Challenges with Applying Vulnerability Prediction Models

Patrick Morrison[1] Kim Herzig[2] Brendan Murphy[2] Laurie Williams[1]
[1]Department of Computer Science, North Carolina State University, Raleigh, NC, USA
[2]Microsoft Research, Cambridge, UK
pjmorris@ncsu.edu kimh@microsoft.com bmurphy@microsoft.com lawilli3@ncsu.edu

Abstract

While Microsoft product teams have adopted defect prediction models, they have not adopted vulnerability prediction models (VPMs). Seeking to understand this discrepancy, we replicated a VPM for two releases of the Windows Operating System, varying model granularity and statistical learners. We reproduced binary-level prediction precision (~0.75) and recall (~0.2). However, binaries often exceed 1 million lines of code, too large to practically inspect, and engineers expressed preference for source file level predictions. Our source file level models yield precision below 0.5 and recall below 0.2. We suggest that VPMs must be refined to achieve actionable performance, possibly through security-specific metrics.

Keywords: Vulnerabilities, Prediction, Metrics, Complexity, Churn, Coverage, Dependencies.

1. Introduction

Software security vulnerabilities are a constant threat to software companies and their customers. In recent years, the list of severe security vulnerabilities and their wide impact on large numbers of customers brought software security more and more into the public and media focus. Security holes such as the Secure Sockets Library (SSL)[1] issues affected thousands of businesses and end customers and led to severe damages.

During the development of a software program, any code change can potentially inject security vulnerabilities or alter the attack surface to expose legacy code that contain security vulnerabilities. Development teams use a number of different techniques to verify if code contains security vulnerabilities, such as fuzzing, static code verifiers, and code reviews. The Microsoft security team has developed the Security Development Lifecycle (SDL) [2] to help its developers build more secure software through the systematic use of these techniques. But for large software products, indiscriminately using reviews to catch security defects can be impractical. For the 70+ million lines of code in the Windows code base, a compete security review could take between 35 and 350 person-years[2]. The application of vulnerability detection and removal techniques must be prioritized to the most suspicious areas of the product.

The Microsoft Windows development teams follow the SDL process and apply a variety of techniques to improve the security of the product. Within the Windows development team, a group of security experts select code areas for complicated and deep-reaching security reviews. To support the selection process, researchers proposed defect prediction models (DPMs) to reduce the search space by exploiting properties of code and processes that are predictive of defects in general, including security vulnerabilities. Basili et al. [3] suggest that prediction models support planning, scheduling and decision-making during software development by enabling allocation of resources to modules more likely to be defect-prone. Following long-standing work in defect prediction model (DPM)-building for software (e.g. [3,4,5]) a number of vulnerability prediction models (VPMs) have been built and tested (e.g. [1,6,7,8,9]). In general, vulnerabilities are a subset of defects, though they occur (or are discovered) much less frequently than general faults. For example, Shin and Williams [7] reports that 21% of files in Mozilla Firefox have defects, while 3% of files have vulnerabilities. Consequently, VPMs have to deal with highly unbalanced datasets. Although discovered security vulnerabilities are rare, they can cause significant problems for software users and providers, and so they call for priority attention by software development teams.

While DPMs have been integrated into development team workflows within Microsoft, VPMs have not. Discussions with the Windows security development team raised concerns regarding the usefulness of proposed VPMs specifically:

- VPMs built for coarse levels of granularity (code binaries) may be precise, but predicting entities containing hundreds of source files simply confirms prior knowledge without identifying code areas suitable for intense code inspection.
- In general, low fine granular models (source files) could be actionable, but give predictions of low accuracy.

With this study, we explore these two issues in the construction of VPMs. *The goal of this research is to investigate whether vulnerability prediction models are accurate and actionable enough to provide helpful recommendations when allocating engineering resources*. We measure how *'actionable'* a VPM is in terms of *recall* (true prediction rate)*, precision* (positive prediction rate), and the inspection effort required to perform security reviews on code areas suggested by the VPM.

Prior research and feedback from the Windows security development team suggests a number of open issues in the construction of effective VPMs:

- *Choice of granularity* – Model granularity, the selection of a unit to collect data on and make predictions for, forces a tradeoff between classification performance and the actionability of the prediction results. Representative DPM and VPM entity granularities include binary [1], source file [7], class [10], and function/method [11]. Binary-level predictions tend to add little new knowledge, as developers are often aware of which binaries are prone to security defects. Due to the size of the binaries, they may contain hundreds of source files, no specific action can be taken based on this information. File-level prediction model operates at the level of typical development tools, a step closer to the source code. Line- and even instruction-level granularity would be desirable;

[1]http://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2014-0092

[2] *Dowd et al.* [37] *suggest that good security reviewers can cover between 100 and 1000 lines per hour.*

method/function-level is the highest granularity of which we are aware [11].

- *Statistical learner choice* – Menzies et al. [12] suggest "the choice of learning method is far more important than which set of the available data is used for learning." We build VPMs using a range of statistical learners, and compare performance results.
- *Classification performance* – While each team will have its own goals for performance, Shin et al. [7] and others [8,13] have suggested that precision and recall values of 0.7 are reasonable for prediction models. Reported performance for individual DPMs ranges both above and below this threshold. Existing VPMs have sometimes shown lower performance than DPMs.

Instead of building a new VPM based on new metric collections, we replicate an existing VPM [1], confirm previously reported results, but further investigate how actionable its proposed recommendations are. Our investigations are guided by the following research questions:

RQ1 Can we replicate VPMs proposed by Zimmermann et al. [1] achieving comparable prediction accuracy on binary level for two newer version of Windows?

