Towards Prioritizing Vulnerability Testing

Halit Alptekin*, Simge Demir, Şevval Şimşek, and Cemal Yilmaz
Faculty of Engineering and Natural Sciences
Sabancı University
Istanbul, Turkey
Email : {simgedemir,sevvalboylu,cyilmaz}@sabanciuniv.edu, *info@halitalptekin.edu }

***Abstract*— Vulnerability assessment is the process of identifying and prioritizing the vulnerabilities in a system. Vulnerability scanners can, for example, scan a website for known vulnerabilities by running a repository of security tests, each of which is designed to reveal a known vulnerability. As the security tests need to be executed on each and every web page encountered, it may take quite a while for these scanners to report vulnerabilities. In this work, we present an approach for revealing the vulnerabilities faster by prioritizing the executions of the security tests on a per web page basis. The approach is based on a simple conjecture that "similar" web pages may possess "similar" vulnerabilities and that identifying these similarities can help prioritize the security tests. The results of the experiments we carried out by using 2927 distinct web pages (collected from 80 web sites), support our basic hypothesis; the percentages of the times the actual vulnerabilities appear in the top 8 and 15 predicted vulnerabilities were 86.9% and 98.4%, respectively.**

Keywords—vulnerability analysis, vulnerability prioritization, automated testing, test prioritization

I. INTRODUCTION

Vulnerability detection and analysis [1] is a process of crawling a system under test with a goal of revealing the security vulnerabilities that the system may have. When it comes to ensuring the security of websites [2], which is the main focus of this paper, one way to carry out this process is to use automated tools, such as Netsparker [3]. In the remainder of the paper, these tools will be referred to as *vulnerability scanners*.

Vulnerability scanners typically implement a repository of test cases, each of which is developed for testing a website against a known vulnerability. Given a website, a vulnerability scanner crawls the website and runs all the applicable test cases in the repository often in an arbitrary order on each and every web page encountered. The potential vulnerabilities of the website are then reported for end-users, so that appropriate countermeasures can be taken in time. Consequently, the faster the vulnerabilities are reported, the better it is.

Considering, however, that websites may have a large number of web pages and that vulnerability scanners may have hundreds (if not thousands) of test cases in their repositories, it may take quite a while for these tools to finish scanning a website for vulnerabilities. When this factor is coupled with the fact that websites should ideally be scanned after every update in their codebases, speeding up the process of revealing vulnerabilities becomes a matter of great practical importance.

In this work, we propose an approach, which aims at prioritizing [4] the test cases on a per web page basis to reveal vulnerabilities faster. The roots of the proposed approach stem from a simple observation we have been consistently making with security experts: Given a web page, security experts typically have an idea about what may go wrong with it by simply glancing through the certain properties of the page, such as the URL, the content, and the look and feel of the page. We, therefore, conjecture that "similar" web pages are likely to possess "similar" vulnerabilities and that exploiting these similarities can help reveal vulnerabilities faster by prioritizing the security tests, such that the tests that are capable of revealing the suspected vulnerabilities can be executed before the other tests. Note that although our focus in this work is on web applications, the same idea can be applied to other domains, such as to mobile testing where similar screens in mobile applications may suffer from similar vulnerabilities.

The remainder of the paper is organized as follows: Section II introduces the proposed approach; Section III describes the experiments we carried to evaluate the proposed approach; and Section IV concludes with some future work ideas.

II. APPROACH

We cast the problem of prioritizing security tests to a multi-class classification problem, where each class corresponds to a vulnerability. Since a web page may have multiple vulnerabilities, instead of making only one prediction for a given page (e.g., assigning a single class to the page), we make multiple predictions by ordering all the classes according to their likelihood (from the most probable one to the least probable one). We then report the top *N* predictions (for this work, $N \in \{1, 3, 5, 8, 15\}$) as the most probable vulnerabilities that may be possessed by the web page. Once these vulnerabilities are identified, the security tests developed to reveal them can be given priority.

To this end, given a web page, we extract a number of features (for this work, 50 features) from the page, regarding the URL, the HTTP request/response contents, the server headers, and the authentication methods employed by the server. Each feature is represented as a binary attribute representing whether the respective feature is present in the web page or not. In our model, the features that are extracted from reports would create the feature vector for training in the light of our assumption. All of these features are binary, representing whether the feature exists in the page (=1) or not (=0).

III. EXPERIMENTS

To evaluate the proposed approach, we carried out a series of experiments.

