the Android methods.

To classify, the authors utilize features extracted from the methods, like the method name, if the method has parameters, the return value type, parameter type, if the parameter is an interface, method modifiers, class modifiers, class name, if the method returns a value from another source method, if one parameter flows into a sink method, if a method parameters flows into a abstract sink and the method required permission.

To categorize the methods, were used features like class name, method invocation, body contents, parameter type and return value type. After that, the methods list is generated containing if it is a sink, source and the method category.

2

Objectives

This work has two main objectives: generate a public database containing methods from Android APIs and evaluate the performance of the most used classification algorithms on this database. The methods should be classified into any of the two Flow Classes, source or sink, or neither.

The motivation behind the database creation comes when RASTHOFER; ARZT; BODDEN (2014) developed the initial classification work. Their objective were create a machine-learning solution to identify sources and sink methods from the code of any Android API. So, every time that you wanted to test other classification algorithms, you had to extract the methods and features at every execution. Then, create a public database extract knowledge and develop better solutions to this problem is very likely.

The objective of evaluate different classification algorithms comes as RASTHOFER; ARZT; BODDEN (2014) shows results for only three classification algorithms, SVM, Decision Tree and Naive Bayes. From that, emerged the question if other classification algorithms have better results in this database. So, the second objective is to evaluate the most used classifiers in the literature, SVM, Naive Bayes, Decision Trees, MLP, KNN, including ensemble classifications methods.

3

Methodology

As first objective, the dataset will be created using the feature extractor developed by RASTHOFER; ARZT; BODDEN (2014). The extractor creates meaningful information using the Android methods names and their real implementation in the Android API. As result, the extractor gives the method class, if has been hand annotaded, and a list of features. These features are semantic and syntactic features, containing information about the method name, parameters, return type, method and classes modifiers, class modifiers, if exists data flow in the method return or parameters and the required permissions.

The extractor will be used in many Android APIs as possible, since there are a low quantity of hand annotated methods, it is important to extract as many methods from classes as possible. It is possible to have duplicated methods at the end of this evaluation due to backward compatibility, so, methods from older APIs will be overwritten.

The methods used in the comparison will be SVM, Naive Bayes, Decision Trees, MLP, KNN, and ensemble classifications methods, which will be evaluated in 30 datasets sampled from the original. Each of these datasets are subdivided into train dataset and test dataset, containing 80% of the original dataset for model training and 20% for test the model effectivity. They must maintain the classes proportion observed in the original dataset, if the original has 45% of source methods, the train and test must have a proportion close to that. Create these datasets are important to make a Hypothesis Test, this help to statically evaluate if a classifier is really effective when applied in this dataset. The proposed evaluation flow is shown in figure 3.1.

$$precison = \frac{TP}{TP+FP} \quad (3.1) \qquad recall = \frac{TP}{TP+FN} \quad (3.3)$$

$$accuracy = \frac{TP+TN}{TP+TN+FP+FN} \quad (3.2) \qquad F1 = \frac{2 \times recall \times precision}{recall + precision} \quad (3.4)$$

Each classifier will be evaluated using precision, accuracy, recall and F1 score. The precision, equation 3.1, is the ratio of correctly predictions to the total predictions done. Accuracy, equation 3.2, is the ratio of correctly true predictions to the total of true predictions. Recall, equation 3.3 is the ratio of correctly positive predictions to all the predictions of a class. F1, equation 3.4, score represents the harmonic mean between precision and recall SASAKI et al. (2007).

We can rewrite precision, accuracy, recall and F1 score in the equations above, using true positives (TP), true negatives (TN), false positives (FP) and false negatives (FN). True positives are all instances of a class C that are correctly classified. True negatives are all instances that do not belong to C and are correctly classified. False positives of a class C are the instances of other classes that has been classified as C. False negatives are instances of C that has been classified as not belonging to C.