RQ2 How does granularity affect classification performance?

RQ3 How does the choice of statistical learner affect classification performance?

RQ4 Are VPMs predicting vulnerable Windows binaries actionable with respect to security inspection effort?

To answer these research questions, we collected code measurements and pre- and post-release security defect counts for Windows 7 and Windows 8 on two levels of granularity: file-level and binary-level. We then used these metric collections to train and evaluate VPM classification models classifying code entities that contained at least one post-release vulnerability.

Our contributions include:

- A comparison between the replicated and the original classification models trained and evaluated on previous releases of Windows.
- A comparison of two different granularities on classification performance.
- A comparison of classification performance for a variety of typical statistical learning methods.
- An assessment of inspection effort for VPMs predicting vulnerable code binaries instead of source code files.

This paper is organized as follows: Section 2 describes related work, Section 3 describes the methodology used to build the VPMs. In Section 4 we report experimental results before discussing these results in more detail in Section 5. We close with a discussion about limitations and threats to validity (Section 6), and final conclusions and future work (Section 7).

2. Related Work

The design and implementation of VPMs has been rooted in the broader category of DPMs.

2.1 Defect Prediction Models (DPM)

DPMs are models or recommendation models predicting the existence or likelihood of code defects in code entities. The number of related work on DPMs is large. For the sake of brevity, we only refer to the most relevant related studies in this section. The first studies on predicting defects using code metrics emerged in the 1990s. In 1996, Basili et al. [3] applied object-oriented metrics to predict fault-proneness in eight systems, finding statistically significant relationships for some metrics tested. Nagappan et al. [5] found improved defect density prediction performance by normalizing change metrics to file size. Zimmerman et al. [9] used code dependencies to predict the presence of defects. Hall et al. [14] performed a systematic literature review on defect prediction in software engineering, finding that model-building methodology had an impact on prediction performance, and advising on good practice for reporting prediction projects and their results. Other studies used change-related metrics [15], developer related metrics [16], organizational metrics [17], process metrics [18], or change dependency metrics [9,19,20] to build defect prediction models, on various software systems and levels of granularity.

2.2 Vulnerability Prediction Models (VPM)

Zimmermann et al. [1] report that 'Vulnerabilities are not as simple to predict as defects' based upon the analysis of two VPMs for Windows. However, Shin and Williams report "the fault prediction model and the vulnerability prediction model provided similar prediction performance" [7] for a VPM that performs comparably with a DPM when trained and evaluated on Mozilla's Firefox. Their results suggest that VPMs can be approximated by DPMs. In general, VPMs follows lessons learned from defect prediction modeling, with several adaptations made to account for the relative rarity of vulnerabilities compared to defects.

Similar to DPMs, the range and variety of proposed VPMs over time is wide. Neuhaus [8] applies historical data on imports and functions used by vulnerability-prone components. Hovsepyan et al. [21] applied text analysis achieving high precision and recall values. Other studies use code complexity, code churn, and other static alerts to predict attack-prone or vulnerable components [6,22,23]. Smith and Williams [22], evaluated the use of database access references ('SQL hotspots') as a vulnerability predictor, finding a positive correlation between database access points in source code and both vulnerabilities and code churn. Doyle and Walden [23] studied the evolution of vulnerability density over time in PHP applications, observing a trend of decreasing vulnerability density over time.

In this paper, we replicate the study of Zimmermann et al. [1] who studied typical metrics for complexity, churn, code coverage, organization as well as dependency metrics and evaluated their correlations with post-release vulnerabilities in Windows Vista binaries. They found that each set of metrics had strengths and weaknesses in terms of recall and precision performance for vulnerability prediction, but performance improvement was needed for binary-level prediction to be practical. Our results will confirm their findings for newer releases of Windows. However, we will also investigate how models using the same metrics but on finer levels of granularity (source file level) perform.

The models described to this point have focused on either the binary/executable file level, or on the source file level. Giger et al. [24] built VPMs at method/function-level granularity, the highest granularity prediction model reported in the literature at present.

2.3 The Impact of Machine Learners

Menzies et al. [25] highlight that how a model is built is as important, or perhaps more important, than the specific metrics used. Early work in DPMs for software engineering often used either linear regression or logistic regression to model defect or vulnerability proneness. In recent years, the software engineering community has begun applying machine learning techniques that take advantage of structure in the data beyond the linear

combinations modeled by regression. These modeling techniques, among many others, include Decision Trees and Naïve Bayes [13], Support Vector Machines [1], clustering algorithms and Random Forests [26]. Comparing different machine learners on the same dataset can yield significant different results. Weyuker et al. [27] compared the effectiveness of several modelling methods for fault prediction and showed that different modelling methods can lead to different prediction accuracies, depending on the dataset. As discussed above, vulnerabilities are rare and thus, VPMs have to deal with highly imbalanced datasets. To investigate the impact of different modelling methods on Windows VPMs, we apply several of these modelling techniques in our experiments to assess whether algorithm choice makes a difference in terms of classification performance. We also include decision tree models, which are believed to work well on unbalanced training sets.

3. Reseearch Methodology

In principal, we replicated earlier VPMs for Windows proposed by Zimmermann at al. [1]. While the original study targeted Windows Vista, the experiments described in this paper were conducted on datasets collected for Window 7 and Windows 8, products of significant size. Choosing the same product as the original study, although different releases, enable a comparison with the original study, giving insight in to how vulnerability prediction metrics in a codebase change over time.

