Predicting Unknown Vulnerabilities using Software Metrics and Maturity Models

Patrick Kamongi, Krishna Kavi, Mahadevan Gomathisankaran

Department of Computer Science and Engineering
University of North Texas
Denton, TX 76203, USA
Emails: patrickkamongi@my.unt.edu, kavi@cse.unt.edu, mgomathi@unt.edu

***Abstract*—We face an increasing reliance on software-based services, applications, platforms, and infrastructures to accomplish daily activities. It is possible to introduce vulnerabilities during any software life cycle and these vulnerabilities could lead to security attacks. It is known that as the software complexity increases, discovering a new security vulnerability introduced by subsequent updates and code changes becomes difficult. This can be seen from the rate of new vulnerabilities discovered after a software release. IT Products' vulnerabilities sometimes remain undiscovered for many years. In this paper, we report our study of IT products' source codes using software maturity models and the history of vulnerabilities discovered. We use this data to develop a model to predict the number of security vulnerabilities contained in a product, including undiscovered vulnerabilities. Our proposed approach can be used to explore proactive strategies for mitigating the risks due to zero-day vulnerabilities.**

Keywords–Vulnerabilities; Metrics; Models.

I. Introduction

Any software product that is in production goes through a series of changes throughout its lifecycle as a result of feature changes or bug fixes among other factors. As any given software product matures, it has been shown that it is vulnerability-prone and that its security vulnerabilities do get discovered throughout its maturity. For the last decade or so, we have seen a rising trend of security vulnerabilities in software products being disclosed on a regular basis [1]. This observed trend calls for an increased security awareness and demands new approaches to stay ahead of this alarming fact.

The type of security vulnerabilities that are discovered and leveraged by malicious actors before the relevant software provider becomes aware of and fixes them, are known as zero-day (0day) vulnerabilities. The worrisome nature of 0day vulnerabilities is due to the endless number of approaches that a malicious actor might employ to abuse a given software product. The security community and software product vendors sponsor bug bounty initiatives in an attempt to stay ahead of the large number of vulnerabilities hidden in software products currently in use. A portion of these hidden vulnerabilities in software products are being discovered using assessment techniques targeting some aspects of the software's design, implementation and use; such as static code analysis and dynamic binary analysis but still undisclosed vulnerabilities remain, and this serves as the motivation for our research.

An estimation of the potential number of undetected or unreported security vulnerabilities is useful because it may lead to proactive strategies for protecting IT assets. In this work, we want to address the following types of questions:

- To what extent do software complexity metrics correlate with the number of vulnerabilities disclosed for any given software product instance?
- Can we extrapolate a list of specific software metrics that shows a high correlation with regard to the above question?
- Can we predict the number of undisclosed vulnerabilities for any given software product?

Previous research has attempted to predict software error (or bug) incidences using software change history [2]. Software metrics have also been used to predict vulnerability prone codes in any given software [3]. Various techniques have been used to study the trend of vulnerabilities in software products [4] [5]. In this study, we explore the correlation between software change history and maturity with the number of vulnerabilities each software release may contain and subsequently exposed.

The main contributions of our work are:

- Sweep: a toolkit that automates software complexity metrics generation and analysis.
- A methodology for using the Sweep toolkit to automatically generate a dataset for any given software product. The produced dataset contains information on all software releases for the given software product along with the relevant software metrics and number of reported vulnerabilities for each release in a timeline fashion.
- A proof of concept predictive web service endpoint, based on a machine learning regression classifier trained on the dataset produced by the Sweep toolkit. This endpoint is leveraged in an automated fashion to predict the number of unknown vulnerabilities for any given software product instance.

The rest of this paper is organized as follows. In Section II, we present our proposed methodology towards building a predictive model for estimating the number of unknown vulnerabilities. In Section III, we present our experimental study and its evaluation. Section IV contains related works pertinent to this study. Finally, Section V contains conclusions of our work thus far, along with possible extensions and future direction of our research.

II. Our Predictive Methodology

In this section, we present our novel model for predicting the number of unknown vulnerabilities for any given IT Product. We start by presenting our data collection approach in Section II-A, then the details of how we designed and validated our prediction model in Section II-B.

