Information Security Journal: A Global Perspective, 24:194–206, 2015

ISSN: 1939-3555 print / 1939-3547 online
DOI: 10.1080/19393555.2015.1111961

Taylor & Francis
Taylor & Francis Group

Predicting Cyber Risks through National Vulnerability Database

Su Zhang[1], Xinming Ou[2], and Doina Caragea[3]
[1]Cloud Platform Engineering, Symantec Corporation, Mountain View, California, USA
[2]Department of Computer Science and Engineering, University of South Florida, Tampa, Florida, USA
[3]Department of Computing and Information Sciences, Kansas State University, Manhattan, Kansas, USA

ABSTRACT Software vulnerabilities are the 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. We suggest possible reasons for why the NVD data have not produced a reasonable prediction model for time to next vulnerability with our current approach, and suggest alternative ways in which the data in NVD can be used for the purpose of risk estimation.

KEYWORDS risk assessment, zero-day vulnerability

1. INTRODUCTION

More and more vulnerabilities are being discovered every day (see Figure 1). Currently, evaluation for known vulnerabilities is becoming more and more mature. Figure 2 shows our current framework for known vulnerabilities evaluation (Xinming et al., 2005). However, evaluation for unknown vulnerabilities (a.k.a. zero-day vulnerabilities) should not be ignored because more and more cyber-attacks come from these unknown security holes and last a long period of time (e.g., in 2010 Microsoft confirmed a vulnerability in Internet Explorer (IE), which affected some versions that were released in 2001). Therefore, in order to have more accurate results on network security evaluation, we must consider

Address correspondence to Su Zhang, Symantec Corporation, 350 Ellis St., Mountain View, CA 94043, USA.
E-mail: westlifezs@gmail.com

Color versions of one or more of the figures in the article can be found online at www.tandfonline.com/uiss.

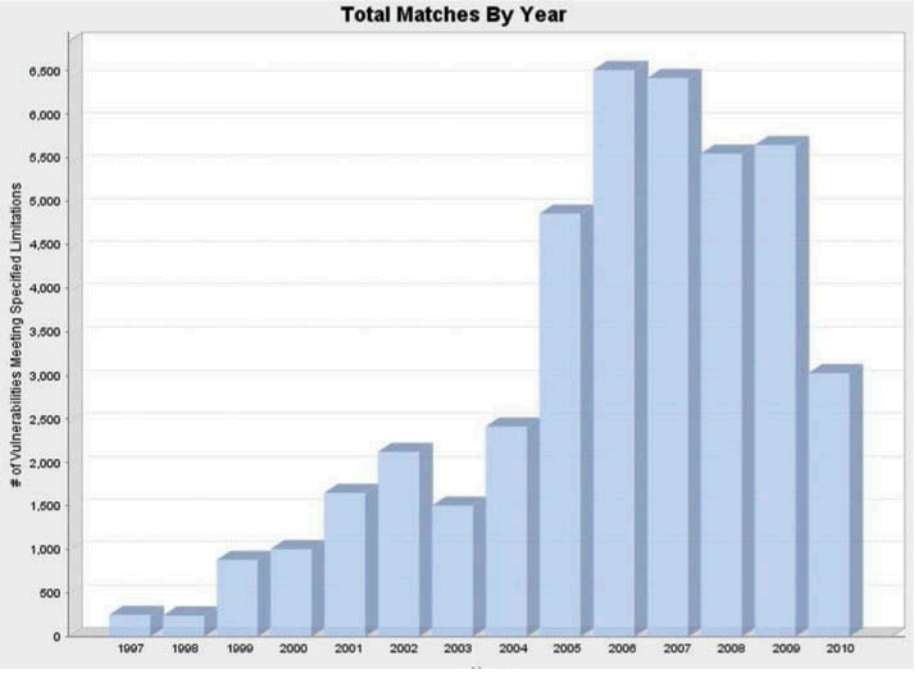

FIGURE 1 The trend of vulnerability numbers.

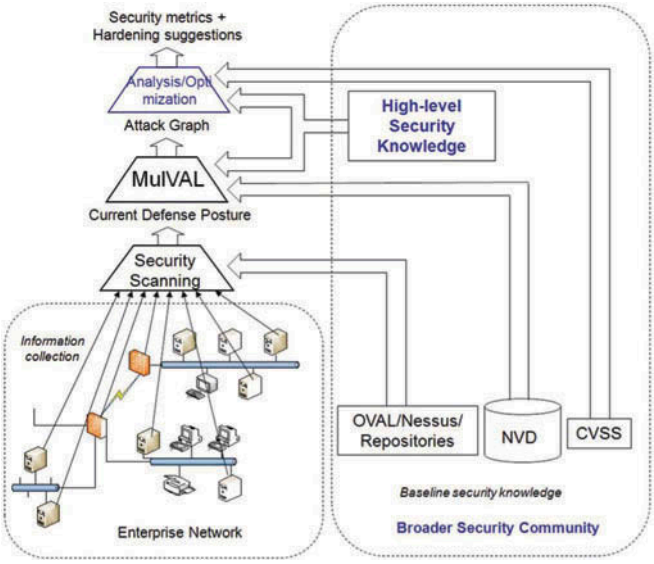

FIGURE 2 Enterprise risk assessment framework.

the effect from zero-day vulnerabilities. National Vulnerability Database (NVD) is a well-known database for vulnerability information, which could be a useful source for estimating zero-day vulnerabilities through data-mining technique. Time to next vulnerability is the target attribute of our experiments, and we believe we can use time to next vulnerability (TTNV) to quantify the risk-level of software. This is because the shorter time to see a software's next vulnerability, the higher vulnerability density it has, which implies higher risks. Also, a couple of related works have been done. Ingols et al. (2009) pointed out the importance of estimating the risk-level of zero-day vulnerability. McQueen et al. (2009) did experiments on estimating the number of zero-day vulnerabilities on each given day. Alhazmi and Malaiya (2006) points out the definition of TTNV which inspired us of the predicted attribute. Ozment (2007) did much work on analyzing NVD and pointed out several limitations of this database, such as the released date is not accurate enough, the data lack of consistency, etc. A short version of this work has been published in Zhang et al. (2011). Zhang et al. by proposing how to use the risk prediction model under a bigger picture (along with other risk assessment tools such as attack graph). Other than that, more completed experiments have been conducted in this paper. The goal is to take zero day vulnerabilities into consideration while evaluating software risks as shown in Figure 3.