Both Windows releases contain thousands of binaries, hundreds of thousands of source files, and well over 70 million lines of code. Windows 7 development began June 1, 2006, and the 'Release to Manufacturing' (RTM) build was produced on July 14, 2009. Windows 8 development began June 14, 2009, and the RTM build was produced on July 25, 2012.

To capture the code metrics and pre- and post- vulnerabilities this research study relied on the CODEMINE process [28]. Microsoft developed CODEMINE to allow the company to monitor the development attributes of its products both during development and following product release. The CODEMINE process provides a central repository of development and vulnerability metrics which were used within this research study.

3.1 Code Metrics

VPMs presented in this study are based on 29 metrics broadly classified into 5 categories:

- *Churn metrics* [5]. Six metrics to verify the theory that change is more likely to introduce error than its absence. Churn measures are relative to a time period; the period for all presented calculations is between the start and RTM date of the project.
- *Complexity metrics* [3]. One metric to verify the theory that more complicated code is more likely to exhibit errors.
- *Dependency metrics* [9] Seven metrics to verify the theory that the degree to which a piece of code is depended upon, or depends upon other code, influences its impact on software vulnerabilities.
- *Legacy metrics.* Eight metrics to characterize a metric of particular interest to Microsoft. The importance of security in the development of software at Microsoft began receiving increased attention after the Bill Gates' 2002 Trustworthy Computing Memo [29], with significant investments made in security training, tools, and process [2]. Code written after these processes were put in place has had a higher, more process-driven, level of attention to security applied in its design, construction and testing. These metrics verify the theory that code written before the security reset may be more likely to contain vulnerabilities.
- *Size metrics.* Seven metrics to verify the theory that larger source files are more difficult to mentally manage, and, therefore, are more prone to defects and vulnerabilities.
- *Pre-Release vulnerabilities*. For VPMs predicting post-release vulnerabilities, we used pre-release vulnerabilities to model usual suspects. More details on collecting pre-release vulnerabilities is given in Section 3.2.

Most of the discussed measures have been used for predicting defects in prior research both within and outside Microsoft (e.g. [1,6,9,22]). Table 3, in the appendix, identifies all metric used in the study and provides a description of the metric. Where noted, average, maximum and total values were taken for several of the metrics. Depending on the metric, data was available at either the source file level or at the function level. In cases where function level data was present, amounts were aggregated up to the file level via averages, totals and maximums. Binary-level data was obtained by aggregating source-file level data up to the binary in which each source file is used. This study uses additional metrics that were not available at the time of the original study by Zimmermann at al. [1]. The table identifies which metrics are common between the two studies and which are unique to this study.

All size, churn, complexity, and dependency metrics were measured as of each releases' RTM date.

3.2 Pre- and Post-Release Vulnerabilities

As dependent variables, we used the number of pre- and post-release security vulnerabilities detected and fixed within the corresponding source files and code binaries respectively. A post-release vulnerability is a security issue detected and corrected after releasing the corresponding software product to the public. Pre-release vulnerabilities are issues that are identified and fixed during software development. Pre-release vulnerabilities of product version N may also be post-release vulnerabilities for product version N-1. We credit pre-release security changes to security practices outlined in the Security Development Lifecycle (SDL) [2] as applied during Windows development. Post-release security changes can be considered as 'escapes' from the SDL. Escapes may be worthy of investigation for SDL application in future releases.

To identify post-release vulnerability fixes, we counted the number of code changes applied in Windows service pack branches marked as security fix. These branches serve as sink of defect fixes that will eventually be shipped to customers as part of a service pack or hot-fix. No feature development is permitted on these branches. Pre-release vulnerabilities were identified by bug reports marked as security vulnerabilities which resulted in changed source files and binaries. We were confident in the accuracy of this security characterization as all bug reports that were labeled as security defects have been triaged by the security team.

In our data set, vulnerable source files represent approximately 0.003% of all source files.

3.3 Prediction Models

For both levels of granularity, binaries and source files, we build classification models that separate code entities that had at least one vulnerability from code entities that had no vulnerabilities. To train individual VPMs, we used the metric data described in Section 10 as independent variables and the number of pre- or post-release vulnerabilities as dependent variables. For

classification models predicting post-release vulnerabilities, we use the number of pre-release vulnerabilities as additional independent variable. For each Windows release and level of granularity, we split the overall data collection into two subsets. One subset containing 2/3 of the data points is used for training, the other for testing purposes. To split the data, we used stratified sampling—the ratio of code entities associated with vulnerabilities from the original dataset is preserved for both subsets. We repeatedly sampled the original dataset 100 times (100-cross-fold-validation). In total, we generated 800 independent training and testing sets: two Windows releases, two levels of granularity, and 100-cross folds each (similar to [1,9,26]).

We conducted the experiments using the R statistical software [30] (version 3.10). Instead of using the original feature vectors provided by the raw metric values, we applied R's prcomp [31] procedure to our data to produce principal components. Principal Component Analysis (PCA) [32] reduces redundancy in our matrix of metrics and observations by maximizing the variance of linearly independent variables. Deciding how many of these variables to use in model building typically takes one of two forms; either a limit on the number of terms in the model is set, or some total amount of variance to be accounted for by the model is set. We selected principal components that accounted for 95% of variance.

In pursuit of high prediction performance, we used Max Kuhn's R package caret [33] to build VPMs based on the components selected by PCA and on a set of series of machine learning techniques [34,35]:

- *Logistic Regression (LR)* - Generalized linear model using a logistic function.
- *Naïve Bayes (NB)* - Applying Bayes' theorem, this is a simple probabilistic classifier assuming strong independence of the independent variables.
- *Recursive Partitioning (RP)* - A variant of decision trees, this model can be represented as a binomial tree and it is often used for classification tasks.
- *Support Vector Machine (SVM)* - This model classifies data by determining a separator that distinguishes the predicted classes with the largest margin. We used the radial kernel for our experiments.
- *Tree Bagging* (TB) - Another variant of decision trees, this model uses bootstrapping to stabilize the decision trees.
- *Random forest* (RF) - A variant of decision trees, this model can be represented as a binomial tree and popularly used for classification tasks.