A. Data collections

For this study, we have devised a generic and automated data collection approach for any software product that has its various release source codes available and written in known programming languages. The data of our interest is based on each studied software product's release complexity metrics, and a timeline trend of the number of disclosed vulnerabilities.

For each IT Product (software product), we start by collecting details about the product's releases (with an identifying name), the number of releases and the number of vulnerabilities already reported. We also analyze the source code of each released version of the IT Product and then compute various software metrics for the given source codes. The collected data is stored in a dataset file as comma-separated values (CSV). The following describes the process for collecting data for any IT product.

1) Select the IT Product to analyze.
2) Download all of this IT Product's released versions and source code for these versions.
3) For each release, represent it using a Common Platform Enumeration (CPE) [6] format as its unique identifier for cataloging the product version and its data into our datasets.
4) For each of the above releases, use our lookup index to match and find all reported or known vulnerabilities and store this information using a timeline trend.
5) Perform software source code analysis for each released version, and store all computed software metrics data [7].

Since in most cases the size of the source code for each release is very large, we leverage the capabilities of the *Understand 4.0* [8] tool for static code analysis and software metrics generation. To alleviate the complexity involved in collecting the data of our interest for any given software product manually, we designed and implemented the Sweep-toolkit to automate data collection and avoid any human error during this process.

Sweep-toolkit is designed as a lightweight framework that implements various plugins to orchestrate the data collection for this study. The currently implemented plugins offer these functionalities:

- Use of the *Understand* [8] command line tool (*und*) to automate the metrics generation for each IT Product's release source codes by automatically generating and executing relevant batch files.
- An automated approach to analyze and summarize the generated metrics for each IT Product release.
- A workflow engine to build and ensure that each IT Product release is represented in a CPE format.
- A Lookup index of the known vulnerabilities [1] and affected IT Products in CPE format.
- An orchestration application to automate and march in all these above functionalities towards data collection for any given IT Product and returns a dataset file which is set to be used in the predictive model experiment (Section II-B).

As described above, by passing relevant details of any given IT Product to the Sweep-toolkit, a dataset is generated as a result of the toolkit execution. The generated dataset is made of a set of 124 features that would represent collected data on each of the analyzed IT Product instances. In Section III, we illustrate a case scenario used for this study as a proof of concept of our proposed predictive model.

Sweep-toolkit [9] is developed for a Linux environment, with its plugins implemented in Python 3.x, Java and Bash scripts (approximately 2K lines of code).

B. Predictive model

In this section, we introduce our model for predicting vulnerabilities and show how we validated our model.

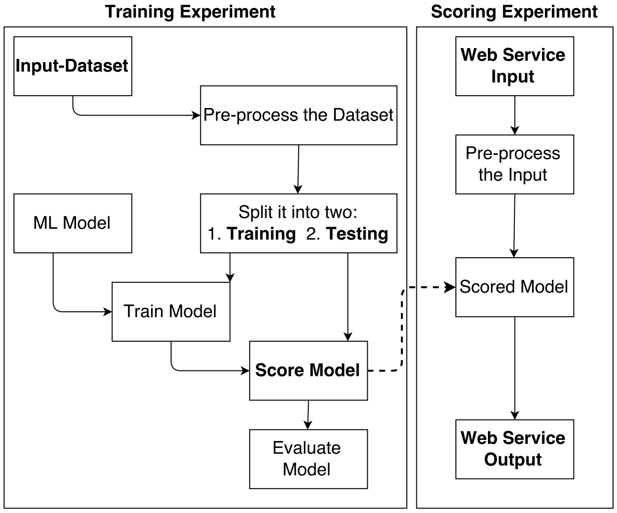

Figure 1. Predictive Model - Framework

In an attempt to address our posed research question of whether we can predict the number of undisclosed vulnerabilities for any given software product, we base our solution on the data that can be collected for this software product using our Sweep-toolkit as discussed in the previous Section II-A. Once a dataset is generated for the software product of interest, we proposed to design and train a machine learning regression classifier to build a model that should be able to predict the number of vulnerabilities for any other release of this software product.