2. DATA SOURCE – NATIONAL VULNERABILITY DATABASE

NVD is a collection of data. The tuples included in it always looks like <D, CPE, CVSS>.

- D is a set of data including published time, summary of the vulnerability and external link about each vulnerability.

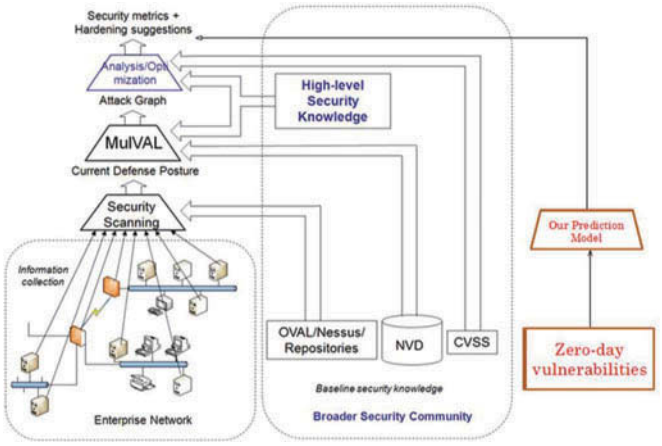

FIGURE 3 Our prediction model.

- CPE (Buttner & Ziring, 2009) stands for Common Platform Enumeration, which will be introduced in 2.1.
- CVSS (Mell, Scarfone, & Romanosky, 2007) represents Common Vulnerability Scoring System, which will be described later as well.

2.1. CPE (Common Platform Enumeration)

- **An open framework for communicating the characteristics and impacts of information technology (IT) vulnerabilities.**
  A CPE can provide us with information of a software, it may include version, edition, language, etc.
- **Example (in primitive format).**
  cpe :/a:acme:product:1.0:update2:pro:en-us
  Professional edition of the "Acme Product 1.0 Update 2 English."

2.2. 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.

- **Metrics Vector**
- Access Complexity indicates the difficult level of the attack required to exploit the vulnerability once an attacker has gained access to the target system. It includes the following three levels in CVSS: H: High M: Medium L: Low
- Authentication indicates whether an attacker must authenticate to a target in order to exploit a vulnerability. It includes the following two levels in CVSS: R: Authentication Required NR: Authentication Not Required
- Confidentiality, Integrity and Availability are three loss types of attacks. Confidentiality lost means information leaked to other people who are not supposed to know it. Integrity lost means the data was modified during transmitting. Availability lost means the server or host received too many requests that they couldn't response any of them in time. All of the three lost types have three following levels in CVSS: N: None P: Partial C: Complete
- **CVSS Score**
  Calculated based on above vector. It indicates the severity of a vulnerability.

2.3. Six Vendors with most NVD Instances

There are many vendors in NVD, with six of them obvious in terms of the instances number (Figure 4):

- Linux: 56925 instances
- Sun: 24726 instances
- Cisco: 20120 instances
- Mozilla: 19965 instances
- Microsoft: 16703 instances
- Apple: 14809 instances

They ranked really high (all within top 8) in terms of vulnerabilities number.

The reason we chose the Six Most Vulnerable/Popular (by number of instances) vendors is because (see Figure 5):

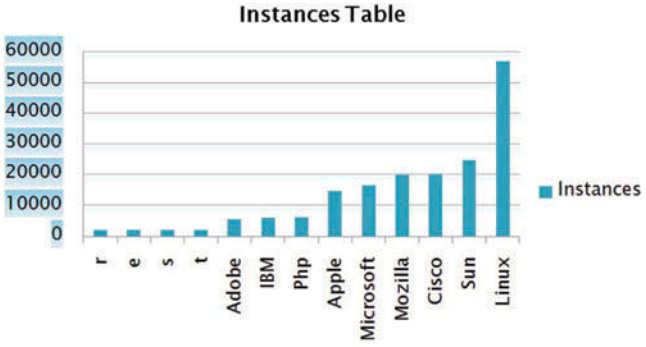

FIGURE 4 Vendors' ranking by number of instances.

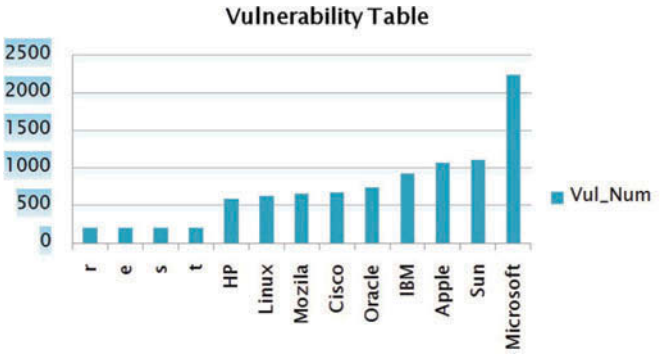

FIGURE 5 Vendors' ranking by number of vulnerabilities.

- Huge size of nominal types (vendors and software) will result in a scalability issue. If we try to build one model for all vendors, then there are too many possibilities (*10,411*) in vendor. This will run for too long to construct our models. Besides, many of smaller vendors have too few instances, which is not enough for our data-mining work.
- Top six take up 43.4% of all instances. The six vendors will take up nearly half of all instances. (Remember, we have 10,411 vendors totally.)
- The seventh vendor (in terms of instances number) is much less than the sixth. The next vendor has fewer instances, so we did not go further than the top six vendors.

