ESTUDAM Bilişim Derg, 2023; 4(2): 1-9.

J ESTUDAM Information, 2023; 4(2): 1-9.

DOI: 10.53608/estudambilisim.1233344

(Geliş Tarihi / Received Date: 12.01.2023, Kabul Tarihi/ Accepted Date: ,03.04.2023)
B. Alagha , "XSS Attack Detection with N-Gram Based Prediction Model", Eskişehir Türk Dünyası Uygulama ve Araştırma Merkezi Bilişim Dergisi, c. 4, sayı. 2, ss. 1-9, May. 2023, doi:10.53608/estudambilisim.1233344

(**Research Article**)

XSS Attack Detection with N-Gram Based Predictive Model

Bilal ALAGHA[*1]

[1] Eskişehir Osmangazi Üniversitesi, Mühendislik Mimarlık Fakültesi, Bilgisayar Mühendisliği Bölümü, 26040, Eskişehir, ORCID No : https://orcid.org/0000-0001-8347-1841

Keywords:
XSS Attacks,
Predictive Model,
Machine Learning,
Word Tokenization,
N-Gram,
Algorithm,

Abstract: The increment developments in technology has empowered the web applications. Meanwhile, the existence of Cross-Site Scripting (XSS) vulnerabilities in web applications has become a concern for users. In spite of the numerous current detection approaches, attackers have been exploiting XSS vulnerabilities for years, causing harm to the internet users. In this paper, a text-mining based approach to detect XSS attacks in web applications is introduced. This approach is built to extract a set of features from a publicly available source code files, which are then used to build a prediction model. The findings include few comparisons between Word Tokenization and N-Gram in accuracy, time spend to build the model and AUC-ROC curve. The results show that N-Gram tokenization outperforms the Word Tokenization.

(**Araştırma Makalesi**)

N-Gram Tabanlı Tahmin Modeli ile XSS Saldırısı Algılama

Anahtar Kelimeler:
XSS Saldırıları,
Tahmin Modeli,
Makine Öğrenme,
Kelime Simgeleştirme,
N-Gram,
Algoritma,

Özet: Teknolojideki hızlı gelişmeler web uygulamalarını güçlendirmiştir. Bu arada, web uygulamalarında Siteler Arası Betik Çalıştırma (XSS olarak da bilinir) güvenlik açıklarının varlığı, kullanıcılar için bir endişe haline gelmiştir. Çok sayıda mevcut algılama yaklaşımına rağmen saldırganlar yıllardır XSS güvenlik açıklarından yararlanarak internet kullanıcılarına zarar veriyor. Bu makalede, web uygulamalarındaki XSS Saldırısını tespit etmek için metin madenciliği tabanlı bir yaklaşım tanıtılmaktadır. Bu yaklaşım, halka açık bir kaynak kod dosyalarından bir dizi özellik çıkarmak için oluşturulmuştur. Bu dosyalar, daha sonra bir tahmin modeli oluşturmaya yardımcı olmak için kullanılır. Bulgular, N-Gram ve Kelime Simgeleştirme arasında doğruluk, modeli oluşturmak için harcanan zaman ve AUC-ROC eğrisi birkaç karşılaştırma içerir. Sonuç olarak N-Gram Kelime Simgeleştirmesinden daha iyi performans göstermektedir.

1. INTRODUCTION

Security of information is always a challenging task, and new challenges are discovered consistently. The communication between web server and web browser used to be secure, and the communication among people and devices over HTTPS protocol is widely known. However, lately, we have witnessed an increase in exploitation of web application vulnerabilities. One of those vulnerabilities is the Cross Site Scripting (also known as XSS). It is challenging to eliminate those vulnerabilities because it is difficult for web applications to clear all user input appropriately. By utilizing XSS, attackers can access their victims' cookies and perform keylogging and phishing attacks to see their sensitive information such as passwords, credit card numbers, and other sensitive information [1].

These days, everyone can evaluate the security level of any website. Publicly available testing tools such as sitecheck.sucuri.net, which developed by Sucuri Inc. Such tools enable us to scan web servers and evaluate them on scale from minimal to critical [2]. The methodology used in such websites is not scientific, but it gives an overview on the security level that web servers have.