For any predictive experiment, the more data you have the more accurate the trained model becomes. With this in mind, among all of the data collected around the 124 features found in any given dataset, we leverage a feature scoring method to identify which set of features correlate well with the number of disclosed vulnerabilities for each of the studied software product releases. Then, we start by exploring different machine learning regression techniques to find one that best fits our collected data. For this study, we leverage Microsoft Azure – Machine Learning Studio [10] to design our predictive model.

Azure Machine Learning Studio provides many easy to use building blocks for designing a predictive solution.

We build our predictive model along these easy to follow steps, which are also illustrated in Fig. 1:

- Create a machine learning workspace within Azure ML Studio for each relevant IT Product's dataset.
- Upload the data generated by Sweep-toolkit for the IT product into Azure Datasets, and design predictive models within Azure ML Studio.
- Train and Evaluate the models and identify the best model.
- Score and publish the best model as a web service.

Some specific aspects that drive our predictive model are our choices for the regression classifier module and feature scoring method. In Section III, we provide details on these choices and how they strengthen the prediction experiment for any given IT Product.

C. *Prediction workflow*

Our approach to predict the number of unknown vulnerabilities for any given software product follow this workflow:

- Start by generating a dataset for the given IT Product as illustrated in Section II-A.
- Using the above dataset, build and test a predictive experiment within Azure ML Studio as illustrated in Section II-B. Once the above predictive experiment has completed successfully, a trained model and web service will be produced as a result.
- Using a subset of the dataset that was reserved for validation, ensure that the trained model is scoring well against this validation data.
- Then, we expose a middleware application which leverages the above predictive web service endpoint to receive input data as illustrated in Fig. 1 and to return the predicted total number of vulnerabilities along with other associated metadata (such as the prediction accuracy and error rates).
- We can now perform dynamic prediction for any of this IT Product's releases, by first passing the release version source code details for data collection. Then using the generated data as input to the above middleware application, we get the predicted number of vulnerabilities for the assessed software product release.
- The predicted number can then be interpreted with two views, one for the overall accumulated number of vulnerabilities and the other view for the unknown numbers of vulnerabilities (this can be easily computed by subtracting the known vulnerabilities from the predicted ones by taking into consideration the prediction error rate). Since predicted vulnerabilities cannot be classified based on the potential threats that can result from their exposure, we separate them from known vulnerabilities such that users can be aware of the potential risk to their software.

This proposed prediction workflow can be repeated as needed to update the dataset and trained model, as the assessed IT Product's code base changes and new vulnerabilities are getting disclosed.

III. Experimental study and evaluation

In this section, we take an in-depth look at a case study of an IT Product that serves as a proof of concept for this work.

A. *Use case: OpenSSL – software product*

The discovery of the *OpenSSL's* Heartbleed bug [11] in 2014 revealed that this vulnerability remained unknown for two years before it was discovered. Heartbleed and other similar types of vulnerabilities serve as the motivation for our research among other facts that are presented in Section I.

For a proof of concept, we looked at different open-source IT Products and selected the *OpenSSL* [12] software product as our first pick for this study. We selected OpenSSL due to its critical role that it plays in Transport Layer Security (TLS) and Secure Sockets Layer (SSL) protocols. In addition, many other software products rely on it (since it is commercial grade and open source) to build complex softwares and services (this has a wider reach, since any new discovered software vulnerability within OpenSSL package impacts other critical software products that rely on it). Then, we selected OpenSSL due to its maturity (which enable us to collect enough data needed for our study and predictive models). Our selection parameters here, should be applicable in selecting any other software product to study and validate our proposed prediction models.

Using our prediction workflow presented in Section II-C, we apply it to the *OpenSSL* software product.

1) OpenSSL – dataset generation: We downloaded and analyzed 154 OpenSSL versions which are released in these categories: 0.9.x, 1.0.0, 1.0.1, 1.0.2, 1.1.0, and fips [12]. Using our Sweep-toolkit, we pass as input the directory path to all of the OpenSSL versions' uncompressed directories of source codes. Once Sweep-toolkit completes its execution, it returns a dataset for OpenSSL.

