An Empirical Study on Using the National Vulnerability Database to Predict Software Vulnerabilities

Su Zhang, Doina Caragea, and Xinming Ou

Kansas State University, {zhangs84,dcaragea,xou}@ksu.edu

Abstract. Software vulnerabilities represent a major cause of cyber-security problems. The National Vulnerability Database (NVD) is a public data source that maintains standardized information about reported software vulnerabilities. Since its inception in 1997, NVD has published information about more than 43,000 software vulnerabilities affecting more than 17,000 software applications. This information is potentially valuable in understanding trends and patterns in software vulnerabilities, so that one can better manage the security of computer systems that are pestered by the ubiquitous software security flaws. In particular, one would like to be able to predict the likelihood that a piece of software contains a yet-to-be-discovered vulnerability, which must be taken into account in security management due to the increasing trend in zero-day attacks. We conducted an empirical study on applying data-mining techniques on NVD data with the objective of predicting the time to next vulnerability for a given software application. We experimented with various features constructed using the information available in NVD, and applied various machine learning algorithms to examine the predictive power of the data. Our results show that the data in NVD generally have poor prediction capability, with the exception of a few vendors and software applications. By doing a large number of experiments and observing the data, we suggest several reasons for why the NVD data have not produced a reasonable prediction model for time to next vulnerability with our current approach.

Keywords: data mining, cyber-security, vulnerability prediction

1 Introduction

Each year a large number of new software vulnerabilities are discovered in various applications (see Figure 1). Evaluation of network security has focused on known vulnerabilities and their effects on the hosts and networks. However, the potential for unknown vulnerabilities (*a.k.a.* zero-day vulnerabilities) cannot be ignored because more and more cyber attacks utilize these unknown security holes. A zero-day vulnerability could last a long period of time (*e.g.* in 2010 Microsoft confirmed a vulnerability in Internet Explorer, which affected some versions that were released in 2001). Therefore, in order to have more accurate results on network security evaluation, one must consider the effect from zero-day vulnerabilities. The National Vulnerability Database (NVD) is a well-known data source for vulnerability information, which could be useful to estimate the likelihood that a specific application contains zero-day vulnerabilities based on historical information. We have adopted a data-mining approach in an attempt to build a prediction model for the attribute "time to next vulnerability" (TTNV), *i.e.* the time that it will take before the next vulnerability about a particular application will be found. The predicted TTNV metrics could be translated into the likelihood that a zero-day vulnerability exists in the software.

Past research has addressed the problem of predicting software vulnerabilities from different angles. Kyle *et al.* [10] pointed out the importance of estimating the risk-level of zero-day vulnerabilities. Mcqueen *et al.* [15] did experiments on estimating the number of zero-day vulnerabilities on each given day. Alhazmi and Malaiya [3] introdced the definition of TTNV. Ozment [19] did a number of studies on analyzing NVD, and pointed out several limitations of this database.

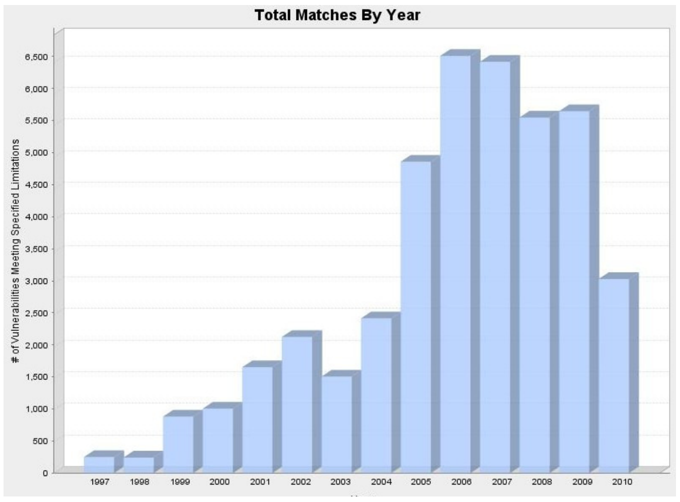

Fig. 1. The trend of vulnerability numbers

In this paper, we present our empirical experience of applying data-mining techniques on NVD data in order to build a prediction model for TTNV. We conduct a rigorous data analysis and experiment with a number of feature construction schemes and learning algorithms. Our results show that the data in NVD generally have poor prediction capability, with the exception of a few vendors and software applications. In the rest of the paper we will explain the features we have constructed and the various approaches we have taken in our attempts to build the prediction model. While it is generally a difficult task to show that data has no utility, our experience does indicate a number of reasons why it is unlikely to construct a reliable prediction model for TTNV given the information available in NVD.