Each model is optimized by the caret package [33] optimizing various tune parameters (please see caret manual for more details). "The performance of held-out samples is calculated and the mean and standard deviations is summarized for each combination. The parameter combination with the optimal resampling statistic is chosen as the final model and the entire training set is used to fit a final model" [33]. The level of performed optimization can be set using the *tuneLength* parameter, which is set to five for all experiments in this paper.

3.4 Inspection Effort

We define 'inspection effort' to be the number of person-hours required to perform security review on the positives (correct or incorrect) identified by the VPM. Following previous practice, e.g. [36,28], we assume that inspection effort is proportional to code size, that all vulnerabilities correctly or incorrectly identified by the VPM must be security reviewed. Dowd [37] suggests that one hour of security review can cover between 100 and 1000 lines. Summing the lines of code present in the positives predicted, and dividing the sum by the inspection rate yields the inspection effort in person-hours for a set of VPM predictions. For constant recall and precision values, lower effort values are more actionable. To illustrate the difficulties with present VPMs, we assume an average inspection rate of 500 lines per person-hour.

4. Results

In this section, we present results collected during our investigations.

We report on a series of experiments in prediction for Windows post-release vulnerabilities. Table 1 reports mean recall and precision for the 100-fold validations run for each combination of release (Windows 7, Windows 8), granularity (binary, file), and Model (LR, NB, RF, RP, SVM, TB). The highest precision and highest recall for each release and granularity are shown in bold.

To align our replicated models with original proposed model by Zimmermann et al. [1] and other related work, we compare our models precision and recall values to closely related work. Table 2 summarizes results for the VPMs built in this paper, together with results for VPMs built in referenced works [1,6,7,8,21,23,38]. The list of referenced VPMs in the table is based on reporting in the referenced paper of the table data on granularity, size and recall and precision performance. The 'Source' column indicates the source of the VPM. 'Granularity' indicates the unit used to train and predict against. 'N' indicates the number of entities used in the experiments. '% Vulnerable' reports the percentage of vulnerable files in the data used to train the VPMs. 'Recall' and 'Precision' report performance ranges for each measure.

Table 1: Vulnerability Prediction Model Performance

	Windows 7		**Windows 8**	
	Precision	**Recall**	**Precision**	**Recall**
		Binary level		
LR	0.5	0.12	0.32	0.09
NB	0.3	**0.42**	0.11	**0.4**
RF	**0.76**	0.27	**0.69**	0.07
RP	0.51	0.22	0.23	0.07
SVM	0.51	0.13	0.64	0.04
TB	0.69	0.13	0.45	0.1
		File level		
LR	0.01	0	0	0
NB	0.07	**0.14**	**0.01**	**0.01**
RF	**0.47**	0.02	0	0
RP	0.21	0.04	0	0
SVM	0.38	0.02	0	0
TB	0.36	0.03	0	0

Table 2: Vulnerability Prediction Model Comparison

Source	Granularity	N	% Vulnerable	Recall	Precision
current paper	binary	1000's	9.5	0.04-0.42	0.11-0.76
Zimmermann et al. [5]	binary	1000's	"very low"	0.20-0.40	0.40-0.67
current paper	source file	100000's	0.33	0.00-0.14	0.00-0.47
Shin [7]	source file	11051	3.0	0.52-1.00	0.21-0.90
Chowdhury et al. [26]	source file	11139	7.0	0.29-0.74	Not reported
Shin [7]	source file	11051	3.0	0.52-1.00	0.21-0.90
Hovsepyan [18]	source file	2888	Not reported	0.88	0.85
Smith and Williams [21]	soure file	2213	26.0	0.32	0.43
Gegick [6]	component	25	0.5	Not reported	Not reported
Neuhaus et al. [9]	component	10452	4.05	0.35-0.55	0.55-0.80

5. Discussion

In this section, we consider how our results address the research questions, and what the results imply for future work on VPMs.

5.1 RQ1 Can We Replicate VPMs Proposed for Windows with Comparable Accuracy?

Table 2 holds the answer for this first research question. Comparing our replicated models trained and evaluated on Windows 7 and Windows 8 with the original study [1] conducted on Windows Vista, we can compare the first two lines in Table 2. Comparing the best models (highest values) for both studies, we see that our replicated models reported slightly better recall and precision values than original models: recall increased from 0.40 to 0.42 while precision increased from 0.67 to 0.76. Zimmermann et al. [1] used logistic regression as statistical learner. Comparing the original recall and precision values to our equivalent LR learner, we see from Table 1 that for both Windows releases the reported recall—0.12 for Windows 7 and 0.09 for Windows 8—is lower than in the original study (0.20-0.40). For precision, we see that LR in this study reported similar values--0.50 for Windows 7 and 0.32 for Windows 8—than the original model (0.40-0.67). Overall, results reported for Windows 7 and Windows 8 were comparable with the results reported by Zimmerman et al. [1]—considering usual fluctuations and differences between product releases.

Compared to other studies (see Table 2), recall values for Windows 7 and Windows 8 appear to be below average, especially recall values. We suspect that this is due to the fact that only very few binaries and source files were reported vulnerable. We discuss this issue in more detail in the next section.