This OpenSSL dataset has 154 entries (one per each OpenSSL version), where each entry contains collected data for all specified 124 features (CPE-Name, Year-1999, ..., Year-X, ,..., Year-Current, #CVEIDs, Understand-Metric-1, .., Understand-Metric-n). With:

- *CPE-Name* representing each analyzed OpenSSL release (i.e., *cpe:/a:openssl:openssl:1.0.0f*).
- *Year-1999* representing the number of disclosed vulnerabilities in the year 1999 (which is the first year recorded in NVD data feeds [1]). The other data for the following years are included as well, up to the current year (i.e., for the above CPE instance, in the Year-2014, Sweep-toolkit found 22 disclosed vulnerabilities).
- *#CVEIDs* representing the accumulated number of known vulnerabilities (i.e., 50 is the total number of disclosed vulnerabilities for the above CPE instance example).
- *Understand-Metric-1, ..., Understand-Metric-n* representing all computed software metrics supported by the Understand tool [7] (e.g., the above CPE instance has a *SumCyclomatic* value of 43479, and the other metrics data are provided accordingly).

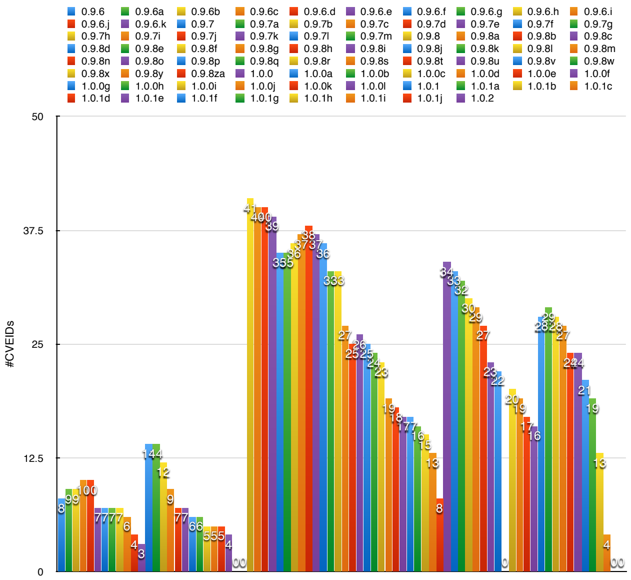

Figure 2. OpenSSL Releases vs. Disclosed Number of Vulnerabilities

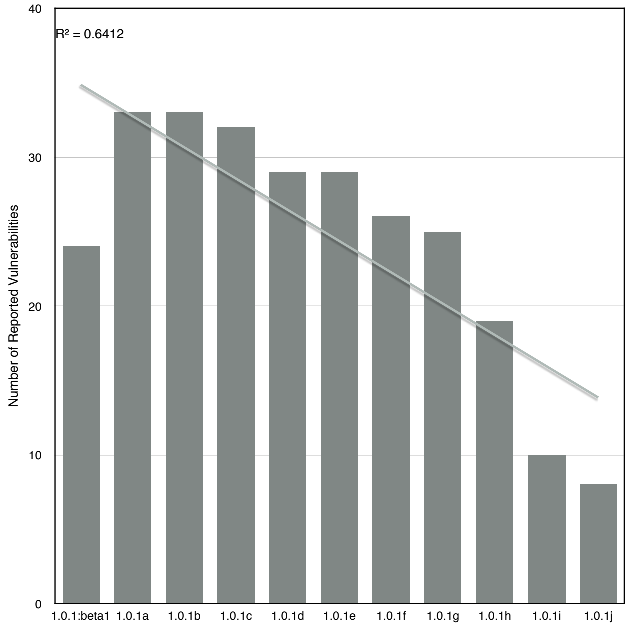

Figure 3. OpenSSL:1.0.1 – Minor Releases Linear Trend

2) OpenSSL – Predictive experiment: Consider tracking product releases (minor and major) and vulnerabilities detected for the OpenSSL product as shown in Fig. 2. By slicing one of the observed trends in Fig. 2, Fig. 3 shows that there is a decreasing trend in terms of vulnerabilities between major releases of the OpenSSL product.

We predict a similar behavior with other IT products since this trend reflects a behavior similar to software maturity or improved quality over time. Here we see the maturity of OpenSSL in terms of the number of reported vulnerabilities over time. We also make the following preliminary observations from the data collected so far, along with this small sample illustrated in Fig. 2 and 3.