*Bilal Alagha: 503020210017@ogrenci.ogu.edu.tr

Implementation of a text-mining technique to detect XSS attacks is the main idea of this article. Text-mining consists of the discovery of previously unknown information from existing resources [3]. It uses techniques from information retrieval, information extraction, and Natural Language Processing (also known as NLP) and connects them with the algorithms, statistics and machine learning technique [4].

XSS attacks occupy the first positions as one of the most common types of web application attacks, which considered to be critical and direct threats to web application security. According to the Positive Technologies report, 62% of web application vulnerabilities considered to be high-severity vulnerabilities compared to the total number of web application vulnerabilities in 2021 [5]. Surprisingly, XSS is classified as one of the top 10 vulnerabilities in web applications [6].

In this paper, an N-Gram based predictive approach to detect XSS attacks in web applications is introduced. It employs a text-mining technique to extract the features from source code files. These features are then used in the generation of a prediction model, which classifies the source code files into vulnerable and invulnerable categories.

2. LITERATURE REVIEW

There are many different methods proposed by researchers to detect XSS attacks and prevent their vulnerabilities. For example, Doupé et al. [7] implement a prototype of "deDacota" to analyze and rewrite ASP.NET web applications. This system is then applied to six open-source and real-world ASP.NET applications. Finally, all known XSS vulnerabilities are eliminated through the verification process.

Weissbacher et al. [8] state that 417 out of 815 websites have CSP enabled because the "phpMyAdmin" web application is hosted. In order to avoid this situation, each domain is only connected once until a successful connection is made. If a redirect URL response is received, then the crawler follows that URL to visit the actual site.

Scholte et al. [9] present “IPAAS” system, which is a technique for preventing the exploitation of XSS and SQL injection vulnerabilities based on automated data type detection of input parameters. IPAAS automatically augments otherwise insecure web application development environments with input validators that result in great security developments for real systems. IPAAS for PHP is implemented and evaluated on five web applications with known XSS and SQL injection vulnerabilities. The evaluation shows that IPAAS would have disallowed 83% of SQL injection vulnerabilities and 65% vulnerabilities resulting from XSS.

Scholte et al. [10] present the design, implementation and evaluation of a client-side countermeasure, which is capable to stand against XSS attacks that are DOM-based. The mechanism relies on the combination of a taint-enhanced JavaScript engine and taint-aware parsers, which block the parsing of attacker controlled syntactic content. In case of client-side vulnerabilities, the approach detects injected syntactic content and, thus, is superior in blocking DOM-based XSS.

Salas and Martins [11] propose an approach that makes use of two Security Testing techniques, namely Penetration Testing and Fault Injection. These techniques help emulate XSS attack against web services. This technology, combined with WS-Security and Security Tokens, can identify the sender and guarantee the legitimate access control to the SOAP messages exchanged. The vulnerability scanner soapUI is used, which is one of the most recognized tools of Penetration Testing. In contrast, WSInject is a new fault injection tool, which introduce faults or errors on web services to analyze the behavior in an environment not robust. This approach can analyze the robustness of web services by Fault Injection with WSInject. The results of the Penetration Testing phase assist to improve the rules for the analysis of vulnerabilities. The security provided by WS-Security standard with the add-on Security Token against XSS attack is also verified. In both phases, the use of WS-Security reduces the number of vulnerabilities to a large degree.

Niakanlahiji and Jafarian [12] present WebMTD, a proactive moving target defense mechanism that thwarts a broad class of code injection attacks on web applications, including XSS, HTML code injection, and server-side code injection attacks. Relying on built-in features of current web browsers, WebMTD randomizes certain attributes of web elements to differentiate the application code from the injected code and prevent the execution.

Athanasopoulos et al. [13] introduce xJs framework to reduce XSS attacks. The fundamental norm is to transfer JavaScript code blocks to another domain on the web server at runtime and to reverse the transposition on the client browser. The XOR operation is used to transfer JavaScript blocks in static HTML documents. Upon requesting an HTML document, xJS XORs each script block on the web page with a private key. Afterwards, it contains the key in an HTTP response header. On the client’s browser, the encrypted code block will be XORed again with the transmitted private key in order to retrieve the original code blocks.

Gundy et al. [14] introduce Noncespaces technique to prevent XSS attacks. In Noncespaces, a random XML namespace is generated for each XHTML document that is requested, and all trusted XHTML element tags in it is then edited to begin with the generated namespace. XHTML elements with proper namespace is then rendered or executed. Noncespaces approach can effectively prevent XSS attack, it requires only checking the tag names on a web page.