We did vendor-based experiments on each vendor because vendors are independent for our approach. It is hard to find relations between vulnerabilities belonging to different vendors. So it will make sense if we build models for each vendor's instances.

Among the aforementioned six vendors, We put most of our effort on the most two vulnerable vendors—Linux and Microsoft (number one in instances number and vulnerabilities number). We also investigated Google, but the time line is an essential limitation for our experiments (most of chrome's instances came up between April and May 2010. Chrome has the largest number of instances among all applications of Google).

3. OUR APPROACH

3.1. Predicted Attribute-Time to Next Vulnerability

We tried to estimate the risk-level comes from zero-day vulnerabilities through data-mining techniques. Therefore, we would like some quantitative output. It could mean "the potential risk level of a given software." Also, because time is a quantitative indicator, we chose to predict TTNV (time to next vulnerability) as our predicted feature. Predictive attributes are time, versiondiff, software name, and CVSS. All of them are derived or extracted from NVD.

3.2. Data Preprocessing

- NVD data training/Testing dataset: For Linux, we chose to start from 2005 since before that the data looks unstable (see Figure 1).
- For some of later coming software-based models construction, since we have limited number of instances, we always use a ratio that training data to testing data is 2 (the data is split based on published time. Earlier data are for training and later data is for testing).
- Remove obvious errors in NVD (e.g., cpe:/o:linux:linux_kernel:390).
- Preprocessing for predictive features, including:
  - Published Date Time → Month
  - Published Date Time → Day
  - Two adjacent vulnerabilities CPE diff (vl,v2) → Versiondiff
  - CPE Specification → Software Name
  - Adjacent different Published Date Time → TTPV (Time to Previous Vulnerability)
  - Adjacent different Published Date Time → TTNV

3.3. Feature Construction and Transformation

Feature construction is of vital importance for all data-mining works. We analyzed the source data and realized time and version are two useful features. These two features need to be transformed to provide the expected prediction behavior. We did a number of experiments based on different combinations of time and version schemas.

3.3.1. Time

We tried two forms of time. One is epoch time, the other is using month and day separately without year. The reason we adopted the second format in the later experiments is because we thought year is not a useful information since year will not be repeated. Even so, we did a couple of experiments comparing the two different schemas, and the latter schema showed better results on most of the experiments.

3.3.2. Version

Intuitively, version is also a useful information from NVD. We tried to calculate the difference between two adjacent instances' versions to get the versiondiff as a predictive feature. The rational of doing this is because we want to use the trend of the version with time to estimate the future situation. Two of the versiondiff schemas are introduced in our approach. First is calculating the versiondiff based on version counters (purely calculate the difference of versions based on their ranks). The second schema is calculating the versiondiff by radix (assigning higher weight to relative major version while doing the diff calculation).

However, we realized that versiondiff did not work well on our case because all of the new vulnerabilities affecting current version will also affect previous versions. Therefore, most values of versiondiff are zero (since if the version has already in existence, then the diff of versions must be zero). In order to mitigate the limitation, we created another predictive feature for our later experiments. The new feature is occurrence number of certain version of each software. More details will be introduced in Section 5.

3.4. Prediction Model

We collected all useful configuration information of the target machine. Then based on the software it has installed, we can judge how potential risk could come from these applications, combined with the evaluation of known vulnerabilities. Eventually it can output a quantitative value showing the risk-level of this system (e.g., Mean Time to Next Vulnerability (MTTNV) along with Common Vulnerability Scoring System (CVSS) Metrics (see Figure 6).

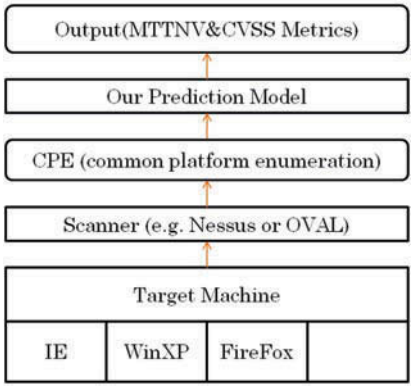

FIGURE 6 Flow chart.

- **Predictive data**
  Month, Day, Versiondiff, TTPV (Time to Previous Vulnerability), CVSS Metrics (indicate the properties of the predicted vulnerabilities).
- **Predicted data**
- TTNV (Time to Next Vulnerability) implies the risk level of zero-day vulnerabilities.

3.5. Machine Learning Function Type

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 can wait until the occurrence of 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.

3.5.1. Regression

There are several regression functions: Linear regression, Least median square, multilayer perceptron, radial basis function (RBF) network, sequential minimal optimization (SMO) regression, and Gaussian processes.

3.5.2. Classification

There are also several classification functions: Logistic, least median square, multilayer perceptron, RBF network, SMO, and simple logistic.

4. EXPERIMENTAL RESULTS

4.1. Definitions

Some academic terms will be used in the following description. (We actually only used correlation coefficient as the indicator of fitness because it is the most straightforward. Other indicators are also included in a standard Waikato Environment for Knowledge Analysis [WEKA; Bouckaert et al., 2010] output). They are used to explain the accuracy of the model. In order to make it easier to understand, we put some of their definitions here:

Mean Absolute Error

Mean absolute error (MAE) is a quantity used to measure how close forecasts or predictions are to the eventual outcomes. The MAE is given by:

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}\left|f_i - y_i\right| = \frac{1}{n}\sum_{i=1}^{n}\left|e_i\right|$$

Root Mean Squared Error

The mean square error (MSE) of an estimator is one of many ways to quantify the difference between an estimator and the true value of the quantity being estimated.