- The history of reported vulnerabilities have shown a decreasing trend throughout each IT Product's minor releases (i.e., OpenSSL:1.0.1.:a, b, c, ..., j) in terms of the number of vulnerabilities, with exceptions for some limited distribution versions (see Fig. 3).
- The average number of reported vulnerabilities spiked whenever the rate of minor releases was high after a major release of an IT Product (i.e., for OpenSSL, this is clearly observed by grouping each of the trends (release families) shown in Fig. 2).
- The newly discovered vulnerability in a current IT Product release affects some of the previous releases due to the fact that the versions share a technology (with unknown vulnerabilities), such as common library, framework, design pattern, and so on.
- The straight line in Fig. 3 reflects the evaluated R^2 value (coefficient of determination) which appears to be above 0.5 when using Linear, Polynomial and Exponential trend estimations. However, while using logarithmic and power trends, the R^2 value was less than 0.5. This hints that there is value in knowing how these data can be used to predict the number of vulnerabilities, although we need a more carefully designed approach to correlate the data with vulnerabilities.

To improve our model and produce a high coefficient of determination with the prediction of the number of vulnerabilities, we considered all the features of the vulnerability details and generated source code metrics to build a predictive model as presented in previous Sections. Using Azure ML Studio [10], we built our predictive experiment as illustrated in Fig. 1.

Typically, the first step is to understand the nature of the training dataset which requires preprocessing to remove noisy data and selecting the most significant features that lead to highly confident predictions. Azure ML Studio offers necessary tools to facilitate this pre-processing task such as: *Project Columns, Filter Based Feature Selection*. For the *Filter Based Feature Selection* module, we used the *pearson correlation* feature scoring method selected for identifying the best contributing features (about 25). Another important aspect to decide was on the amount of training data needed for each machine learning technique and this is defined by the *Split* module (using a 50% split to avoid any bias) of Azure ML Studio.

The goal of our research is to predict the number of vulnerabilities for any given IT Product version based on the data we collected. We needed to identify a machine learning algorithm to train and predict using the generated OpenSSL dataset. We found that regression based models are well suited for our purpose. We explored all available Azure ML Studio's [10] machine learning regression models, and determined that a *Boosted Decision Tree Regression* [13] is the best for our predictions. Then, by following the easy to use predictive framework illustrated in Fig. 1, we trained and scored our chosen model using the generated OpenSSL dataset (from the previous section) targeting the #CVEIDs feature name (as presented in Section III-A1) and included its evaluation results in Table I. We present a detailed discussion of our results in Section III-B.

Note that the number of OpenSSL releases used to train

TABLE I. USING OUR LABELED OPENSSL DATASET TO EVALUATE ITS BOOSTED DECISION TREE REGRESSION MODEL

Number of Desired Features	Mean Absolute Error	Relative Absolute Error	Coefficient of Determination
5	7.995	0.618	0.597
10	3.341	0.258	0.927
15	3.347	0.258	0.926
25	3.780	0.292	0.917
40	3.985	0.308	0.912
65	3.851	0.297	0.916
80	3.597	0.278	0.918
95	3.597	0.278	0.918

our model is less than the total number available because the originally produced dataset was divided into sub-datasets for training, testing and validation.

Table II shows a comparison of the number of reported vulnerabilities and the number of predicted vulnerabilities for some OpenSSL releases used for the training experiment (with the selection of 25 desired features selected by the Filter based feature selection as described previously). To use any of the predicted number of vulnerabilities, one has to account for the estimated Mean Absolute Error. The slight differences in the known vulnerabilities and the scored number is a result of the chosen training dataset, and automatically selected number of features. Since, our interest is to predict the number of unknown vulnerabilities for any subsequent prediction, we consider the upper bound predicted value including the error rate.

TABLE II. SAMPLE OF SCORED OPENSSL INSTANCES

OpenSSL Releases	Known #Vulnerabilities	Predicted #Vulnerabilities
cpe:/a:openssl:openssl:0.9.8s	22	17.139
cpe:/a:openssl:openssl:1.0.1f	53	41.747
cpe:/a:openssl:openssl:0.9.7g	35	30.823
cpe:/a:openssl:openssl:1.0.1g	52	43.777
cpe:/a:openssl:openssl:0.9.6m	30	37.448

3) OpenSSL – Predictive Model Validation: In Table III, we present a comparison of the OpenSSL releases sample used to evaluate and validate the scored predictive model. These results tell us how well our model was able to score against our validation dataset.

