Vulnerability Prediction Models: A case study on the Linux Kernel

Matthieu Jimenez    Mike Papadakis    Yves Le Traon

Jimenez et al. “Vulnerability Prediction Models: A Case Study on the Linux Kernel” SCAM’16
Vulnerabilities ?
A vulnerability

“An information security ‘vulnerability’ is a mistake in a software that can be directly used by a hacker to gain access to a system or network.”

~ CVE - website ~
Vulnerabilities are special

More **Important** - Critical

There are **more bugs** than vulnerabilities

Uncovered differently - defects can be easily noticed, while vulnerabilities not.
Vulnerabilities are Web server used to remotely control the glassware-cleaning machine

CVE for that...
Prediction Model
Prediction Models

Models analysing current and historical events to make prediction about the future and/or unknown events!
Vulnerability Prediction Model ?
Vulnerability Prediction

Take advantage of the knowledge on some part of a software system and/or previous releases.
Vulnerability Prediction

to automatically classify software entities as vulnerable or not!
Granularity

Possibility to work at:

• *module* level
• *file* level
• *function* level
• …
In this *work*, we stay at the *file* level!

*Morrison* et al. “Challenges with applying vulnerability prediction models,” in *HotSoS’15.*
GOAL
Replicating and comparing the main VPMs approaches on the same software system.
Replication ...
Exact independent replication
Exact replication

procedures of an experiment are followed as closely as possible

e.g. here we replicate using the same machine learning settings
Independent replication

deliberately vary one or more major aspects of the conditions of the experiment
e.g. we use our dataset
Approaches ...
# Include

and $f(n)$ calls
Include & Function calls

Introduced by Neuhaus et al. at CCS’07
Include & Function calls

Introduced by Neuhaus et al. at CCS’07

Intuition: vulnerable files share similar set of imports and function calls
Include & Function calls

Introduced by Neuhaus et al. at CCS’07

Intuition: vulnerable files share similar set of imports and function calls

build a model based on either includes or function calls of a file.
## Overview

<table>
<thead>
<tr>
<th></th>
<th>Preprocessing</th>
<th>Learning</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Include &amp; function calls</strong></td>
<td>Retrieve all include and function calls of a file</td>
<td>SVM with a linear kernel</td>
</tr>
</tbody>
</table>

2 models are built
Software Metrics
Several works on using metrics to predict vulnerabilities, mostly by Shin et al.
Several works on using metrics to predict vulnerabilities, mostly by Shin et al.

Software metrics are used in defect prediction build a model based software metrics (complexity, code churn, …)
# Overview

<table>
<thead>
<tr>
<th>Software Metrics</th>
<th>Preprocessing</th>
<th>Learning</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Compute complexity metrics of each function (keeping sum, avg and max) code churn and the number of authors of every files.</td>
<td>Logistic regression</td>
</tr>
</tbody>
</table>
Text Mining
Text Mining

suggested by Scandariato et al. in 2014.

Abstract—This paper presents an approach based on machine learning to predict which components of a software application contain security vulnerabilities. The approach is based on text mining the source code of the components. Namely, each component is characterized as a set 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.

Index Terms—Vulnerabilities, prediction model, machine learning

1 INTRODUCTION

Verification and validation (V&V) techniques like security testing, code review and formal verification are becoming effective means to reduce the number of post-release vulnerabilities in software products [1]. 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 [2]. However, V&V is not inexpensive. An early estimation assessed that V&V out to be correct [6]. In the above examples, the choice of the features that are used as predictors is determined by the expectations of a knowledgeable individual.

In our work, we investigated a technique that relies less on a particular underlying axiom. Starting from the observation that a programming language is a language after all (like English) and that syntax tokens equate to words, we set out to analyze the source code by means of text mining techniques, which are commonplace in information retrieval. Text mining applied to source code was introduced by Hata et al. [7] for the prediction of software defects and is hence applied to the domain of software vulnerabilities. We use the bag-of-words representation, in which a software component (a Java source file in this paper) is seen as a set of terms with associated frequencies. The terms are the features we use as predictors. Hence, the set of features used for modeling is not fixed or predetermined but rather depends on the vocabulary used by the developers. In this sense, this technique is less constrained or biased by an underlying theory of what is a priori expected to

romium.
Text Mining

suggested by Scandariato et al. in 2014.

Aim: building a model requiring no human intuition for feature selection
Text Mining

suggested by Scandariato et al. in 2014.