$$\text{RMSE}\left(\theta_1, \theta_2\right) = \sqrt{MSE\left(\theta_1, \theta_2\right)} = \sqrt{E\left(\left(\theta_1 - \theta_2\right)^2\right)}$$

$$= \sqrt{\frac{\sum_{i=1}^{n}\left(x_{1,i} - x_{2,i}\right)^2}{n}}$$

Relative Absolute Error

Relative absolute error (RAS) is defined as the summation of the difference between predictive value and given value for the sample case i to that divide it by the summation of the difference between the given value and average of the given value (for information on RAS, see http://www.tutorvista.com/math/relative-absolute-error).

$$E_i = \frac{\sum_{i=1}^{n}\left|P_{(ij)} - T_j\right|}{\sum_{i=1}^{n}\left|T_j - \bar{T}\right|}$$

Root Relative Squared Error

The root relative squared error (RRSE) is relative to what it would have been if a simple predictor had been used. More specifically, this simple predictor is just the average 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 square root of the relative squared error, one reduces the error to the same dimensions as the quantity being predicted (see http://www.gepsoft.com).

$$E_i = \sqrt{\frac{\sum_{i=1}^{n}\left(P_{(ij)} - T_j\right)^2}{\sum_{i=1}^{n}\left(T_j - \bar{T}\right)^2}}$$

Correlation Coefficient

The correlation coefficient—a concept from statistics—is a measure of how well trends in the predicted values follow trends in past actual values. It is a measure of how well the predicted values from a forecast model "fit" with the real-life data.

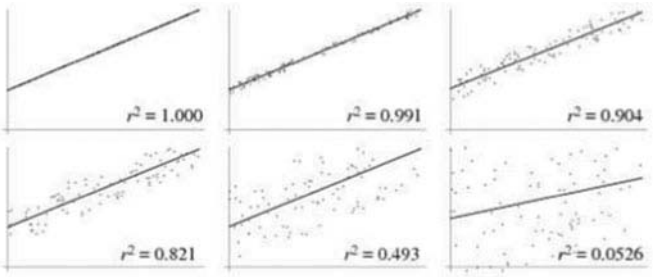

FIGURE 7 **Different similarity levels from different correlation coefficient.**

The correlation coefficient is a number between 0 and 1. If there is no relationship between the predicted values and the actual values, the correlation coefficient is 0 or very low (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. Thus the higher the correlation coefficient the better. Figure 7 can demonstrate the correlation coefficient ("What is," http://www.forecasts.org).

4.2. Experiments

For all of our experiments, we did them on a cluster of computing nodes from Beocat, which is a private cloud within Kansas State University Computing and Information Science department (see https://www.cis.ksu.edu/beocat/about). Because most of the experiments will take a lot of time on a single host, for all of the experiments we use single core and 4G RAM. Weka (Waikato Environment for Knowledge Analysis) (Bouckhaert et al., 2010) is a data-mining suite for all experiments.

We did a couple rounds of experiments including using different versiondiff schemas, different time schemas, including CVSS metrics or not. Overall, for different models, different feature combinations will output optimal results (having highest correlation coefficient value). Hence, we believe it is hard to build a single model for all of the software. Also, if we build a single model for all of the software, then it will turn out a scalability issue. It won't finish within at least one week. Because we need to have both vendor name and software name as predictive features, and both of them have a lot of possibilities, which means we need to have two large nominal types as predictive features, which will too long time to be finished.

In order to resolve this scalability issue, we tried to build a model for top six vendors (as mentioned), but the problem still could not be resolved. It still needs more than four days; (it could not be much longer than four days; we set up a time limit for our experiment, and it will terminate within 100 hours no matter it is finished or not) to be finished. Therefore, we tried to build models for each vendor. We believe it is hard to find some relation between different vendors. And since there are so many vendors there, we could not do all of the experiments on each of them. We put more effort on several vendors (e.g., Linux, Microsoft, Mozilla and Google) to build our models. For some (Linux, Microsoft and Mozilla), we tried to build software-based models as well. For the remaining vendors (Apple, Sun and Cisco), we only did vendor-based experiments.

4.3. Results

We tried to build vendor-based models initially, then models for single software. Most of these single software are web browsers since they are more vulnerable than other applications, so they have enough data for our experiments.

Linux. We tried to use two versiondiff schemas including counter-based versiondiff and radix-based versiondiff to compare which one is more appropriate for our model construction. Also, we compare two different time schemas (epoch time and using month and day separately). Besides, we binned the class feature (TTNV) as a comparison group since we noticed there are clusters in the distribution of TTNV.

Counter versiondiff. For this versiondiff schema, we did not differentiate between minor versions and major versions. For example, if one software has three versions: 1.1, 1.2, 2.0, then the 1.1, 1.2, and 2.0 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 versiondiff between 1.2 and 2.0.

Epoch time. Epoch time is very popularly used time schema by many people within computer realm. It represents how many seconds elapsed since the midnight of 1970-1-1, which is the year of Linux's birth. We did some experiments by using this time schema. But the results are not very ideal (all correlation coefficients are below 0.5). Table 1 shows the comparing results between epoch time and month day schema.

Month and day. We noticed that year is useless for our prediction work because year will not show again in the future. We tried another time schema by only including day and month. The results from this schema are a little bit better (correlation coefficient is higher. Specifically, half of the training correlation coefficient is higher than 0.5 for month and day set but none of such values is higher than 0.5 for the epoch time set). Then for all of the later experiments we only used this schema.