- Mean Absolute Error : 2.927
- Root Mean Squared Error : 4.068
- Relative Absolute Error : 0.203
- Relative Squared Error :0.059
- Coefficient of Determination :0.940

TABLE III. SAMPLE OF OPENSSL INSTANCES FOR VALIDATION

OpenSSL Releases	Known #Vulnerabilities	Predicted #Vulnerabilities
cpe:/a:openssl:openssl:1.0.1a	60	52.498
cpe:/a:openssl:openssl:1.0.0h	46	46.815
cpe:/a:openssl:openssl:0.9.8m	34	33.867
cpe:/a:openssl:openssl:1.0.2c	15	15.729
cpe:/a:openssl:openssl:1.0.2d	14	11.526

TABLE IV. OPENSSL'S VERSIONS – PREDICTED NUMBER OF VULNERABILITIES

OpenSSL Instances	Known #Vulnerabilities	Predicted #Vulnerabilities
openssl-1.0.0:beta1	0	11.054
openssl-1.0.0:beta2	0	11.054
openssl-1.0.2:beta1	0	11.667
openssl-1.0.2:beta2	0	11.667
openssl-1.1.0:pre2	0	8.641
openssl-engine:0.9.6m	0	11.148
openssl-fips:2.0.9	0	11.148
openssl:0.9.8zd	0	5.489
openssl:1.0.0t	0	8.843

4) OpenSSL – Prediction of Unknown Vulnerabilities: Transforming the previously built predictive model for OpenSSL into a web service endpoint is straightforward. In Table IV, we show some examples of the predicted number of vulnerabilities for some versions of OpenSSL releases using our published web service. These OpenSSL instances have no currently known vulnerabilities, therefore the predicted number reflects the unknown vulnerabilities that may be discovered using different assessment techniques (to be explored in our future work).

B. Discussions

In Table I, we presented our model evaluation results using the ”Boosted Decision Tree Regression” technique for predicting vulnerabilities for the OpenSSL software product. The following preliminary observations can be made from these results and our overall study experience.

- The studied *OpenSSL* dataset fit well with the selected machine learning technique, yielding a high coefficient of determination above 0.5 (which is better than a random guess prediction).
- The scored *OpenSSL* predictive model shows a positive correlation between the known vulnerabilities (#CVEIDs) and predicted ones (Scored Labels), which can be viewed via the scatter plot generated within the Azure ML Studio workspace. This reaffirms one of our hypotheses that we can predict the number of vulnerabilities contained in an IT Product using software metrics and vulnerability disclosure history.
- The coefficient of determination of the scored model is at the lowest, when the desired number of features is set to 5. By taking a close look at the automatically selected 5 features (Year-2010 to Year-2014), it reveals that they are all about the vulnerability disclose history timeline per *Year-X* (this is due to the fact that the trained model targets the accumulated total number of vulnerabilities (*#CVEIDs*) and this targeted feature is a result of these found vulnerabilities timeline).
- The coefficient of determination is improved and reaches its highest, when the desired number of features is increased from 5 to 10 or a higher value. This improvement in the prediction accuracy is a result of our feature selector capabilities used to identify additional and unique features that are part of the computed software metrics (to name a few: *CountLineCodeExe, Knots, CountPath, SumEssential, Cyclomatic*, etc.). Depending on the desired prediction

accuracy, a matching model is scored to build a predictive web service.

Table IV contains results of our evaluation of the *OpenSSL* dataset for new or beta instances (or releases) of the product with no reported vulnerabilities thus far. It should be noted that these two OpenSSL instances (*openssl-1.0.2:beta1* and *openssl-1.0.2:beta2*) have very similar source code bases, therefore we predict that both versions will likely contain the same number of vulnerabilities. These vulnerabilities should be viewed as the potential number of vulnerabilities that will likely be discovered in these products. This information can be used to plan for defensive mechanisms to mitigate security risks due to the unknown vulnerabilities.