2 Data Source – National Vulnerability Database

Each data entry in NVD consists of a large number of fields. We represent them as <D, CPE, CVSS>. D is a set of data including published time, summary of the vulnerability and external links about each vulnerability. CPE [6] and CVSS [21] will be described below.

2.1 CPE (Common Platform Enumeration)

CPE is *an open framework for communicating the characteristics and impacts of IT vulnerabilities.* It provides us with information on a piece of software, including version, edition, language, *etc.* An example is shown below:

```
cpe:/a:acme:product:1.0:update2:pro:en-us
    Professional edition of the "Acme Product 1.0 Update 2 English".
```

2.2 CVSS (Common Vulnerability Scoring System)

CVSS is a vulnerability scoring system designed to provide an open and standardized method for rating IT vulnerabilities. CVSS helps organizations prioritize and coordinate a joint response to security vulnerabilities by communicating the base, temporal and environmental properties of a vulnerability. Currently NVD only provides the Base group in its metric vector. Some components of the vector are explained below.

– *Access Complexity* indicates the difficulty level of the attack required to exploit the vulnerability once an attacker has gained access to it. It includes three levels: High, Medium, and Low.
– *Authentication* indicates whether an attacker must authenticate in order to exploit a vulnerability. It includes two levels: Authentication Required (R), and Authentication Not Required (NR).
– *Confidentiality, Integrity and Availability* are three loss types of attacks. Confidentiality loss means information will be leaked to people who are not supposed to know it. Integrity loss means the data can be modified illegally. Availability loss means the compromised system cannot perform its intended task or will crash. Each of the three loss types have the three levels: None (N), Partial (P), and Complete (C).

The *CVSS Score* is calculated based on the metric vector, with the objective of indicating the severity of a vulnerability.

3 Our Approach

We choose TTNV (time to next vulnerability) as the predicted feature. The predictive attributes are time, versiondiff (the distance between two different versions by certain measurement), software name and CVSS. All are derived or extracted directly from NVD.

3.1 Data Preparation and Preprocessing

Division of training/test data: As the prediction model is intended to be used to forecast future vulnerabilities, we divide the NVD data into training and test data sets based on the time the vulnerabilities were published. The ratio of the amount of training to test data is 2. We chose to use the data starting from 2005, as the data prior to this year looks unstable (see Figure 1).

Removing obvious errors: Some NVD entries are obviously erroneous (*e.g.* in one entry for Linux the kernel version was given as 390). To prevent these entries from polluting the learning process, we removed them from the database.

3.2 Feature Construction and Transformation

Identifying and constructing predictive features is of vital importance to data-mining. For the NVD data, intuitively Time and Version are two useful features. As we want to predict time to next vulnerability, the published time for each past vulnerability will be a useful information source. Likewise, the version information in each reported vulnerability could indicate the trend of vulnerability discovery, as new versions are released. Although both Time and Version are useful information sources, they need to be transformed to provide the expected prediction behavior. For example, both features in their raw form increase monotonically. Directly using the raw features will provide little prediction capability for future vulnerabilities. Thus, we introduce several feature construction schemes for the two fields and studied them experimentally.

Time: We investigated two schemes for constructing time features. One is epoch time, the other is using month and day separately without year. Like explained before, the epoch time is unlikely to provide useful prediction capability, as it increases monotonically. Intuitively, the second scheme shall be better, as the month and day on which a vulnerability is published may show some repeating pattern, even in future years.

Version: We calculate the difference between the versions of two adjacent instances and use the versiondiff as a predictive feature. An instance here refers to an entry where a specific version of an application contains a specific vulnerability. The rationale for using versiondiff as a predictive feature is that we want to use the trend of the versions with time to estimate future situations. Two versiondiff schemas are introduced in our approach. The first one is calculating the versiondiff based on version counters (rank), while the second is calculating the versiondiff by radix.

Counter versiondiff: For this versiondiff schema, differences between minor versions and differences between major versions are treated similarly. For example, if one software has three versions: 1.1, 1.2, 2.0, then the versions will be assigned counters 1, 2, 3 based on the rank of their values. Therefore, the versiondiff between 1.1 and 1.2 is the same as the one between 1.2 and 2.0.