TABLE 1 Correlation Coefficient for Linux Vulnerabilities Using Two Formats of Time

	epoch time		Month and day	
Functions	Train	Test	Train	Test
Linear Regression	0.3104	0.1741	0.6167	−0.0242
Least Mean Square	0.1002	0.1154	0.1718	0.1768
Multilayer Perceptron	0.2943	0.1995	0.584	−0.015
RBF	0.2428	0	0.1347	0.181
SMOReg	0.2991	0.2186	0.2838	0.0347
Gaussian Processes	0.3768	−0.0201	0.8168	0.0791

Radix-based versiondiff. We realized the difference between major versions are different from the difference between minor versions. Since a change on major versions suggest a higher degree of difference on the functionality of software. So while calculating versiondiff, we need to assign a higher weight to relatively major version and lower weight to relatively minor version. For example, if a software has three versions 1.0, 1.1, 2.0, then we assign a weight of 10 to the major version and a weight of 1 to the minor version. Then the versiondiff between 1.1 and 1.0 is 1, while the versiondiff between 2.0 and 1.1 will be different from the previous one. It'll be9 ($2 \times 10 + 0 \times 1 - 1 \times 10 - 1 \times 1$). The results from this schema is a little bit better. However, since the most versiondiff values are zero, we have to abandon this schema eventually. Table 2 shows the comparing results of the two different versiondiff schemas.

Binning. Since we found that the class feature (TTNV) of Linux has obvious clusters. We then divided the class feature into two categories. One is more than 10 days the other is no more than 10 days. After this, the results became better in terms of the corrected classified rates. However, the false positive rates are still high (above 0.7). We also tried to use Gaussian (RBF) kernel for SMO function, it showed better results in terms of the correctly classified rate. However, it still has a false positive rate about 0.74, which is far from acceptable. Table 3 shows the correctly classified rates for these experiments.

Besides the binned on TTNV, we did binning on versions of Linux kernel. We round all of the sub-versions

TABLE 2 Correlation Coefficient for Linux Vulnerabilities Using Two Formats of Versiondiff

	Version Counter		Radix based	
Functions	Train	Test	Train	Test
Linear Regression	0.6167	−0.0242	0.6113	0.0414
Least Mean Square	0.1718	0.1768	0.4977	−0.0223
Multilayer Perceptron	0.584	−0.015	0.6162	0.1922
RBF	0.1347	0.181	0.23	0.0394
SMOReg	0.2838	0.0347	0.2861	0.034
Gaussian Processes	0.8168	0.0791	0.6341	0.1435

TABLE 3 Correctly Classified Rate for Linux Vulnerabilities Using Binned TTNV

Functions	Correctly classified	
	Train	Test
Simple Logistic	97.9304%	59.6769%
Logistic Regression	98.7675%	59.7943%
Multilayer Perceptron	97.7754%	59.6456%
RBF	92.9075%	61.0068%
SMO	97.9381%	66.9757%
SMO (RBF kernel)	98.1009%	78.8117%

to its third significant major version, for example, $Bin\,(2.6.3.1) = 2.6.3$. The reason we binned the first three most significant versions is because more than half instances (31834/56925) have a version longer than 3. And only 1% (665/56925) instances versions are longer than 4. Also, the difference on the third subversion will be regarded as a huge dissimilarity for Linux kernel. We didn't do the same experiments on Microsoft because the versions of Microsoft products are naturally discrete (all of them have versions less than 20). Table 4 indicates the comparisons between binned versions schema or not. The results are not good enough since many of versiondiffs are zero.

CVSS metrics. For all of the experiments, we created another comparing group by adding CVSS metrics as predictive features. However, we could not see a lot of differences by adding CVSS metrics or not. Hence, for Linux kernel, CVSS metrics cannot tell us much.

Adobe. Since we have already realized the limitation of versiondiff schema, for Adobe instances we use occurrence number of certain version of a software instead of using versiondiff. Also, we used month and day format instead of epoch time. The results for testing data are not good

TABLE 4 Correlation Coefficient for Linux Vulnerabilities Using Binned Versions or not

	Non-bin versions		Binned versions	
Functions	Train	Test	Train	Test
Linear Regression	0.6113	0.0414	0.6111	0.0471
Least Mean Square	0.4977	−0.0223	0.5149	0.0103
Multilayer Perceptron	0.6162	0.1922	0.615	0.0334
RBF	0.23	0.0394	0.0077	−0.0063
SMOReg	0.2861	0.034	0.285	0.0301
Gaussian Processes	0.6341	0.1435	0.6204	0.1369

TABLE 5 Correlation Coefficient for Adobe Vulnerabilities Using CVSS Metrics or not

	Include cvss		Without CVSS	
Functions	Train	Test	Train	Test
Linear Regression	0.7932	0.0625	0.7031	−0.1058
Least Mean Square	0.6097	−0.351	0.6124	−0.3063
Multilayer Perceptron	0.9268	0.174	0.8759	−0.0469
RBF	0.1817	0.3845	0.1165	0.2233
SMOReg	0.7178	0.1299	0.6191	0.0979
Gaussian Processes	0.9111	0.0322	0.8563	−0.1317

on adobe. All of the correlation coefficient values are low (less than 0.4). Since there are not any obvious clusters in the TTNV distribution, we did not bin the class feature, which means we only tried regression functions. Also, we noticed two applications of Adobe, Acrobat Reader and Flash Player, are much more vulnerable than others. We wanted to build models for these two software applications, but each of them has a number of instances less than 2,000, which is far from enough for our task. Table 5 indicates the difference while adding CVSS metrics to Adobe instances.

