Poster : Predicting Vulnerable Software Components via Text Mining

Sanjana Kacholia
North Carolina State University
Raleigh, North Carolina
skachol@ncsu.edu

ABSTRACT

This paper presents an approach based on text mining the source code of components to predict which software contain security vulnerabilities. Namely, each component is characterized as a series of terms contained in its source code, with the associated frequencies. These features are used to forecast whether each component is likely to contain vulnerabilities. In an exploratory validation with 20 Android applications, we discovered that a dependable prediction model can be built. Such model could be useful to prioritize the validation activities, e.g., to identify the components needing special scrutiny.

1 INTRODUCTION

Verification and validation (VV) techniques like security testing, code review and formal verification are becoming effective means to reduce the number of post release vulnerabilities in software products. This is an important achievement, as fixing a bug after the software has been released can cost much more than resolving the issue at development time. However, it is not inexpensive. An early estimation assessed that VV adds up to 30 percent to development costs. In this respect, being able to predict where the vulnerabilities are likely to show up in a code base provides the opportunity to point VV activities in the right direction and to manage and optimize cost.

2 RELATED WORK

In order to compare the results of these studies with each other and with the paper, five dimensions are defined along which each related work has been characterized: the type of prediction features used, the source of the vulnerability data, the type of prediction technique, the applications used for the validation, and the available performance indicators.

3 RESEARCH GOALS

Overall research goal is to build a prediction model in the form of a binary classifier that can predict whether a software component is likely to be vulnerable.
RQ1: Can a prediction model be built?
RQ2: Can predictions in the future be made?
RQ3: Can cross-project predictions be made?
In the context of security, recall is often considered to be more important because, in general, it is preferable that positives are not disregarded. Therefore, F2 is a more useful indicator as it weights recall higher than precision. Our selection criteria for applications included the programming language, application size, and the number of versions released. Machine learning techniques focused in

Listing 1. Source code in file HelloWorldApp.java

```
/* The HelloWorldApp class
prints "Hello World!" */
class HelloWorldApp {
  public static void main(String[] args) {
    System.out.println("Hello World!");
  }
}
```

Listing 2. Feature vector for file HelloWorldApp.java

```
args: 1, class: 2, Hello: 2, HelloWorldApp: 2,
main: 1, out: 1, println: 1, prints: 1,
public: 1, static: 1, String: 1, System: 1,
The: 1, void: 1, World: 2
```

this paper are Naive Bayes and Random Forest. In order to facilitate the comparison of our results with the results of other studies, we also report the fall-out (O), which represents the probability that a false positive is generated by the model. Our selection criteria for applications included the programming language, application size, and the number of versions released. To identify vulnerable files, we used HP Fortify SCA, an automated static code analysis tool that has specific support for Android applications written in Java. Independent variables were generated through tokenization.

4 RESEARCH METHODOLOGY

4.1 Cross-Validation

If we consider the models that achieved at least 75 percent of recall, we observe that Naive Bayes reached that goal in 15 applications and Random forest in 14. In summary, concerning RQ1, this exploratory experiment shows that the presented prediction technique can be used to build high quality prediction models for Android applications.

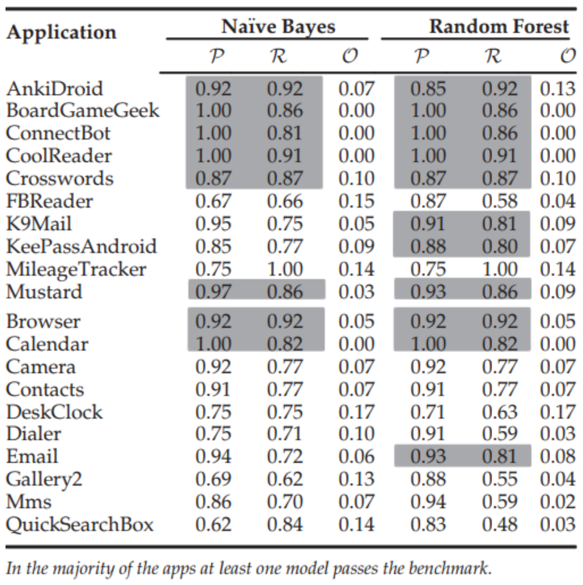
4.2 Prediction in Future Releases

In summary, concerning RQ2, this exploratory experiment shows that the presented prediction technique can forecast with excellent performance the vulnerable files of the future versions of an Android application. Furthermore, such a model is stable for at least 13 months on average.