Radix-based versiondiff: Intuitively, the difference between major versions is more significant than the difference between minor versions. Thus, when calculating versiondiff, we need to assign a higher weight to relatively major version changes and lower weight to relatively minor version changes. For example, for the three versions 1.0, 1.1, 2.0, if we assign a weight of 10 to the major version and a weight of 1 to each minor version, the versiondiff between 1.1 and 1.0 will be 1, while the versiondiff between 2.0 and 1.1 will be 9.

When analyzing the data, we found out that versiondiff did not work very well for our problem because, in most cases, the new vulnerabilities affect all previous versions as well. Therefore, most values of versiondiff are zero, as the new vulnerability instance must affect an older version that also exists in the previous instance, thus, resulting in a versiondiff of zero. In order to mitigate this limitation, we created another predictive feature for our later experiments. The additional feature that we constructed is the number of occurrences of a certain version of each software. More details will be provided in Section 4.

3.3 Machine Learning Functions

We used either classification or regression functions for our prediction, depending on how we define the predicted feature. The TTNV could be a number representing how many days we need to wait until the occurrence of the next vulnerability. Or it could be binned and each bin stands for values within a range. For the former case, we used regression functions. For the latter case, we used classification functions. We used WEKA [5] implementations of machine learning algorithms to build predictive models for our data. For both regression and classification cases, we explored all of the functions compatible to our data type, with default parameters. In the case of the regression problem, the compatible functions are: linear regression, least median square, multi-layer perceptron, RBF network, SMO regression, and Gaussian processes. In the case of classification, the compatible functions are: logistic, least median square, multi-layer perceptron, RBF network, SMO, and simple logistic.

4 Experimental Results

We conducted the experiments on our department's computer cluster - Beocat. We used a single node and 4G RAM for each experiment. As mentioned above, WEKA [5], a data-mining suite, was used in all experiments.

4.1 Evaluation Metrics

A number of metrics are used to evaluate the performance of the predictive models learned.

Correlation Coefficient: The correlation coefficient is a measure of how well trends in the predicted values follow trends in actual values. It is a measure of how well the predicted values from a forecast model "fit" the real-life data. The correlation coefficient is a number between -1 and 1. If there is no relationship between the predicted values and the actual values, the correlation coefficient is close to 0 (i.e., the predicted values are no better than random numbers). As the strength of the relationship between the predicted values and actual values increases, so does the correlation coefficient. A perfect fit gives a coefficient of 1.0. Opposite but correlated trends result in a correlation coefficient value close to -1. Thus, the higher the absolute value of the correlation coefficient, the better; however, when learning a predictive model, negative correlation values are not usually expected. We generate correlation coefficient values as part of the evaluation of the regression algorithms used in our study.

Root Mean Squared Error: The mean squared error (MSE) of a predictive regression model is another way to quantify the difference between a set of predicted values, x_p, and the set of actual (target) values, x_t, of the attributed being predicted. The root mean squared error (RMSE) can be defined as:

$$\text{RMSE}(x_p,\, x_t) = \sqrt{MSE(x_p, x_t)} = \sqrt{E[(x_p - x_t)^2]} = \sqrt{\frac{\sum_{i=1}^{n}(x_{p,i} - x_{t,i})^2}{n}}$$

Root Relative Squared Error: According to [1], the root relative squared error (RRSE) is relative to what the error would have been if a simple predictor had been used. The simple predictor is considered to be the mean/majority of the actual values. Thus, the relative squared error takes the total squared error and normalizes it by dividing by the total squared error of the simple predictor. By taking the root of the relative squared error one reduces the error to the same dimensions as the quantity being predicted.

$$\text{RRSE}(x_p,\, x_t) = \sqrt{\frac{\sum_{i=1}^{n}\left(x_{p,i} - x_{t,i}\right)^2}{\sum_{i=1}^{n}\left(x_{t,i} - \bar{x}\right)^2}}$$

Correctly Classified Rate: To evaluate the classification algorithms investigated in this work, we use a metric called correctly classified rate. This metric is obtained by dividing correctly classified instances by all instances. Obviously, a higher value suggests a more accurate classification model.