Microsoft. We analyzed the instances and found that more than half of the instances did not have version information. And for most of these cases, the software used was Windows. Besides Windows, more than 70% of instances have version information, so we used two different occurrence features for these different kinds of instances. For Windows instances, we only used the occurrence of each software as a predictive feature. For the rest of the instances, we used the occurrence of each version of this software

TABLE 6 Correlation Coefficient for Win and Non-Win Instances

	Win Instances		Non-win Instances	
Functions	Train	Test	Train	Test
Linear Regression	0.4609	0.1535	0.5561	0.0323
Least Mean Square	0.227	0.3041	0.2396	0.1706
Multilayer Perceptron	0.7473	0.0535	0.5866	0.0965
RBF	0.1644	0.1794	0.1302	−0.2028
SMOReg	0.378	0.0998	0.4013	−0.0467
Gaussian Processes	0.7032	−0.0344	0.7313	−0.0567

as a predictive feature. Also, there is no obvious clusters for either Windows or non-Windows instances TTNV, so we just tried regression functions here. Again, for time schema we used month and day separately. We did not use versiondiff as a predictive feature since we realized most of the versiondiff's value are 0 and because all of the vulnerabilities affecting current versions will affect all previous versions as well. This may not be the true case, but from our observation, many vendors (e.g., Microsoft or Adobe) claim so. For the same reason, we abandoned versiondiff schema for the rest of experiments. Instead, we used the aforementioned feature (occurrence of certain software or occurrence of certain version of each software), but the results were not very good (see Table 6). All of the correlation coefficients were less than 0.4. We further tried to build models for non-Windows individual applications. For example, we extracted IE instances and tried to build models for it. When CVSS metrics were included, the results (correlation coefficient is about 0.7) looked better than without CVSS metrics (correlation coefficient is about 0.3). We also tried to do experiments on office. However, there were only 300 instances for office, and the after Office-related instances were all about individual software such as Word, Powerpoint, Excel and Access. Each had less than 300 instances. So we gave up building models for Microsoft office. Table 7 shows the difference between adding CVSS metrics to IE instances and without CVSS metrics.

Mozilla. We only tried to build models for Firefox. Then results were relatively good(0.7) whether or not CVSS metrics were included. However, one concern was that the number of instances were small (fewer than 5000). Table 8 shows the comparing results of firefox including CVSS metrics or not.

TABLE 7 Correlation Coefficient for IE Vulnerabilities Using CVSS Metrics or not

	Include cvss		Without CVSS	
Functions	Train	Test	Train	Test
Linear Regression	0.8023	0.6717	0.7018	0.3892
Least Mean Square	0.6054	0.6968	0.4044	0.0473
Multilayer Perceptron	0.9929	0.6366	0.9518	0.0933
RBF	0.1381	0.0118	0.151	−0.1116
SMOReg	0.7332	0.5876	0.5673	0.4813
Gaussian Processes	0.9803	0.6048	0.9352	0.0851

TABLE 8 Correctly Classified Rate for Firefox Vulnerabilities Using CVSS Metrics or Not

	Include CVSS		Without CVSS	
Functions	Train	Test	Train	Test
SimpleLogistic	97.5%	71.4%	97.5%	71.4%
Logistic Regression	97.5%	70%	97.8%	70.5%
Multilayer Perceptron	99.5%	68.4%	99.4%	68.3%
RBF	94.3%	71.9%	93.9%	67.1%
SMO	97.9%	55.3%	97.4%	55.3%

Google (Chrome). We also realized that Google has become a vulnerable vendor (in terms of instances number) in the last three months (March-May 2010). We found most Google instances originate from chrome. We thought of building models for Google Chrome initially. However, we realized more than half of the instances of Chrome appeared within two months (April–May 2010). Therefore, we think it will be hard to predict something within such a short timeline.

Apple, Sun and Cisco. We merely tried to build vendor-based models for these three manufacturers for the time reason. The results were not very good, similar to the results of the Adobe experiments. Table 9 indicates the results of comparing two different time schemas with Apple instances. (For scalability reason, Cisco and Sun instances can only finish Linear Regression experiments and the results are similar to Apple. The rest of the experiments could not be finished within two days.)

4.4. Parameter Tuning

Based on each group of test, we tuned the paramters? and g for SVM (support vector machine) on all of the

TABLE 9 Correlation Coefficient for Apple Vulnerabilities Using Two Formats of Time

	Include CVSS		Without CVSS	
Functions	Train	Test	Train	Test
Linear Regression	0.6952	0.4789	0.763	0.4851
Least Mean Square	0	0	0	0
Multilayer Perceptron	0.8027	0.6007	0.9162	0.6994
RBF	0.0255	0.0096	0.0308	0.0718
SMOReg	0.6421	0.501	N/A	N/A
Gaussian Processes	N/A	N/A	N/A	N/A

regression models. We picked serveral values for each paramter (g is 0, 0.05, 0.1, 0.2, 0.3, 0.5, 1.0, 2.0, 5.0 and? is 0.5, 1.0, 2.0, 3.0, 5.0, 7.0, 10, 15, 20) and did a brute force search in order to find out which one could provide us with the optimal value (in terms of correlation coefficient, root mean square error (RMSE), and root relative square error (RRSE)). We picked up the optimal parameters at the first round of validation. In order to make sure the optimal parameters really work well, we did a second round test (the size of the dataset for the validation and test are almost the same but the test dataset is later than the validation dataset). Tables 10 and 11 indicate the results in terms of correlation coefficient, RRSE, and RMSE. Including both validation and test parts.

4.5. Summary

The experiments outlined in this article indicate that it is hard to build one prediction model for a single vendor, since the trend of different software of each vendor could be quite different. Also, the data we can use are limited. For example, we could not have version information for most of Microsoft instances (most of them are Windows instances). Some results look promising (e.g., the models we built for Firefox and IE). However, this can also strengthen our belief that the trends of different software are far from each other. We used two different predictive sets to construct the best models for two different software. Therefore, in order to capture the trends of different popular applications, we need more accurate or matured data. Also, if we could build models for similar software, then we could build several models and each of which could include a set of popular applications.

5. RELATED WORKS

A number of related works are available, all mentioning the threat from zero-day vulnerabilities.

Alhazmi and Malaiya (2006) did a number of experiments on building models for predicting the number of vulnerabilities that will appear in the future. They targeted operating systems instead of building a whole model for all of the applications. The Alhazmi-Malaiya Logistic model looks good. However, it can only estimate vulnerabilities for operating systems rather than software applications, which is important for quantitative risk assessment. Also, the number of vulnerabilities is not enough for quantifying the risk level of software because there are differences between different vulnerabilities and different software, which is considered in our experiments.