4.3 Cross-Project

Concerning RQ3, this exploratory experiment illustrates that some models built on a single application can predict which software components are vulnerable in other applications. However, we do not have a technique to identify which applications have data that

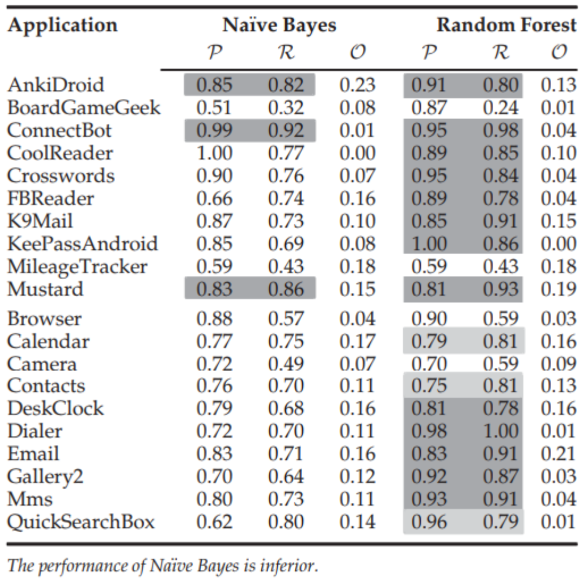
Experiment 1: Average Performance across 10 Folds



Application	Naïve Bayes			Random Forest		
	$\mathcal{P}$	$\mathcal{R}$	$\mathcal{O}$	$\mathcal{P}$	$\mathcal{R}$	$\mathcal{O}$
AnkiDroid	0.92	0.92	0.07	0.85	0.92	0.13
BoardGameGeek	1.00	0.86	0.00	1.00	0.86	0.00
ConnectBot	1.00	0.81	0.00	1.00	0.86	0.00
CoolReader	1.00	0.91	0.00	1.00	0.91	0.00
Crosswords	0.87	0.87	0.10	0.87	0.87	0.10
FBReader	0.67	0.66	0.15	0.87	0.58	0.04
K9Mail	0.95	0.75	0.05	0.91	0.81	0.09
KeePassAndroid	0.85	0.77	0.09	0.88	0.80	0.07
MileageTracker	0.75	1.00	0.14	0.75	1.00	0.14
Mustard	0.97	0.86	0.03	0.93	0.86	0.09
Browser	0.92	0.92	0.05	0.92	0.92	0.05
Calendar	1.00	0.82	0.00	1.00	0.82	0.00
Camera	0.92	0.77	0.07	0.92	0.77	0.07
Contacts	0.91	0.77	0.07	0.91	0.77	0.07
DeskClock	0.75	0.75	0.17	0.71	0.63	0.17
Dialer	0.75	0.71	0.10	0.91	0.59	0.03
Email	0.94	0.72	0.06	0.93	0.81	0.08
Gallery2	0.69	0.62	0.13	0.88	0.55	0.04
Mms	0.86	0.70	0.07	0.94	0.59	0.02
QuickSearchBox	0.62	0.84	0.14	0.83	0.48	0.03

In the majority of the apps at least one model passes the benchmark.

Experiment 2: In the Majority of the Cases, the Average Performance in Future Prediction Is Above the Benchmark with Random Forest

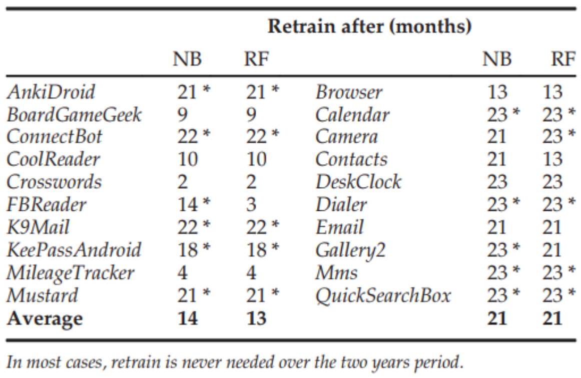


Application	Naïve Bayes			Random Forest		
	$\mathcal{P}$	$\mathcal{R}$	$\mathcal{O}$	$\mathcal{P}$	$\mathcal{R}$	$\mathcal{O}$
AnkiDroid	0.85	0.82	0.23	0.91	0.80	0.13
BoardGameGeek	0.51	0.32	0.08	0.87	0.24	0.01
ConnectBot	0.99	0.92	0.01	0.95	0.98	0.04
CoolReader	1.00	0.77	0.00	0.89	0.85	0.10
Crosswords	0.90	0.76	0.07	0.95	0.84	0.04
FBReader	0.66	0.74	0.16	0.89	0.78	0.04
K9Mail	0.87	0.73	0.10	0.85	0.91	0.15
KeePassAndroid	0.85	0.69	0.08	1.00	0.86	0.00
MileageTracker	0.59	0.43	0.18	0.59	0.43	0.18
Mustard	0.83	0.86	0.15	0.81	0.93	0.19
Browser	0.88	0.57	0.04	0.90	0.59	0.03
Calendar	0.77	0.75	0.17	0.79	0.81	0.16
Camera	0.72	0.49	0.07	0.70	0.59	0.09
Contacts	0.76	0.70	0.11	0.75	0.81	0.13
DeskClock	0.79	0.68	0.16	0.81	0.78	0.16
Dialer	0.72	0.70	0.11	0.98	1.00	0.01
Email	0.83	0.71	0.16	0.83	0.91	0.21
Gallery2	0.70	0.64	0.12	0.92	0.87	0.03
Mms	0.80	0.73	0.11	0.93	0.91	0.04
QuickSearchBox	0.62	0.80	0.14	0.96	0.79	0.01