4.2 Experiments

We performed a large number of experiments, by using different versiondiff schemes, different time schemes, and by including CVSS metrics or not. For different software, different feature combinations produce the best results. Hence, we believe it is not effective to build a single model for all the software. Instead, we build separate models for different software. This way, we also avoid potential scalability issues due to the large number of nominal type values from vendor names and software names.

Given the large number of vendors in the data, we did not run experiments for all of them. We focused especially on several major vendors (Linux, Microsoft, Mozilla and Google) and built vendor-specific models. For three vendors (Linux, Microsoft and Mozilla), we also built software-specific models. For other vendors (Apple, Sun and Cisco), we bundled all their software in one experiment.

4.3 Results

Linux: We used two versiondiff schemes, specifically counter-based and radix-based, to find out which one is more effective for our model construction. We also compared two different time schemes (epoch time, and using month and day separately). In a first set of experiments, we predicted TTNV based on regression models. In a second set of experiments, we grouped the predictive feature (TTNV) values into bins, as we observed that the TTNV distribution shows several distinct clusters, and solved a classification problem.

Table 1 shows the results obtained using the *epoch time* scheme versus the results obtained using the *month and day* scheme, in terms of correlation coefficient, for regression models. As can be seen, the results of our experiments did

Table 1. Correlation Coefficient for Linux Vulnerability Regression Models Using Two Time Schemes

Regression Functions	Epoch time		Month and day	
	training	test	training	test
Linear regression	0.3104	0.1741	0.6167	-0.0242
Least mean square	0.1002	0.1154	0.1718	0.1768
Multi-layer perceptron	0.2943	0.1995	0.584	-0.015
RBF network	0.2428	0	0.1347	0.181
SMO regression	0.2991	0.2186	0.2838	0.0347
Gaussian processes	0.3768	-0.0201	0.8168	0.0791

not show a significant difference between the two time schemes that we used, although we expected the month and day feature to provide better results than the absolute epoch time, as explained in Section 3.2. Thus, neither scheme has acceptable correlation capability on the test data. We adapted the month and day time schema for all of the following experiments.

Table 2 shows a comparison of the results of the two different versiondiff schemes. As can be seen, both perform poorly as well. Given the unsatisfactory results, we believed that the large number of Linux sub-versions could be potentially a problem. Thus, we also investigated constructing the versiondiff feature by binning versions of the Linux kernel (to obtained a smaller set of sub-versions). We round each sub-version to its third significant major version (*e.g.* Bin(2.6.3.1) = 2.6.3). We bin based on the first three most significant versions because more than half of the instances (31834 out of 56925) have version longer than 3, and Only 1% (665 out of 56925) versions are longer than 4. Also, the difference on the third subversion will be regarded as a huge dissimilarity for Linux kernels. We should note that the sub-version problem may not exist for other vendors, such as Microsoft, where the versions of the software are naturally discrete (all Microsoft products have versions less than 20). Table 3 shows the comparisons between regression models that use binned versions versus regression models that do not use binned versions. The results are still not good enough as many of the versiondiff values are zero, as explained in Section 3.2 (new vulnerabilities affect affect previous versions as well).

Table 2. Correlation Coefficient for Linux Vulnerability Regression Models Using Two Versiondiff Schemes

Regression Functions	Version counter		Radix based	
	training	test	training	test
Linear regression	0.6167	-0.0242	0.6113	0.0414
Least mean square	0.1718	0.1768	0.4977	-0.0223
Multi-layer perceptron	0.584	-0.015	0.6162	0.1922
RBF network	0.1347	0.181	0.23	0.0394
SMO regression	0.2838	0.0347	0.2861	0.034
Gaussian processes	0.8168	0.0791	0.6341	0.1435

TTNV Binning: Since we found that the feature (TTNV) of Linux shows distinct clusters, we divided the feature values into two categories, more than 10 days and no more than 10 days, thus transforming the original regression problem into an easier binary classification problem. The resulting models are evaluated in terms of corrected classified rates, shown in Table 4. While the models are better in this case, the false positive rates are still high (typically above 0.4). In this case, as before, we used default parameters for all classification functions. However, for the SMO function, we also used the Gaussian (RBF) kernel. The results of the SMO (RBF kernel) classifier are better than the results of most

Table 3. Correlation Coefficient for Linux Vulnerability Regression Models Using Binned Versions versus Non-Binned Versions

Regression Functions	Non-binned versions		Binned versions	
	training	test	training	test
Linear regression	0.6113	0.0414	0.6111	0.0471
Least mean square	0.4977	-0.0223	0.5149	0.0103
Multi-layer perceptron	0.6162	0.1922	0.615	0.0334
RBF network	0.23	0.0394	0.0077	-0.0063
SMO regression	0.2861	0.034	0.285	0.0301
Gaussian processes	0.6341	0.1435	0.6204	0.1369

other classifiers, in terms of correctly classified rate. However, even this model has a false positive rate of 0.436, which is far from acceptable.