Ozment (2007) examined the vulnerability discovery models (pointed out by Alhazmi & Malaiya, 2006) and

TABLE 10 Parameter Tuning for Correlation Coefficient

	Parameters		Validation			Test		
Test Group	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
Internet Explorer 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
Internet Explorer	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 Bin	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 11 Parameter Tuning for RMSE and RRSE

	Parameters		Validation			Test		
Test Group	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
Internet Explorer CVSS	2.0	1.0	8.4729	74.8645%	0.4466	11.4604	91.1018%	−0.3329
Non-win	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
Internet Explorer	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 Bin	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

realized there are some limitations making these models unapplicable. For example, there is not enough information included in government-supported vulnerability databases (e.g., National Vulnerability Database). This illustrated why our current results are not good enough from another angle.

Ingols et al. (2009) tried to model network attacks and countermeasures through an attack graph. They pointed out the dangers from zero-day attacks and also mentioned the importance of modeling zero-day attacks. However, they pointed out this without doing any experiments. But this inspired us to conduct our data-mining work.

McQueen et al. (2009) realized the importance of unknown security holes. They tried to build logarithm-like models, which could tell us the approximate number of zero-day vulnerabilities on each given day. This number can somehow tell us the overall risk level from zero-day vulnerabilities. However, the situation could be different for different applications, and not every vulnerability has the same level of risk. We considered these differences in our experiments.

Nguyen and Tran (2010) and Massacci and Nguyen (2014) compared several existing vulnerability databases by what features of vulnerability are included in each. They mentioned that many important features are not included in ALL of the database, for example, discovery date, for which it is hard to find precise value. Even though certain databases (OSVDB as we realized) claim they include the features, most of the entries are blank. Source code-related features also may be missing, since commercialized products are not open source, so these could not be included in NVD. For their Firefox vulnerability database, the authors used some textual retrieval technique to take some key words from CVS commiter's check-in log and to get several other features by cross reference through CVE IDs. They realized by using one or two different data sources for the same experiment, the results could be quite different due to the high degree of inconsistency of the data available. So they 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 priortize the security level of software component for Firefox.

6. CONCLUSIONS

This research mainly focuses on the prediction of time to next vulnerability, which is in days. This will potentially provide us with a value associated with unknown risks. Also, we may apply the prediction model to the MulVAL attack-graph analyzer (Xinming et al., 2005; Homer et al., 2013; Huang et al., 2011; Zhang et al., 2011), which is a quantitative risk assessment framework based on known vulnerabilities. Ideally the whole work will provide a comprehensive risk assessment solution along with other environments like the Cloud (Zhang, 2014; Zhang, Zhang, & Ou, 2014; DeLoach, Ou, Zhuang, & Zhang, 2014; Zhang, 2012; Zhuang et al., 2012) and software dependency (Zhang et al., 2015). However, until now we could not find any model that could handle all software. Besides, the data source has a number of limitations (e.g., all later vulnerabilities will affect all previous versions which makes versiondff unapplicable) that restrict its use in a data-mining-based approach. Moreover, the quality of the data is inconsistent. Even though there are CPEs for most CVE entries, many of the attributes are

missing. (e.g., Windows instances do not have any version information). Also, even though some information (e.g., version) is given, we could not use it because of the aforementioned reason. So the data left for our data-mining work are much less than we originally expected. Therefore, we could not build any accurate model from NVD in its current form.

7. FUTURE WORK

Through this data-mining work on NVD, we realized there are limitations in the NVD, which limits its use as an effective source. We believe that the number of zero-day vulnerability (of each software) could be another informative indicator for quantifying the risk-level of zero-day vulnerabilities. This will use the vulnerability life span (since in order to have the number of current zero-day vulnerabilities, we need to count the number of zero-day vulnerabilities between current time point and another time point when all current vulnerabilities have been discovered. Since the number of current zero-day vulnerabilities may not be enough for quantifying the risk-level of zero-day vulnerabilities, we also need to consider the severity of each vulnerability of each application. We can design metrics to indicate the severity level of these vulnerabilities. For example, we can consider calculating the average CVSS score as the indicator. These possible future works may be simply done through statistic analysis without using data-mining techniques.

REFERENCES

Alhazmi, O.H., & Malaiya, Y.K. (2006). Prediction capabilities of vulnerability discovery models. In *Annual Reliability and Maintainability Symposium, 2006. RAMS '06.* Annual Reliability and Maintainability Symposium, Newport Beach, CA, IEEE.

Bouckaert, R.R., Frank, E., Hall, M., Kirkby, R., Reutemann, P., Seewald, A., & Scuse, D. (2010). *WEKA manual for version 3.7*. The University of Waikato, Hamilton, New Zealand.

Buttner, A., & Ziring, N. (2009). *Common platform enumeration (CPE) C specification*. The MITRE Corporation AND National Security Agency. McLean, VA: Mitre Corp.

DeLoach, S., Ou, X., Zhuang, R., & Zhang, S. (2014). Model-driven, moving-target defense for enterprise network security. In U. Amann, N. Bencomo, G. Blair, B.H.C. Cheng, & R. France (Eds.), *State-of-the-art survey volume on models @run.time* (pp. 137–161). New York: Springer.

Gopalakrishna, R., Spafford, E.H., & Vitek, J. (2005). *Vulnerability likelihood: A probabilistic approach to software assurance*. Purdue University. Amsterdam: IOS Press.

Homer, J., Zhang, S., Ou, X., Schmidt, D., Du, Y., Raj Rajagopalan, S., & Singhal, A. (2013). Aggregating vulnerability metrics in enterprise networks using attack graphs. *Journal of Computer Security (JCS)*, *21* (4).