Menzies et al. [25] observe that even low precision models can be useful, if the value of the item being recalled is high. This situation applies for many VPMs reported in earlier, related studies. However, considering Windows recall values for both levels of granularity (recall between 0.00-0.14 for files), we doubt that moderate precision values will be useful.

5.2 RQ2 How Does Granularity Affect Classification Performance?

Comparing VPMs for binary level (first line in Table 2) to VPMs on source file level (third line in Table 2), we see a significant drop in precision and recall. Comparing the best source file VPM learner (RF) to results of the same VPM learner on binary level, we see for Windows 7 a drop in precision from 0.76 to 0.46 and a drop in recall from 0.27 to 0.02. Even worse for Windows 8 where the highest recall and precision rounds up to 0.01 for Naïve Bayes. Models based on other statistical learners reported recall and precision below 0.01. This trend is true for file level VPMS for both releases. Source file VPMs report significant lower accuracy measures when compared to binary-level VPMs.

Comparing our source file VPMs to other studies (lines four to ten in Table 2) shows that Windows file VPMs report exceptionally low accuracy measures. Comparing the number of source files and the relative number of vulnerable source files for Windows and binary level and related source file studies, we see a significant difference. Please note that binaries can contain hundreds of source files and thus accumulate for more vulnerabilities, when compared to source files only. Compared to binary level, the number of entities (column 'N' in Table 2) is two orders of magnitude higher while the relative number of vulnerable entities drops by one order or magnitude. Compared to other study subjects, Windows contains one magnitude more source files and also one magnitudes fewer vulnerabilities.

We conclude that for Windows and our statistical learners, file-level granularity decreases recall and precision performance compared with binary-level granularity, therefore we can replicate prior studies. We also conclude that Windows source file VPMs have to deal with a relative number of vulnerable source files that is a magnitude lower than reported in related studies.

5.3 RQ3 Does The Choice of Statistical Learner Affect Classification Performance?

We built classification models using six separate statistical learning methods. The main rational behind using multiple models was to check for inconsistencies and performance differences rather than trying to build the best prediction model. By construction, some statistical learners, such as decision trees, are known to be a more adequate technique than decision trees with imbalanced data [39].

Recall and precision performance results for different learners and different levels of granularity are listed in Table 1. For both levels of granularity, precision and recall values reported by different statistical learners differ significantly. As expected, decision trees (RF) report best precision values for both releases and levels of granularity, except Windows 8 file level. Similar, Naïve Bayes (NB) report best recall measurements. Compared to the baseline performances of logistic regression (LR) reported by Zimmermann et al. [1], Random Forests (RF) and Recursive Partitioning (RP) and Support Vector Machines (SVM) report better precision values for binary level. Naïve Bayes (NB) models reported better recall values.

Taking recall and precision together (for non-zero data points), there is a 3:1 performance difference when comparing the best performing learner against the worst performing learner for a given release and granularity. Model selection should be approached with caution. Several alternatives, notably Naïve

Bayes (NB) and Random Forests (RF), should be tried, and the choice should likely be revisited over time.

We conclude that statistical learner choice can alter performance. Naïve Bayes and Random Forests perform best for our dataset which contains relatively few vulnerability-prone files (highly imbalanced dataset).

5.4 RQ4 Are VPMs Predicting Windows Binaries Actionable With Respect To Security Inspection Effort?

From a practitioner's point of view, VPMs must be considered as development tool recommending additional quality assurance efforts for predicted code entities. At Microsoft, security teams have developed the Security Development Lifecycle (SDL) [2] guiding developers to build more secure software. Among others, one key aspect of the SDL is to conduct deep reaching security code reviews to spot security issues or bad coding practices. For Window development teams, code entities predicted to contain vulnerable source code will be promoted for additional code reviews. The effort to conduct these reviews is proportional to the size of the predicted code entity. VPMs predicting binaries, which may contain hundreds of source files, would trigger code review of significant more effort than source file VPMs would do.

To validate the feasibility of code reviews triggered by binary VPMs, we follow the approach by Dowd [37] who suggests that one hour of security review can cover between 100 and 1000 lines. Microsoft Windows 8.1 is made up of over 6000 binaries with the average binaries made up of hundreds of files and millions lines of code. If we assume that is it is possible to perform a security review on 500 lines per hour (average number reported by Dowd [37]), it will take 100's of work days (8 hour working day) to review a single predicted binary. Even if we assume security reviews of 10k lines of code per hour, it would still take more than a week to review a single binary. Even in an unrealistic scenario of 10k lines of code per hour and a perfect VPM precision of 1.0, using binary VPMs is infeasible—it simply takes too long to review an entire binary. Please keep in mind that binaries change daily and that VPMs usually predict more than one binary to be vulnerable.

Predicting source files instead of binaries, decreases review effort significantly, although the model would predict more files that are potentially vulnerable than binaries. Some Windows code files can still contain thousands of lines of code, but the average file size is in the hundreds of lines. Assuming an average review speed of 500 lines per hour, reviewing a single predicted source files takes about an hour. Even if we assume a lower bound of 100 lines of code per hour as suggested by by Dowd [37], reviewing a single file would take less than a day. Compared to binary prediction a feasible and actionable scenario.

We conclude that VPMs at binary level are not actionable. Reviewing only one predicted binary takes 100's of working days. Reviews at file level can be completed within a day per file.

5.5 Other Observations

Here, we make several observations based on what we have observed over the course of the project.