The performance of Naïve Bayes is inferior.

can be used to produce vulnerability prediction models with the above-mentioned property.

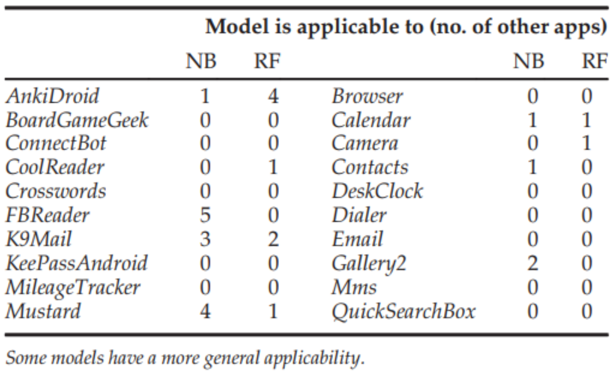
Experiment 2: Need for Re-Training



	Retrain after (months)				
	NB	RF		NB	RF
AnkiDroid	21 *	21 *	*Browser*	13	13
BoardGameGeek	9	9	*Calendar*	23 *	23 *
ConnectBot	22 *	22 *	*Camera*	21	23 *
CoolReader	10	10	*Contacts*	21	13
Crosswords	2	2	*DeskClock*	23	23
FBReader	14 *	3	*Dialer*	23 *	23 *
K9Mail	22 *	22 *	*Email*	21	21
KeePassAndroid	18 *	18 *	*Gallery2*	23 *	21
MileageTracker	4	4	*Mms*	23 *	23 *
Mustard	21 *	21 *	*QuickSearchBox*	23 *	23 *
Average	**14**	**13**		**21**	**21**

In most cases, retrain is never needed over the two years period.

Experiment 3: Applicability of Each Model across Projects



	Model is applicable to (no. of other apps)				
	NB	RF		NB	RF
AnkiDroid	1	4	*Browser*	0	0
BoardGameGeek	0	0	*Calendar*	1	1
ConnectBot	0	0	*Camera*	0	1
CoolReader	0	1	*Contacts*	1	0
Crosswords	0	0	*DeskClock*	0	0
FBReader	5	0	*Dialer*	0	0
K9Mail	3	2	*Email*	0	0
KeePassAndroid	0	0	*Gallery2*	2	0
MileageTracker	0	0	*Mms*	0	0
Mustard	4	1	*QuickSearchBox*	0	0

Some models have a more general applicability.

5 TREATS TO VALIDITY

Construct Validity. We used the Fortify SCA tool to identify vulnerabilities via static source code analysis rather than using the vulnerabilities reported in a database such as the NVD. This choice was obligatory, as there are no public databases with sufficient numbers of vulnerabilities to analyze for Android applications. *Internal Validity.* In this study, we considered only the Java source files of each Android application. We excluded the XML manifest file that is packaged with each Android application. The manifest contains important information, such as the permissions that need to be granted to the application. *External Validity.* Our study is exploratory and a larger scale validation is necessary. Our results might be specific to the 20 applications we have selected. In an attempt to counter this threat, we evaluated our approach using an range of applications that vary in terms of size, revision history, and popularity. However, further studies using a different set of experimental materials are necessary in order to generalize the results to the entire class of Android applications.

6 CONCLUSIONS AND FUTURE WORK

n this paper, we presented an approach to predict whether a software component is vulnerable based on text mining of its source code. The approach has never been used before in the context of vulnerability prediction. The approach presented here analyzes the source code directly instead of indirectly via software or developer metrics. The results of our exploratory study demonstrate that the approach has good performance for both precision and recall when it is used for within-project prediction. In general, we obtained a prediction power that is equal or even superior to what is achieved by state of the art vulnerability prediction models.

7 REFERENCES

[1] Riccardo Scandariato James Walden Aram Hovsepyan Wouter Joosen Predicting Vulnerable Software Components via Text Mining