Table 4. Correctly Classified Rates for Linux Vulnerability Classification Models Using Binned TTNV

Classification Functions	Correctly classified		FPR	TPR
	training	test		
Simple logistic	97.6101%	69.6121%	0.372	0.709
Logistic regression	97.9856%	57.9542%	0.777	0.647
Multi-layer perceptron	98.13%	64.88%	0.689	0.712
RBF network	95.083%	55.18%	0.76	0.61
SMO	97.9061%	61.8259%	0.595	0.658
SMO (RBF kernel)	96.8303%	62.8392%	0.436	0.641

CVSS Metrics: In all cases, we also perform experiments by adding CVSS metrics as predictive features. However, we did not see much differences.

Microsoft: As we have already observed the limitation of versiondiff scheme in the analysis of Linux vulnerabilities, for Microsoft instances, we use only the number of occurrences of a certain version of a software or occurrences of a certain software, instead of using versiondiff, as described below. We analyzed the set of instances and found out that more than half of the instances do not have version information. Most of these case are Windows instances. Most of the non-Windows instances (more than 70%) have version information. Therefore, we used two different occurrence features for these two different types of instances. For Windows instances, we used the occurrence of each software as a predictive feature. For non-Windows instances, we used the occurrence of each version of the software as a predictive feature.

Also based on our observations for Linux, we used only the month and day scheme, and did not use the epoch time scheme in the set of experiments we performed for Windows. We analyzed instances to identify potential clusters of

TTNV values. However, we did not find any obvious clusters for either windows or non-windows instances. Therefore, we only used regression functions. The results obtained using the aforementioned features for both Windows and non-Windows instances are presented in Table 5. As can be seen, the correlation coefficients are still less than 0.4.

Table 5. Correlation Coefficient for Windows and Non-Windows Vulnerability Regression Models, Using Occurrence Version/Software Features and Day and Month Time Scheme

Regression Functions	Win Instances		Non-win Instances	
	training	test	training	test
Linear regression	0.4609	0.1535	0.5561	0.0323
Least mean square	0.227	0.3041	0.2396	0.1706
Multi-layer perceptron	0.7473	0.0535	0.5866	0.0965
RBF network	0.1644	0.1794	0.1302	-0.2028
SMO regression	0.378	0.0998	0.4013	-0.0467
Gaussian processes	0.7032	-0.0344	0.7313	-0.0567

We further investigated the effect of building models for individual non-Windows applications. For example, we extracted Internet Explorer (IE) instances and build several models for this set. When CVSS metrics are included, the correlation coefficient is approximately 0.7. This is better than when CVSS metrics are not included, in which case, the correlation coefficient is approximately 0.3. The results showing the comparison between IE models with and without CVSS metrics is shown in Table 6. We tried to performed a similar experiment for Office. However, there are only 300 instances for Office. Other office-related instances are about individual software such as Word, PowerPoint, Excel and Access, *etc*, and each has less than 300 instances. Given the small number of instances, we could not build models for Office.

Table 6. Correlation Coefficient for IE Vulnerability Regression Models, with and without CVSS Metrics

Regression Functions	With CVSS		Without CVSS	
	training	test	training	test
Linear regression	0.8023	0.6717	0.7018	0.3892
Least mean square	0.6054	0.6968	0.4044	0.0473
Multi-layer perceptron	0.9929	0.6366	0.9518	0.0933
RBF network	0.1381	0.0118	0.151	-0.1116
SMO regression	0.7332	0.5876	0.5673	0.4813
Gaussian processes	0.9803	0.6048	0.9352	0.0851

Mozilla: At last, we built classification models for Firefox, with and without the CVSS metrics. The results are shown in Table 7. As can be seen, the correctly classified rates are relatively good (approximately 0.7) in both cases. However, the number of instances in this dataset is rather small (less than 5000), therefore it is unclear how stable the prediction model is.