In reviewing the defect and vulnerability prediction literature, and in discussions with the Windows security team prior to the beginning of the project, several themes about expectations of metric relationships to vulnerability prediction emerged, notably:

- *Complexity*: The more complex a file is, the more likely it is to exhibit vulnerabilities.
- *Churn*: The more a file changes, the more likely it is to exhibit vulnerabilities.
- *Age*: The older the code in a file, the more likely it is to exhibit vulnerabilities.
- *Process*: Code written with attention to security is less likely to exhibit vulnerabilities.

We take each of these notions, and compare them to the empirical evidence we collected.

- Complexity did not appear directly as a factor in our investigations. Correlations found for Windows 7 and Windows 8 for our direct complexity measures were negative, and complexity measures did not dominate the high variance principal components. We find this result surprising, as it seems intuitive that complexity in code is more likely to yield defects and vulnerabilities, and we would expect to see an effect. Among alternative hypotheses are the notions that attackers will target the easiest means of compromising a system, thus favoring simpler (but vulnerable) targets within the code, or that some other attribute of code (e.g. churn, see below) offers an easier target.
- Churn appears as a factor in vulnerability prediction. All but one of the correlations was positive, and several churn measures were selected by PCA for both projects.
- Age appears as a factor in vulnerability prediction. The Legacy measures had positive correlations. Legacy LOC Percent was selected by PCA for Windows 7, but not for Windows 8. One interpretation of this is that enough of the legacy code issues were addressed during Windows 7 that the legacy code was less of a factor for Windows 8.
- Process can be indirectly implicated as a factor. We ran a correlation analysis between vulnerabilities found pre-release in Windows 8 (which we credit to the SDL), and those found post-release. There was a 0.41 correlation, suggesting a relationship between the two pools of vulnerabilities, from which we infer that the SDL is, at least in part, catching the kinds of vulnerabilities that are found post-release.

5.6 Why Did We Fail To Build An Adequate VPM, And What Can We Do About It?

By selecting a wide range of metrics, multiple statistical learners, and by following accepted practice for building prediction models, we intended to obtain VPMs that performed well enough for use by development teams. We did not achieve that goal. While using prediction models to identify potentially vulnerable binaries might have acceptable precision values, reviewing a single entity would require hundreds of days to complete. On the other hand, reviews for source files seem to be feasible, however, source file VPMs are not precise enough to be trusted.

Nevertheless, we believe the results presented in this paper can help scope appropriate next steps.

With less than 1% vulnerable files in over 100,000 files, Windows contributes a unique dataset to the VPM literature. Further investigation of how the number of entities and rarity of vulnerable entities affects performance, seems warranted. Distinct from dataset size, and unsurprisingly, higher granularity reduces prediction performance.

Even though related work reported VPMs on source file level with higher precision and recall values, the number of source files falsely predicted to be vulnerable must be put in perspective. False positive rates around 0.5 would still not be good enough to trigger real action. Assuming each security review triggered by a

false prediction to be a waste of time would introduce considerable cost and delays to the Windows development process.

The variance in statistical learner performance within experimental unit (release, granularity) and between granularities, suggests that learner choice is heavily influenced by characteristics of the data.

6. LIMITATIONS AND THREATS TO VALIDITY

Our study concerned two versions of Windows, and reported results from a previous version of Windows. We cannot say whether these results generalize outside of the Windows codebase, Microsoft development processes, or the application of the SDL. Even within the relatively constrained environment of collecting data on two releases of Windows, many context variables influence our measurements and any conclusions drawn from them. Within the development process, people, process and tools changed over the years during which each release was developed. Between releases, different features were focused on, presumably featuring varying levels of size, complexity and subject matter. Over time, different areas of the code received differing amounts of use by the general population, and differing amounts of attention by attackers. The development team may close attack vectors, or attackers may choose alternative attack vectors.

For a variety of technical reasons, we were unable to access metric data for about 5% of the binaries and their constituent source files, however no files were intentionally excluded from the analysis. While the model-building and predictions were based on post-release vulnerabilities reported within a year after release for Windows 7 and Windows 8, the Windows 7 metric correlation data was collected over a similar timeframe as the Windows Vista data at the time of its reporting, about three years after each releases' RTM date; Windows 8 has approximately one year's worth of Windows 8 vulnerability reporting. All of these factors influence the measurements, correlations and models developed here in ways that we may not have properly controlled for.

7. CONCLUSIONS AND FUTURE WORK

The objective of the work was to understand why Microsoft product groups did not use VPM's as part of their development process, in spite of the fact that they did use DPM's. To validate our approach we replicated a previous study of generating VPM's to identify vulnerabilities in Microsoft code. As part of our study we collected metrics down to the source file level for two releases of Windows, built VPMs for each release, two granularities of prediction unit, and six statistical learners. We trained each VPM on metric data collected from Windows code and process databases, and used it to predict post-release vulnerabilities. The results of our studies were comparable to prior studies in predicting vulnerabilities at the binary level they were inaccurate at the source file level.

The paper explores the practicalities of using the results of the VPM models at the binary level and highlights how impractical the data is for developers to take action. Alternatively the paper shows that predictions at the source level are actionable but the model is inaccurate. This explains why VPM models are not used within Microsoft. The study also confirms prior research that the choice of the statistical leaner does affect classification performance.

Performance enhancements can be made through further experimentation combining of metrics and learning methods, but there does appear to be an upper limit to what is possible via these avenues. We conjecture that security domain knowledge must be added to VPMs before acceptable performance will be achieved.

8. ACKNOWLEDGMENTS

Patrick Morrison was an intern at Microsoft Research, Cambridge, UK, when this work was performed. The Tools for Software Engineers team provided significant support, especially Jacek Czerwonka, Michaela Greiler, and John Smyth. We thank the Realsearch group at North Carolina State University for their valuable input to this paper. Funding for Patrick Morrison for part of the writing of this paper was provided by the National Security Agency.