A. *Operational Framework*

In these experiments, we scanned 80 websites for vulnerabilities by using Netsparker. Netsparker generated us a total of 2927 reports, each of which was for a distinct web page. The report of a web page contained a list of CWE (Common Weakness Enumeration) indices [5], representing the vulnerabilities that the page possessed. We use these CWE indices as the classes to be predicted.

For a web page, we first extracted 50 binary features from the page (Section II) and created a separate record for each vulnerability possessed by the page. That is, each record had the 50 features extracted from the web page as the attributes and a distinct vulnerability possessed by the page as the class. All told, we had a total of 3291 records as many of the web pages had more than one reported vulnerabilities. We then randomly splitted this data set into non-overlapping training (80%) and test sets (20%).

To train the prediction models, we used Keras MultiClass Classifier [6], which is a neural network-based classifier. In particular, we used three-layered neural networks[7] and experimented with two different activation functions, namely *softmax* and *sigmoid*. The models were trained by using the training set and evaluated by using the test set.

B. *Evaluation Framework*

Given a record obtained from a web page, the output of the prediction model was a ranked list of probable classes (from the most probable to the least probable) that the page may belong to, i.e., a ranked list of vulnerabilities that the page may have. We then reported the top $N \in \{1, 3, 5, 8, 15\}$ predictions.

To evaluate the quality of the predictions, we computed *accuracy* as the percentage of the times the classes associated with the records appear in the top N predictions. The higher the accuracy, the better the approach is.

C. *Data and Results*

Table I summarizes the results we obtained. In this table, the columns indicate the activation function used and the accuracies obtained when N = 1, 3, 5, 8, and 15, respectively.

The *softmax* model, compared to the *sigmoid* activation function, led to better accuracies when N = 1 and N = 15. For the remaining values of N, however, the *sigmoid* model function performed better. For example, when N = 8 the *sigmoid* model provided an accuracy of 86.9%. And, when N = 15 the *softmax* model provided an accuracy of and 98.4%. Compared to the no information case, if top 15 predictions are taken into consideration, this corresponds to 0.01856 of all possible weakness types, hence the time and resource gained is dramatically high. There is also a top 25 list [8] published in 2019, if we were to take that list into consideration, top 15 predictions correspond to 60% of the time and resources, and we should not forget that some of the vulnerabilities that our classifier predicts might not even be on the list and would be missed if the software were only tested for weaknesses in Top25 list [8].

Given that there are typically dozens of vulnerabilities to be checked against a web page, the results of our experiments, suggest that the proposed approach can help prioritize security tests, such that vulnerabilities can be revealed faster.

Activation functions	***Top1***	***Top3***	***Top5***	***Top8***	***Top15***
softmax	**0.527**	0.696	**0.776**	0.865	**0.984**
sigmoid	0.492	**0.701**	0.771	**0.869**	0.974

Table I. Accuracy of Keras Multiclass Classifier

IV. CONCLUDING REMARKS

We believe that this line of research is promising and of great practical importance. Therefore, we continue working on developing new approaches for prioritizing security tests. One possible avenue for future research is to evaluate different classification models for predictions. Another avenue is to increase both the number and the diversity of the features we extract from web pages and leverage them together with various feature selection algorithms. Last but not least, the proposed approach needs to be evaluated with more websites and its ability in revealing security vulnerabilities faster than the existing approaches, should be demonstrated.

REFERENCES

[1] Cohen et al., “Method to consolidate and prioritize web application vulnerabilities” U.S. Publication 2007/0094735 A1, April 26,2007

[2] Hurst et al., “Webcrawl internet security analysis and process” U.S. Patent 7.444,680 B2, Oct. 28, 2008

[3] Netsparker, Netsparker Ltd. Netsparker Ltd. Accessed: Mar 18, 2020. Available: https://www.netsparker.com/

[4] M.G. Dondo, “A Vulnerability Prioritization System Using A Fuzzy Risk Analysis Approach”, Proceedings of The Ifip Tc 11 23rd Intern. Info. Security Conf. 2008 – The International Federation for Information Processing, vol 278. Springer, Boston, MA, 2008, pp. 525-539

[5] “Common Weakness Enumeration, A Community-Developed List of Common Software and Hardware Weakness Types”, Accessed: Mar 18, 2020. Available: https://cwe.mitre.org/index.html

[6] Chollet, F., & others. (2015). Keras. https://keras.io.

[7] “ Neural network models”, Accessed: Mar 18, 2020. Available: https://scikit-learn.org/stable/modules/neural_networks_supervised.html

[8] Cwe.mitre.org. (2020). *CWE - 2019 CWE Top 25 Most Dangerous Software Errors*. [online] Available at: https://cwe.mitre.org/top25/archive/2019/2019_cwe_top25.html