Gupta et al. [15] present the Document Object Model (DOM)-Guard, a mobile cloud-based framework that alleviates the DOM-based XSS vulnerabilities from the contemporary platforms of mobile cloud-based HTML5 web applications. DOM-Guard is a runtime DOM tree generator and context-aware sanitization-based framework that scans for XSS vulnerabilities that are DOM-based in the mobile cloud-based HTML5 web applications.

Pelizzi and Sekar [16] present XSSFilt system, which is a modern client-side XSS defense system that addresses the disadvantages of built-in filters of Internet Explorer and Google Chrome browsers. The architecture of XSSFilt is presented, which is a browser-resident XSS defense. Unlike the previous browser resident defenses, which relies on exact string matching, XSSFilt system uses approximate string matching. Additionally, policies to detect attacks involving the whole or partial scripts injections are presented.

Table 1. presents a summary of the detection approaches and some contributions of few researchers in XSS attacks and vulnerabilities detection.

Table 1. Summary of detection approaches and some researchers' contributions

Approach	Author	Area of focus	Type of XSS	Comment
Client-side	Mitropoulos et al. [17]	Attack Detection and Prevention	Reflected and Stored XSS	No false positives encountered during the test
	Gupta et al. [18]	Attack Detection and Vulnerability Detection	DOM based XSS	System used training and detection modes
	Weissbacher et al. [19]	Attack Detection and Prevention	Not specified	ZigZag approach to defend benign-but-buggy JavaScript applications against CSV attacks
Server-side	Maurya [20]	Attack Detection and Prevention	Stored XSS	Uses one-level (i.e. whole HTML tags are allowed) and two-level (i.e. specific tags are allowed) whitelist
	Gupta et al. [21]	Attack Detection	Reflected XSS	Framework detects the XSS worms with low false positives and false negatives
	Guo et al. [22]	Vulnerability Detection	Reflected and Stored XSS	Optimized attack vector repertory; an optimization model to reduce the size of XSS attack vectors, and detect XSS vulnerabilities
Client-Server	Sundareswaran et al. [23]	Attack Prevention	All	XSS-Dec, a security-by-Proxy approach to protect end-users against XSS attacks
	Panja et al. [24]	Attack Detection and Prevention	All	Maintains a cache, which may be corrupted or injected
	Goswami et al. [25]	Attack Detection	Reflected and Stored XSS	Combined attribute clustering algorithm and rank aggregation to cluster the malicious and benign scripts
	This paper	Attack Detection	Not Specified	N-Gram Tokenization outperforms the word tokenization

3. CROSS SITE SCRIPTING (XSS) ATTACK TYPES

XSS can be classified into three categories—Stored XSS, Reflected XSS, and DOM-based XSS [26]. Figure 1 depicts the XSS attack taxonomy.

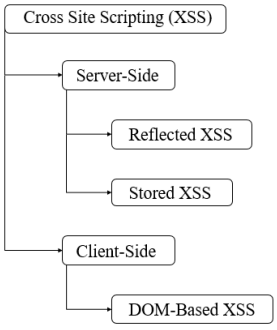

Figure 1. Taxonomy of XSS attacks [27]

1) Reflected XSS: an attacker injects browser executable code within URI or HTTP parameters. It only harms users who click the maliciously crafted link to redirect to the third-party web page embedded with malicious code.
2) Stored XSS: a malicious script is injected directly into a web application with a backend server. The script is stored in the server so that any user who visits the application will be harmed.
3) DOM based XSS: it is an XSS attack modifying the DOM (Document Object Model) in user browser where the original script on user side will be executed in the manner different to its original intension. That is, the web page itself including the HTTP response does not change, but the code of the user side contained in the page executes differently due to the malicious modifications to the DOM environment.

Since these three categories were confusing, the research community proposed and started using two new major terms to help organize these categories—Server-Side XSS and Client-Side XSS [28], which are also depicted in Figure 1.

1) Server XSS: it occurs when an untrusted user injects data in an HTTP response generated by the server. The source of this data could be from the request, or from a stored location. Therefore, we get both Reflected Server XSS and Stored Server XSS.
2) Client XSS: it occurs when an untrusted user injects the malicious payload to update the DOM, typically with a malignant JavaScript call, or via some untrusted data source that is utilized by the JavaScript on the page. That is, any vulnerability or attack that does not use a web server to serve the payload to the victim is Client XSS.