Aim: building a model requiring no human intuition for feature selection

build a model based on a bag of word extracted from a file
## Overview

<table>
<thead>
<tr>
<th>Text mining</th>
<th>Preprocessing</th>
<th>Learning</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Creating a bag of word (splitting the code according to the language grammar) for every files</td>
<td>• Discretisation of the features (making them boolean) • Remove of all features considered useless • Random Forest with 100 trees</td>
</tr>
</tbody>
</table>
Dataset
Introducing the dataset

based on commit and not release
Introducing the dataset

• CVE-NVD database as a source of vulnerabilities
• Bugzilla as a source of bugs
Introducing the dataset

• build automatically
• with the latest data available
• on the Linux Kernel
Overall dataset statistics

2006-June 2016

- **1,640 vulnerable files**, accounting for **743 vulnerabilities**
- **4,900 buggy files** related to **3,400 bug reports**
- **more than 50,000 files** in total
Research Questions

• RQ1. Can we **distinguish** between **buggy** and **vulnerable files**?
Research Questions

• **RQ1.** Can we distinguish between buggy and vulnerable files?

• **RQ2.** Can we distinguish between vulnerable and non-vulnerable files?
Research Questions

• RQ1. Can we distinguish between buggy and vulnerable files?

• RQ2. Can we distinguish between vulnerable and non vulnerable files?

• RQ3. Can we predict future vulnerable when using past data?
Research Questions

• RQ1. Can we distinguish between buggy and vulnerable files?

• RQ2. Can we distinguish between vulnerable and non vulnerable files?

• RQ3. Can we predict future vulnerable when using past data?
  ✦ Distinguish between buggy and vulnerable files
  ✦ Distinguish between vulnerable and non vulnerable files?
Experimental Dataset

*Buggy vs Vulnerable files*
Can we distinguish between buggy and vulnerable files?

- files related to **bug report patches** vs files from **vulnerability patches**

- **ratio** 3.3 : 1
Realistic Dataset

*Vulnerable vs Non-Vulnerable files*
Realistic dataset

• Can we distinguish between **Vulnerable** and **Non-Vulnerable** files?
  • Reproduce observed **ratio** between different categories of files
  • 3% of (likely) vulnerable files
  • 47% of (likely) buggy files
  • 50% of clear files
Evaluation
RQ1 - Bugs vs Vulnerabilities
RQ2 - Vulnerable vs Non-
RQ3 Time - **Bugs vs**

**Precision** vs **Recall**

- **Precision** graphs show a general decrease over time for all categories, with slight variations.
- **Recall** graphs show an overall increase over time, with some categories maintaining higher values than others.

Legend:
- Red: Function Calls
- Green: Includes
- Cyan: Software Metrics
- Purple: Text Mining

Release: 5, 10, 15, 20
RQ3 Time - Bugs vs

- Function Calls
- Includes
- Software Metrics
- Text Mining

mcc
0.00
0.25
0.50
0.75
1.00
release
5
10
15
20
RQ3 Time - Vulnerable vs Non-

![Graph showing precision and recall over releases for different categories: Function Calls, Includes, Software Metrics, and Text Mining.](image)
RQ3 Time - Vulnerable vs

![Graph showing time versus vulnerable vs with markers for Function Calls, Includes, Software Metrics, and Text Mining](image-url)
Discussion - Findings
VPM’s are working well with historical data
Good precision observed even with unbalanced data
In the **practical case**, the **best trade off** is in favour of including and function calls.
In the **general case**, or favouring **precision** the best one is **text mining**.
Previous studies

Include and Function calls

There is no comparison with Metrics or Text Mining

There are no results related to time

In the context of Linux we have similar results...

Reported

Precision 70%
Recall 45%

We found

Precision 70%
Recall 64%

Neuhaus et al. “Predicting vulnerable software components” CCS’07.
Previous studies

Software Metrics

Reported 10 fold cross validation
Precision 3-5, 97-92%
Recall 87-90, 91, 66-79%

We found
Precision 65%
Recall 22%

In the context of Linux there are significant differences...

Reported results based on time
Precision 3%
Recall 79-85%

We found
Precision 42 : 39%
Recall 16 : 24%

Shin et al. “Evaluating Complexity, Code Churn, and Developer Activity Metrics as Indicators of Software Vulnerabilities” TSE’11.
Shin and et al. “Can traditional fault prediction models be used for vulnerability prediction?” ESE’13.
Previous studies

Text Mining

We found Precision 76%
Recall 58%

In the context of Linux, there are again significant differences.

We found Precision 74 : 93%
Recall 37 : 27%

Reported
Precision 90, 2-57% 10 fold cross validation
Recall 77, 74-81%

Reported
Precision 86%
Recall 77%

Reported results based on time

Scandariato et al. “Predicting Vulnerable Software Components via Text Mining” TSE’14.
DataSet and Replication package and additional results will be available soon...

Please contact Matthieu Jimenez (Matthieu.Jimenez@uni.lu)
Thank you for your attention!