Table 7. Correctly Classified Rate for Firefox Vulnerability Models with and without CVSS Metrics

Classification Functions	With CVSS		Without CVSS	
	training	test	training	test
Simple logistic	97.5%	71.4%	97.5%	71.4%
Logistic regression	97.5%	70%	97.8%	70.5%
Multi-layer perceptron	99.5%	68.4%	99.4%	68.3%
RBF network	94.3%	71.9%	93.9%	67.1%
SMO	97.9%	55.3%	97.4%	55.3%

4.4 Parameter Tuning

As mentioned above, we used default parameters for all regression and classification models that we built. To investigate if different parameter settings could produce better results, we chose to tune parameters for the support vector machines algorithm (SVM), whose WEKA implementations for classifications and regression are called SMO and SMO regression, respectively. There are two main parameters that can be tuned for SVM, denoted by C and σ. The C parameter is a cost parameter which controls the trade-off between model complexity and training error, while σ controls the width of the Gaussian kernel [2].

To find the best combination of values for C and σ, we generated a grid consisting of the following values for C: 0.5, 1.0, 2.0, 3.0, 5.0, 7.0, 10, 15, 20 and the following values for σ: 0, 0.05, 0.1, 0.2, 0.3, 0.5, 1.0, 2.0, 5.0, and run the SVM algorithm for all possible combinations. We used a separate validation set to select the combination of values that gives the best values for correlation coefficient, and root squared mean error and root relative squared error together. The validation and test datasets have approximately equal sizes; the test set consists of chronologically newer data, as compared to the validation data, while the validation data is newer than the training data.

Table 8 shows the best parameter values when tuning was performed based on the correlation coefficient, together with results corresponding to these parameter values, in terms of correlation coefficient, RRSE and RMSE (for both validation and test datasets). Table 9 shows similar results when parameters are tuned on RRSE and RMSE together.

Table 8. Parameter Tuning Based on Correlation Coefficient

Group Targeted	Parameters		Validation			Test		
	C	G	RMSE	RRSE	CC	RMSE	RRSE	CC
Adobe CVSS	3.0	2.0	75.2347	329.2137%	0.7399	82.2344	187.6%	0.4161
IE CVSS	1.0	1.0	8.4737	74.8534%	0.4516	11.6035	92.2%	-0.3396
Non-Windows	1.0	0.05	92.3105	101.0356%	0.1897	123.4387	100.7%	0.223
Linux CVSS	15.0	0.1	12.6302	130.8731%	0.1933	45.0535	378.3%	0.2992
Adobe	0.5	0.05	43.007	188.1909%	0.5274	78.2092	178.5%	0.1664
IE	7.0	0.05	13.8438	122.2905%	0.2824	14.5263	115.5%	-0.0898
Apple Separate	3.0	0.05	73.9528	104.0767%	0.2009	91.1742	116.4%	-0.4736
Apple Single	0.5	0.0	493.6879	694.7868%	0	521.228	1401.6%	0
Linux Separate	2.0	0.05	16.2225	188.6665%	0.3105	49.8645	418.7%	-0.111
Linux Single	1.0	0.05	11.3774	83.2248%	0.5465	9.4743	79.6%	0.3084
Linux Binned	2.0	0.05	16.2225	188.6665%	0	49.8645	418.7%	-0.111
Windows	5	0.05	21.0706	97.4323%	0.1974	72.1904	103.1%	0.1135

Table 9. Parameter Tuning Based on RMSE and RRSE

Group Targeted	Parameters		Validation			Test		
	C	G	RMSE	RRSE	CC	RMSE	RRSE	CC
Adobe CVSS	0.5	0.2	19.4018	84.8989%	0.2083	61.2009	139.6667%	0.5236
IE CVSS	2.0	1.0	8.4729	74.8645%	0.4466	11.4604	91.1018%	-0.3329
Non-Windows	0.5	0.1	91.1683	99.7855%	0.188	123.5291	100.7%	0.2117
Linux CVSS	2.0	0.5	7.83	81.1399%	0.1087	19.1453	160.8%	0.3002
Adobe	1.0	0.5	19.5024	85.3392%	-0.4387	106.2898	242.5%	0.547
IE	0.5	0.3	12.4578	110.0474%	0.2169	13.5771	107.9%	-0.1126
Apple Separate	7.0	1.0	70.7617	99.5857%	0.1325	80.2045	102.4%	-0.0406
Apple Single	0.5	0.05	75.9574	106.8979%	-0.3533	82.649	105.5%	-0.4429
Linux Separate	0.5	2.0	14.5428	106.3799%	0.2326	18.5708	155.9%	0.1236
Linux Single	5.0	0.5	10.7041	78.2999%	0.4752	12.3339	103.6%	0.3259
Linux Binned	0.5	2.0	14.5428	106.3799%	0.2326	18.5708	155.9%	0.1236
Windows	5.0	0.05	21.0706	97.4323%	0.1974	72.1904	103%	0.1135