C. Recommended Proactive Strategies

Ideally one should be able to implement countermeasures to patch or mitigate risks due to known vulnerabilities. Some organizations provide defensive mechanisms for some known vulnerabilities of popular IT products [14] [15]. However, the rate at which new vulnerabilities are being detected and reported is making it difficult to maintain up-to-date lists of patches. Moreover, as we have shown in this work, IT products very likely contain unknown or yet to be discovered vulnerabilities. Thus it is necessary to explore additional (beyond patching) defensive measures to increase our confidence in IT products. We include some recommendations in this regard.

- Any unknown pattern or behavior observed for an IT Product being assessed via security penetration testing approaches or monitored via deployed security infrastructures can serve as an indicator that zero day (or undiscovered) vulnerabilities are present or being exploited in the IT Product of interest. Therefore, these unknown behaviors can be categorized in our predicted number of unknown vulnerabilities which in turn should raise an awareness to stress test the IT Product to find them.
- We recommend exploring various software rejuvenation techniques in an attempt to mitigate some malware that may be exploiting hidden or unknown vulnerabilities before taking a foothold in the product. It has been shown that software rejuvenation [16] can minimize security risks due to malware. It has also been shown how the cost of rejuvenation can be used to plan the frequency of rejuvenation schedules.

IV. Related work

Yonghee et al. [17] attempted to understand whether there is a correlation between software complexity measure and security, primarily focusing on the JavaScript Engine in the Mozilla application framework. They show a weak correlation, primarily because of the small number of features used. In our study, we expanded on the number of software metrics and used product releases to obtain higher correlations to reported vulnerabilities. Our predictive model works well when there is a large number of product releases and the product has a mature user base.

Other prior works have explored various software properties to serve as indicators of vulnerabilities where they used techniques such as software complexity and developer activity metrics [18] [19] [3]. Then using these software metrics coupled with some empirical models, there are works [20] [21] [22] [23] that have proposed solutions towards representing and predicting trends in software vulnerabilities. Our research has some key concepts similar to these works, but we extend the scope in terms of automatic data analysis and vulnerability prediction.

V. Conclusion and Future Work

In this paper, we presented our novel approach for predicting the number of unknown vulnerabilities in a given IT Product. We have shown how to generate a dataset that represents product maturity in terms of source-code base growth and vulnerability disclosure history. We have shown how to use such a dataset and develop a model that results in accurate predictions. We used the Azure cloud based machine learning framework for this purpose. We validated our approach for the *OpenSSL* IT product. We plan to broaden our experimentation to support any open-source software product and extend the number of features to include in the relevant dataset.

Our proposed approach for analyzing the source code of a given IT Product and leveraging its vulnerability disclosure history toward building a predictive model serves as a basis for building other solutions. A planned solution that can leverage our model is to categorize the predicted number of vulnerabilities into threat types (i.e., STRIDE [24]) using some inherent IT Product properties along with some actionable threat intelligences and, in turn, propose relevant mitigation techniques to counter these vulnerabilities and threats.

The other aspect of our work that we plan to extend is the ability to design and train our predictive model in a generic way that would allow IT Products that may need a different machine learning and training approach. We plan to group IT products into different categories, identify representative features of products that belong to a group and provide suggested approaches that result in highly accurate prediction of security vulnerabilities. We plan to expand on the IT product features to enhance our prediction accuracies using security threat intelligence reports, inherent vulnerabilities associated with different programming languages and development platforms.

Acknowledgment

The authors would like to acknowledge Mr. David Struble, former Senior Software Technologist in Raytheon Company's Net-Centric Systems group, for his editorial contributions to this paper. Our research is supported in part by the NSF Net-centric and Cloud Software and Systems Industry/University Cooperative Research Center and its member organizations.

References

[1] "National Vulnerability Database," 2016, URL: https://nvd.nist.gov/ [accessed: 2016-07-12].

[2] T. L. Graves, A. F. Karr, J. S. Marron, and H. Siy, "Predicting fault incidence using software change history," in IEEE Transactions on Software Engineering," IEEE, vol. 26, no. 7, pp. 653–661, 2000.