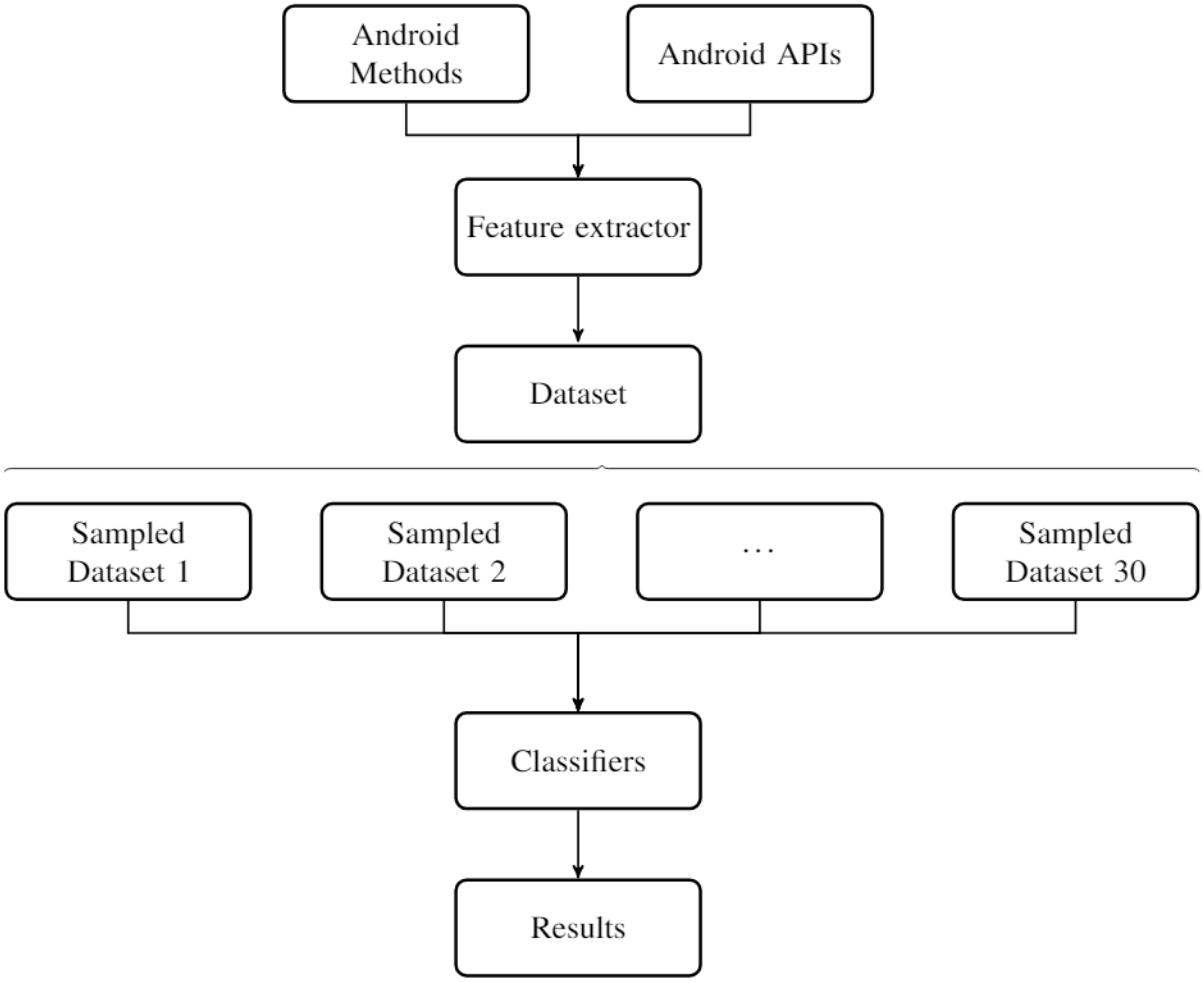

Figure 3.1: Overview of the proposed evaluation flow. First, the feature extractor uses Android Methods and Android APIs to create a dataset, which will be sampled into other 30 different datasets that will be used to evaluate the Classifiers.

4

Schedule

	Period															
Activity	August		September				October				November				December	
Literature Review	x	x	x													
Dataset creation			x	x	x	x										
Algorithm studies						x	x	x								
Experiments							x	x	x	x						
Evaluation									x	x	x					
Writing										x	x	x	x	x	x	
Presentation preparation															x	x

5

Possível Avaliador

Paulo Salgado Gomes de Mattos Neto

References

FERNANDES, E. et al. FlowFence: practical data protection for emerging iot application frameworks. In: USENIX SECURITY SYMPOSIUM. **Anais...** [S.l.: s.n.], 2016. p.531–548.

GORDON, M. I. et al. Information Flow Analysis of Android Applications in DroidSafe. In: NDSS. **Anais...** [S.l.: s.n.], 2015. v.15, p.110.

LI, L. et al. Static analysis of android apps: a systematic literature review. **Information and Software Technology**, [S.l.], v.88, p.67–95, 2017.

MCCABE, J. D. **Network Analysis, Architecture and Design, Second Edition (The Morgan Kaufmann Series in Networking)**. [S.l.]: Morgan Kaufmann, 2003.

MYERS, A. C. JFlow: practical mostly-static information flow control. In: ACM SIGPLAN-SIGACT SYMPOSIUM ON PRINCIPLES OF PROGRAMMING LANGUAGES, 26. **Proceedings...** [S.l.: s.n.], 1999. p.228–241.

RASTHOFER, S.; ARZT, S.; BODDEN, E. A Machine-learning Approach for Classifying and Categorizing Android Sources and Sinks. In: NETWORK AND DISTRIBUTED SYSTEM SECURITY SYMPOSIUM NDSS. **Anais...** [S.l.: s.n.], 2014.

SASAKI, Y. et al. The truth of the F-measure. **Teach Tutor mater**, [S.l.], v.1, n.5, p.1–5, 2007.

SUN, C. et al. Data-Oriented Instrumentation against Information Leakages of Android Applications. In: IEEE 41ST ANNUAL COMPUTER SOFTWARE AND APPLICATIONS CONFERENCE (COMPSAC), 2017. **Anais...** [S.l.: s.n.], 2017. p.485–490.

VALLÉE-RAI, R. et al. Optimizing Java bytecode using the Soot framework: is it feasible? In: INTERNATIONAL CONFERENCE ON COMPILER CONSTRUCTION. **Anais...** [S.l.: s.n.], 2000. p.18–34.

VIET TRIEM TONG, V.; CLARK, A. J.; MÉ, L. Specifying and enforcing a fine-grained information flow policy: model and experiments. **Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications**, [S.l.], v.1, n.1, p.56–71, 2010.