4.5 Summary

The experiments above indicate that it is hard to build good prediction models based on the limited data available in NVD. For example, there is no version information for most Microsoft instances (especially, Windows instances). Some results look promising (*e.g.* the models we built for Firefox), but they are far from usable in practice. Below, we discuss what we believe to be the main reasons for the difficulty of building good prediction models for TTNV from NVD.

4.6 Discussion

We believe the main factor affecting the predictive power of our models is the low quality of the data from the National Vulnerability Database. Following are several limitations of the data:

- Missing information: most instances of Microsoft do not have the version information, without which we could not observe how the number of vulnerabilities evolves over versions.
- "Zero" versiondiffs: most of versiondiff values are zero because earlier-version applications are also affected by the later-found vulnerabilities (this is assumed by a number of large companies, *e.g.* Microsoft and Adobe) and significantly reduces the utility of this feature.
- Vulnerability release time: The release date of vulnerability could largely be affected by vendors' practices. For example, Microsoft usually releases their vulnerability and patch information on the second Tuesday of each month, which may not accurately reflect the discovery date of the vulnerabilities.
- Data error: We found a number of obvious errors in NVD, such as the aforementioned Linux kernel version error.

5 Related Works

Alhazmi and Malaiya [3] have addressed the problem of building models for predicting the number of vulnerabilities that will appear in the future. They targeted operating systems instead of applications. The Alhazmi-Malaiya Logistic model works well for fitting existing data, when evaluated in terms of average error (AE) and average bias (AB) of number of vulnerabilities over time. However, fitting existing data is a prerequisite of testing models: predictive power is the most important criteria [18] . They did test the predictive accuracy of their models and got satisfactory results [18].

Ozment [19] examined the vulnerability discovery models (proposed by Alhazmi Malaiya [3]) and pointed some limitations that make these models inapplicable. One of them is that there is not enough information included in a government supported vulnerability database (*e.g.* National Vulnerability Database). This is confirmed by our empirical study.

McQueen *et al.* [15] designed algorithms for estimating the number of zero-day vulnerabilities on each given day. This number can indicate the overall risk level from zero-day vulnerabilities. However, for different applications the risks could be different. Our work aimed to construct software-specific prediction models.

Massacci *et al.* [14, 16] compared several existing vulnerability databases based on the type of vulnerability features available in each of them. They mentioned that many important features are not included in most databases. *e.g.* discovery date is hard to find. Even though certain databases (such as OSVDB that as we also studied) claim they include the features, most of the entries are blank. For their Firefox vulnerability database, they employed textual retrieval techniques and took keywords from CVS developer's commit log to get several other features by cross-referencing through CVE ids. They showed that by using two different data sources for doing the same experiment, the results could be quite different due to the high degree of inconsistency in the data available for the research community at the current time. They further tried to confirm the correctness of their database by comparing data from different sources. They used data-mining techniques (based on the database they built) to prioritize the security level of software components for Firefox.

Ingols *et al.* [10] tried to model network attacks and countermeasures using attack graphs. They pointed out the dangers from zero-day attacks and also mentioned the importance of modeling them. There has been a long line of attack-graph works [4, 7–9, 11–13, 17, 20, 22] which can potentially benefit from the estimation of the likelihood of zero-day vulnerabilities in specific applications.

6 Conclusions

In this paper we present our effort in building prediction models for zero-day vulnerabilities based on the information contained in the National Vulnerability Database. Our research found that due to a number of limitations of this data source, it is unlikely that one can build a practically usable prediction model at this time. We presented our rigorous evaluation of various feature construction schemes and parameter tuning for learning algorithms, and notice that none of the results obtained shows acceptable performance. We discussed possible reasons as of why the data source may not be well suited to predict the desired features for zero-day vulnerabilities.