Huang, H., Zhang, S., Ou, X., Prakash, A., & Sakallah, K. (2011). Distilling critical attack graph surface iteratively through minimum-cost SAT solving. (best student paper). In *Proceedings of Annual Computer Security Applications Conference (ACSAC 11)*, Orlando, FL.

Ingols, K., Chu, M., Lippmann, R., Webster, S., & Boyer, S. (2009). Modeling modern network attacks and countermeasures using attack Graphs. In AC-SAC. Paper presented at Annual Computer Security Applications Conference (ACSAC) 2009, Honolulu, Hawaii, USA.

Massacci, F., & Nguyen, V.H. (2014). Which is the right source for vulnerability studies? An empirical analysis on Mozilla Firefox. In MetriSec. Paper presented at MetriSec '10, September 15, 2010, Bolzano-Bozen, Italy.

May, P., Ehrlich, H.C., & Steinke, T. (2006). ZIB structure prediction pipeline: Composing a complex biological workflow through web services. In W.E. Nagel, W.V. Walter, & W. Lehner, (Eds.), *Euro-Par 2006, LNCS*, *4128*, 1148–1158.

McQueen, M.A., McQueen, T.A., Boyer, W.F., & Chaffin, M.R. (2009). Empirical estimates and observations of 0 day vulnerabilities. Paper presented at Hawaii International Conference on System Sciences, HICSS 09, January 5–8, 2009, Big Island, Hawaii, USA.

Mell, P., Scarfone, K., & Romanosky, S. (2007). *A Complete guide to the common vulnerability scoring system, version 2.0*. National Institute of Standards and Technology AND Carnegie Mellon University.

Nguyen, V.H., & Tran, L.M.S. (2010). Predicting vulnerable software components with dependency graphs. In MetriSec. Paper presented at MetriSec '10, September 15, 2010, Bolzano-Bozen, Italy.

Ozment, A. (2007). Vulnerability discovery & software security. University of Cambridge: Cambridge University Press.

Smith, T.F., & Waterman, M.S. (1981). Identification of common molecular subsequences. *Journal of Molecular Biology*, *147*, 195–197.

Ou, X., Govindavajhala, S., & Appel, A.W. (2005, August). MulVAL: A Logic-based Network Security Analyzer. Paper presented at 14th USENIX Security Symposium, Baltimore, Maryland, USA.

Zhang, Su, Caragea, D., & Ou, X. (2011). An empirical study on using the national vulnerability database to predict software vulnerabilities. In *Proceedings of the 22nd International Conference on Database and Expert Systems Applications (DEXA 11), Vol. Part I,* Toulouse, France.

Zhang, S., Ou, X., & Homer, J. (2011). Effective network vulnerability assessment through model abstraction. In *Proceedings of the Eighth SIG SID AR Conference on Detection of Intrusions and Malware & Vulnerability Assessment (DIMVA 11)*, Amsterdam, The Netherlands.

Zhang, S., Ou, X., Singhal, A., & Homer, J. (2011). An empirical study of a vulnerability metric aggregation method. In *Proceedings of the 2011 International Conference on Security and Management (SAM 11)*, Las Vegas, NV.

Zhang, S. (2012). Deep-diving into an easily-overlooked threat: Inter-VM attacks. (Technical Report). Manhattan, Kansas: Kansas State University.

Zhang, S. (2014). *Quantitative risk assessment under multi-context environment* (Doctoral dissertation). Kansas State University, Manhattan, Kansas.

Zhang, S., Zhang, X., & Ou, X. (2014). After we knew it: Empirical study and modeling of cost-effectiveness of exploiting prevalent known vulnerabilities across IaaS cloud. In *Proceedings of the 9th ACM Symposium on Information*, Computer and Communications Security (ASIACCS 14), Kyoto, Japan.

Zhang, S., Zhang, X., Ou, X., Chen, L., Edwards, N., & Jin, J. (2015). Assessing attack surface with component-based package dependency. In *Proceedings of 9th International Conference on Network and System Security(NSS 15)*, New York: Springer.

Zhuang, R., Zhang, S., DeLoach, S.A., Ou, X., & Singhal, A. (2012). Simulation-based approaches to studying effectiveness of moving-target network defense. In *Proceedings of National Symposium on Moving Target Research*, Annapolis, MD.

Zhuang, R., Zhang, S., Bardas, A., DeLoach, S.A., Ou, X., & Singhal, A. (2013). Investigating the application of moving target defenses to network security. In *Proceedings of the 1st International Symposium on Resilient Cyber Systems (ISRCS)*, San Francisco, CA.

BIOGRAPHIES

Dr. Su Zhang is a senior software engineer at Symantec Corporation. He earned his Ph.D in Computer Science from Kansas State University. He has extensive engineering and research experience, especially in cyber security domain. He has held positions at VMWare Huawei and Apigee, and worked on various security engineering and research projects. He also was heavily involved in open source project development (as a major contributor of a widely used open source network analyzer-MulVAL, which can be downloaded at http://www.arguslab.org/mulval.html)

Dr. Xinming Ou is an associate professor at University of South Florida. He was at Kansas State University between 2006 and 2015. He previously was a postdoctoral research associate at Purdue University's Center for Education and Research in Information Assurance and Security from September 2005 to May 2006, and a research associate at Idaho National Laboratory from May 2006 to August 2006. He received his doctorate from Princeton University in 2005 and earned a bachelor's and a master's degree in computer science from Tsinghua University in Beijing, China.

Dr. Doina Caragea is an associate professor at Kansas State University. Her research interests include artificial intelligence, machine learning, and data mining, with applications to bioinformatics, social network analysis, and scientometrics, among others. She received her Ph.D. in Computer Science from Iowa State University in August 2004 and was honored with the Iowa State University Research Excellence Award for her work. She has published more than 80 refereed conference and journal articles and was co-editor for *Computational Methodologies in Gene Regulatory Networks*, published by IGI Publishers. She is teaching machine learning, data mining, and bioinformatics courses.