9. REFERENCES

[1] Zimmermann, T., Nagappan, N., and Williams, L. Searching for a Needle in a Haystack: Predicting Security Vulnerabilities for Windows Vista. In *Software Testing, Verification and Validation (ICST), 2010 Third International Conference on* (2010), 421--428.

[2] Howard, M. and Lipner, S. *The Security Development Lifecycle*. Microsoft Press, 2006.

[3] Basili, V.R., Briand, L.C., and Melo, W.L. A validation of object-oriented design metrics as quality indicators. *Software Engineering, IEEE Transactions on*, 22 (1996), 751--761.

[4] Emam, K.E., Melo, W., and Machado, J.C. The prediction of faulty classes using object-oriented design metrics. *J. Syst. Softw.*, 56 (feb 2001), 63–75.

[5] Nagappan, N. and Ball, T. Use of relative code churn measures to predict system defect density. In *Software Engineering, 2005. ICSE 2005. Proceedings. 27th International Conference on* (2005), 284--292.

[6] Gegick, M., Williams, L., Osborne, J., and Vouk, M. Prioritizing software security fortification throughcode-level metrics. In *Proceedings of the 4th ACM workshop on Quality of protection* (2008), ACM, 31–38.

[7] Shin, Y. and Williams, L. Can traditional fault prediction models be used for vulnerability prediction? *Empirical Software Engineering*, 18 (2013), 25--59.

[8] Neuhaus, S., Zimmermann, T., Holler, C., and Zeller, A. Predicting vulnerable software components. In *Proceedings of the 14th ACM conference on Computer and communications security* (2007), ACM, 529–540.

[9] Zimmermann, T. and Nagappan, N. Predicting defects using network analysis on dependency graphs. In *Proceedings of the 30th international conference on Software engineering* (2008), ACM, 531--540.

[10] Arisholm, E. and Briand, L.C. Predicting Fault-prone Components in a Java Legacy System. In *Proceedings of the 2006 ACM/IEEE International Symposium on Empirical Software Engineering* (2006), ACM, 8–17.

[11] Mende, T. and Koschke, R. Revisiting the Evaluation of Defect Prediction Models. In *Proceedings of the 5th International Conference on Predictor Models in Software Engineering* (2009), ACM, 7:1–7:10.

[12] Menzies, T., Greenwald, J., and Frank, A. Data Mining Static Code Attributes to Learn Defect Predictors. *IEEE*

Transactions on Software Engineering, 33 (2007), 2--13.

[13] D'Ambros, M., Lanza, M., and Robbes, R. Evaluating defect prediction approaches: a benchmark and an extensive comparison. *Empirical Softw. Engg.*, 17 (aug 2012), 531–577.

[14] Hall, T., Beecham, S., Bowes, D., Gray, D., and Counsell, S. A Systematic Literature Review on Fault Prediction Performance in Software Engineering. *Software Engineering, IEEE Transactions on*, 38 (2012), 1276--1304.

[15] Moser, R., Pedrycz, W., and Succi, G. A comparative analysis of the efficiency of change metrics and static code attributes for defect prediction. In *Proceedings of the 30th international conference on Software engineering* (2008), ACM, 181--190.

[16] Pinzger, M., Nagappan, N., and Murphy, B. Can developer-module networks predict failures? In *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering* (2008), ACM, 2--12.

[17] Nagappan, N., Murphy, B., and Basili, V. The Influence of Organizational Structure on Software Quality: An Empirical Case Study. In *Proceedings of the 30th International Conference on Software Engineering* (2008), ACM, 521--530.

[18] Hassan, A.E. Predicting faults using the complexity of code changes. In *Proceedings of the 31st International Conference on Software Engineering* (2009), IEEE Computer Society, 78--88.

[19] Herzig, K. and Zeller, A. Mining cause-effect-chains from version histories. In *Software Reliability Engineering (ISSRE), 2011 IEEE 22nd International Symposium on* (2011), 60--69.

[20] Herzig, K., Just, S., Rau, A., and Zeller, A. Predicting Defects Using Change Genealogies. In *Proceedings of the 2013 IEEE 24nd International Symposium on Software Reliability Engineering* (2013), IEEE Computer Society.

[21] Hovsepyan, A., Scandariato, R., Joosen, W., and Walden, J. Software Vulnerability Prediction Using Text Analysis Techniques. In *Proceedings of the 4th International Workshop on Security Measurements and Metrics* (2012), ACM, 7–10.

[22] Shin, Y., Meneely, A., Williams, L., and Osborne, J.A. Evaluating Complexity, Code Churn, and Developer Activity Metrics as Indicators of Software Vulnerabilities. *Software Engineering, IEEE Transactions on*, 37 (2011), 772--787.

[23] Chowdhury, I. and Zulkernine, M. Using complexity, coupling, and cohesion metrics as early indicators of vulnerabilities. *Journal of Systems Architecture*, 57 (2011), 294--313.

[24] Giger, E., D'Ambros, M., Pinzger, M., and Gall, H.C. Method-level bug prediction. In *Empirical Software Engineering and Measurement (ESEM), 2012 ACM-IEEE International Symposium on* (2012), 171--180.

[25] Menzies, T., Dekhtyar, A., Distefano, J., and Greenwald, J. Problems with Precision: A Response to Comments on Data Mining Static Code Attributes to Learn Defect Predictors. *Software Engineering, IEEE Transactions on*, 33 (2007), 637--640.