[3] I. Chowdhury and M. Zulkernine, "Using complexity, coupling, and cohesion metrics as early indicators of vulnerabilities," in Journal of Systems Architecture," Elsevier, vol. 57, no. 3, pp. 294–313, 2011.

[4] M. Bozorgi, L. K. Saul, S. Savage, and G. M. Voelker, "Beyond heuristics: learning to classify vulnerabilities and predict exploits," in Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining," ACM, pp. 105–114, 2010.

[5] S. Zhang, D. Caragea, and X. Ou, “An empirical study on using the national vulnerability database to predict software vulnerabilities,” in Database and Expert Systems Applications,” Springer, pp. 217–231, 2011.

[6] “Common Platform Enumeration (CPE),” 2016, URL: http://scap.nist.gov/specifications/cpe/ [accessed: 2016-07-12].

[7] “What Metrics does Understand have?” 2016, URL: https://scitools.com/support/metrics_list/ [accessed: 2016-07-12].

[8] “Understand Scitools,” 2016, URL: https://scitools.com/ [accessed: 2016-07-12].

[9] “Sweep-Toolkit,” 2016, URL: https://github.com/kamongi/sweep-toolkit [accessed: 2016-07-12].

[10] “Microsoft Azure Machine Learning,” 2016, URL: https://studio.azureml.net/ [accessed: 2016-07-12].

[11] “Heartbleed Bug,” 2016, URL: https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2014-0160 [accessed: 2016-07-12].

[12] “OpenSSL,” 2016, URL: https://www.openssl.org/ [accessed: 2016-07-12].

[13] “Boosted Decision Tree Regression,” 2016, URL: https://msdn.microsoft.com/en-us/library/azure/dn905801.aspx [accessed: 2016-07-12].

[14] “Apple security updates,” 2016, URL: https://support.apple.com/en-us/HT201222 [accessed: 2016-07-12].

[15] “Adobe – Security Bulletins and Advisories,” 2016, URL: https://helpx.adobe.com/security.html [accessed: 2016-07-12].

[16] C.-Y. Lee, K. M. Kavi, M. Gomathisankaran, and P. Kamongi, “Security Through Software Rejuvenation,” in The Ninth International Conference on Software Engineering Advances (ICSEA),” IARIA, pp. 347–353, 2014.

[17] Y. Shin and L. Williams, “Is complexity really the enemy of software security?” in Proceedings of the 4th ACM workshop on Quality of protection” ACM, pp. 47–50, 2008.

[18] Y. Shin, A. Meneely, L. Williams, and J. A. Osborne, “Evaluating complexity, code churn, and developer activity metrics as indicators of software vulnerabilities,” in IEEE Transactions on Software Engineering,” IEEE, vol. 37, no. 6, pp. 772–787, 2011.

[19] Y. Shin, “Exploring complexity metrics as indicators of software vulnerability,” in Proceedings of the 3rd International Doctoral Symposium on Empirical Software Engineering, Kaiserslautem, Germany, 2008, URL: http://www4.ncsu.edu/~yshin2/papers/esem2008ds_shin.pdf [accessed: 2016-07-12].

[20] Y. Shin and L. Williams, “An empirical model to predict security vulnerabilities using code complexity metrics,” in Proceedings of the Second ACM-IEEE international symposium on Empirical software engineering and measurement,” ACM, pp. 315–317, 2008.

[21] O. H. Alhazmi, Y. K. Malaiya, and I. Ray, “Measuring, analyzing and predicting security vulnerabilities in software systems,” in Computers & Security,” Elsevier, vol. 26, no. 3, pp. 219–228, 2007.

[22] O. H. Alhazmi and Y. K. Malaiya, “Modeling the vulnerability discovery process,” in 16th IEEE International Symposium on Software Reliability Engineering, 2005. ISSRE 2005.,” IEEE, pp. 10–pp, 2005.

[23] ——, “Prediction capabilities of vulnerability discovery models,” in Reliability and Maintainability Symposium, 2006. RAMS’06. Annual,” IEEE, pp. 86–91, 2006.

[24] “The STRIDE Threat Model,” 2016, URL: http://msdn.microsoft.com/en-US/library/ee823878(v=cs.20).aspx [accessed: 2016-07-12].