References

1. Root relative squared error. Website. `http://www.gepsoft.com/gxpt4kb/Chapter10/Section1/SS07.htm`.
2. Support vector machines. Website. `http://www.dtreg.com/svm.htm`.
3. Omar H. Alhazmi and Yashwant K. Malaiya. Prediction capabilities of vulnerability discovery models. In *Annual Reliability and Maintainability Symposium (RAMS)*, 2006.
4. Paul Ammann, Duminda Wijesekera, and Saket Kaushik. Scalable, graph-based network vulnerability analysis. In *9th ACM Conference on Computer and Communications Security(CCS)*, 2002.
5. Remco R. Bouckaert, Eibe Frank, Mark Hall, Richard Kirkby, Peter Reutemann, Alex Seewald, and David Scuse. *WEKA Manual for Version 3.7. The University of Waikato*. The University of Waikato, 2010.
6. Andrew Buttner and Neal Ziring. Common platform enumeration (cpe) c specification. Technical report, The MITRE Corporation AND National Security Agency, 2009.
7. Marc Dacier, Yves Deswarte, and Mohamed Kaâniche. Models and tools for quantitative assessment of operational security. In *IFIP SEC*, 1996.
8. J. Dawkins and J. Hale. A systematic approach to multi-stage network attack analysis. In *Proceedings of Second IEEE International Information Assurance Workshop*, pages 48 – 56, April 2004.
9. Rinku Dewri, Nayot Poolsappasit, Indrajit Ray, and Darrell Whitley. Optimal security hardening using multi-objective optimization on attack tree models of networks. In *14th ACM Conference on Computer and Communications Security (CCS)*, 2007.

10. Kyle Ingols, Matthew Chu, Richard Lippmann, Seth Webster, and Stephen Boyer. Modeling modern network attacks and countermeasures using attack graphs. In *25th Annual Computer Security Applications Conference (ACSAC)*, 2009.
11. Kyle Ingols, Richard Lippmann, and Keith Piwowarski. Practical attack graph generation for network defense. In *22nd Annual Computer Security Applications Conference (ACSAC)*, Miami Beach, Florida, December 2006.
12. Sushil Jajodia, Steven Noel, and Brian O'Berry. Topological analysis of network attack vulnerability. In V. Kumar, J. Srivastava, and A. Lazarevic, editors, *Managing Cyber Threats: Issues, Approaches and Challanges*, chapter 5. Kluwer Academic Publisher, 2003.
13. Richard Lippmann and Kyle W. Ingols. An annotated review of past papers on attack graphs. Technical report, MIT Lincoln Laboratory, March 2005.
14. Fabio Massacci and Viet Hung Nguyen. Which is the right source for vulnerability studies? an empirical analysis on mozilla firefox. In *MetriSec*, 2010.
15. Miles McQueen, Trever McQueen, Wayne Boyer, and May Chaffin. Empirical estimates and observations of 0day vulnerabilities. In *42nd Hawaii International Conference on System Sciences*, 2009.
16. Viet Hung Nguyen and Le Minh Sang Tran. Predicting vulnerable software components with dependency graphs. In *MetriSec*, 2010.
17. Xinming Ou, Wayne F. Boyer, and Miles A. McQueen. A scalable approach to attack graph generation. In *13th ACM Conference on Computer and Communications Security (CCS)*, pages 336–345, 2006.
18. Andy Ozment. Improving vulnerability discovery models analyzer. In *QoP07*, 2007.
19. Andy Ozment. *Vulnerability Discovery & Software Security*. PhD thesis, University of Cambridge, 2007.
20. Cynthia Phillips and Laura Painton Swiler. A graph-based system for network-vulnerability analysis. In *NSPW '98: Proceedings of the 1998 workshop on New security paradigms*, pages 71–79. ACM Press, 1998.
21. Mike Schiffman, Gerhard Eschelbeck, David Ahmad, Andrew Wright, and Sasha Romanosky. *CVSS: A Common Vulnerability Scoring System*. National Infrastructure Advisory Council (NIAC), 2004.
22. Oleg Sheyner, Joshua Haines, Somesh Jha, Richard Lippmann, and Jeannette M. Wing. Automated generation and analysis of attack graphs. In *Proceedings of the 2002 IEEE Symposium on Security and Privacy*, pages 254–265, 2002.