[26] Premraj, R. and Herzig, K. Network Versus Code Metrics to Predict Defects: A Replication Study. In *Proceedings of the 2011 International Symposium on Empirical Software Engineering and Measurement* (2011), IEEE Computer Society, 215–224.

[27] Weyuker, E., Ostrand, T., and Bell, R. Comparing the effectiveness of several modeling methods for fault prediction. *Empirical Software Engineering*, 15, 277-295.

[28] Czerwonka, J., Nagappan, N., Schulte, W., and Murphy, B. CODEMINE: Building a Software Development Data Analytics Platform at Microsoft. *Software, IEEE*, 30, 4 (2013), 64--71.

[29] *Four Grand Challenges in Trustworthy Computing*. , 2003.

[30] Team, R.D.C. *R: A Language and Environment for Statistical Computing*. , 2010. R Foundation for Statistical Computing.

[31] Venables, W.N. and Ripley, B.D. *Modern Applied Statistics with S. Fourth Edition*. Springer, 2002.

[32] Pearson, K. LIII. On lines and planes of closest fit to systems of points in space. *Philosophical Magazine Series 6*, 2 (1901), 559-572.

[33] Kuhn, M. *caret: Classification and Regression Training*. , 2011.

[34] Witten, I.H. and Frank, E. Data mining: practical machine learning tools and techniques with Java implementations. *SIGMOD Rec.*, 31 (mar 2002), 76--77.

[35] Friedman, J., Hastie, T., and Tibshirani, R. *The Elements of Statistical Learning*. Springer Publishing Company, Incorporated, 2009.

[36] Nagappan, N., Ball, T., and Zeller, A. Mining Metrics to Predict Component Failures. In *Proceedings of the 28th International Conference on Software Engineering* (2006), ACM, 452--461.

[37] Dowd, M., McDonald, J., and Schuh, J. *The Art of Software Security Assessment: Identifying and Preventing Software Vulnerabilities*. Addison-Wesley Professional, 2006.

[38] Smith, B. and Williams, L. Using SQL Hotspots in a Prioritization Heuristic for Detecting All Types of Web Application Vulnerabilities. In *Software Testing, Verification and Validation (ICST), 2011 IEEE Fourth International Conference on* (March 2011), 220-229.

[39] Chawla, N.V. C4. 5 and imbalanced data sets: investigating the effect of sampling method, probabilistic estimate, and decision tree structure. In *Proceedings of the ICML* (2003).

[40] *Beautiful Evidence*. 2006.

10. APPENDIX

Table 3: Metrics Definitions

Metric	[Haystack]	Category	Definition
Added LOC		Churn	Lines of code added during the development cycle.
ChurnedLOC	Total Churn	Churn	Lines of code added, deleted or altered during the development cycle.
DeletedLOC		Churn	Lines of code deleted during the development cycle.
Editors	Number of Engineers	Churn	Count of unique users who have made changes to each file
NumberOfEdits	Edit Frequency	Churn	Count of distinct commits made to each file
RelativeChurn		Churn	Relative code measure, ratio of Churned LOC to LOC Project End
Complexity (Avg, Sum, Max)	Cyclomatic Complexity	Complexity	McCabe complexity measure, number of linearly-independent paths through each function, rolled up to source-file level
Arcs (Avg, Sum, Max)		Dependency	Number of transfer of control points between basic blocks defined in the file
FanIn (Avg, Sum, Max)	Fan-In	Dependency	Number of calls to functions within each source file
FanOut (Avg, Sum, Max)	Fan-Out	Dependency	Number of calls by functions within each source file to other functions
FanOutExternal (Avg, Sum, Max)		Dependency	Number of calls by functions within each source file to other functions outside the source file
Incoming Cross Binary		Dependency	Number of calls from outside a source file's binary to functions within the source file
Incoming Dependencies (Avg, Sum, Max)	Incoming Direct	Dependency	Number of function call, import, export, RPC, COM and Registry access dependencies on the source file
Outgoing Dependencies (Avg, Sum, Max)	Outgoing Direct	Dependency	Number of function call, import, export, RPC, COM and Registry access dependencies by the source file
AddedLOCSinceReset		Legacy	Lines of code added since the security reset date
AgeinWeeks		Legacy	Number of weeks since recorded file creation date of source file
ChurnedLOCSinceReset		Legacy	Lines of code added, deleted or altered between the security reset date and RTM
ChurnSinceReset		Legacy	Lines of code deleted or altered between the security reset date and RTM.
DeletedLOCSinceReset		Legacy	Lines of code deleted between the security reset date and RTM
LegacyLOCPct		Legacy	Relative code measure, percentage of LOC written prior to reset
LOC Pre-Reset		Legacy	Total lines of code in the file on the security reset date
NumberOfEditsSinceReset		Legacy	Count of distinct commits made to each file since the reset
Arguments (Avg, Sum, Max)		Size	Number of function arguments defined within the file
Blocks (Avg, Sum, Max)		Size	Number of basic blocks contained within the file (a block is a single, contiguous set of instructions with one entry, one exit and no branches)
Functions		Size	Number of functions defined within the file
LOC Project End	LOC	Size	Total lines of code in the file at Release-To-Manufacturing (RTM)
LOC Project Start		Size	Total lines of code in the file at the start of the development cycle
Locals (Avg, Sum, Max)		Size	Number of local variables defined within the file
Paths (Avg, Sum, Max)		Size	Number of paths
VulnCount_PreRelase		Defects	Number of vulnerabilities fixed prior software release
VulnCount_SecurityReview		Defects	Number of security related changes prior software release