4. MATERIALS AND METHODS

The goal of this study is to allow users to safely display websites that contain safe scripts. In this section, the dataset, experimental setting and performance measures that is used to evaluate the predictive model are discussed.

4.1. Dataset

To evaluate the performance of different approaches, an open source dataset repository is used [29]. This dataset has 10080 PHP source code files, 5728 of them are safe (i.e. invulnerable) and 4352 files are unsafe (i.e. vulnerable). Evaluation of the proposed method is performed on this dataset; it provides various cases to evaluate the efficiency of predictive models. This dataset contains a set of PHP source codes with their vulnerability labels.

4.2. Machine Learning Algorithms

A feature set with different machine learning algorithms generates different prediction models and produces different results. In this experiment, three machine learning algorithms are used- IBk (also known as Instance Based Learner), Random Tree and Decision Stump- with the default parameter values to evaluate their prediction performance. The IBk algorithm [30] uses a distance measure to locate k “close” instances in the training data for each test instance. It uses those selected instances to make a prediction. A Random Tree [31] is a graphical tree that is formed by a stochastic process, and it includes some types. Decision Stump [32-33] is a machine learning classification model that has a single level decision tree.

4.3. Feature Extraction Algorithms

Feature extraction is an essential task for building predictive models. The proper set of features provides better performance for predictive models. Figure 2 depicts the proposed approach, which uses Attribute Selected Classifier. It selects attributes based on the training data only, even when working with cross-validation. Then it trains the classifier on the training data, and then it evaluates the whole process on the testing data. Here, Correlation Attribute Evaluator is used, it evaluates the worth of an attribute by measuring the correlation between it and the class.

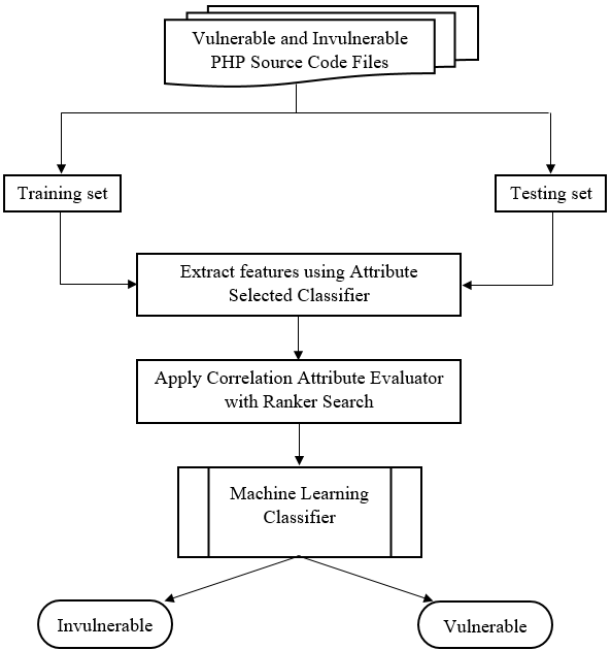

Figure 2. Flow chart of the proposed vulnerability prediction approach

4.4. Tokenization

In text-mining, tokenization is the process of replacing data with unique identification symbols that retain all the essential information about the data without compromising its security. In this paper, the following tokenizers are used to evaluate the accuracy of the predictive model:

4.4.1. Word tokenization

It is the process of splitting a large text into words. This is necessary in NLP tasks, where each word needs to be captured and subjected to further analysis. Hvitfeldt and Silge [34] have details about word tokens (Figure 3).

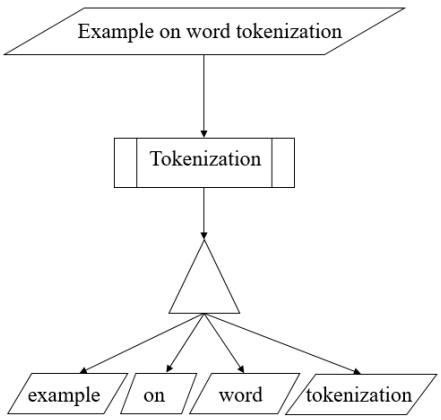

Figure 3. Word tokenization [34]

4.4.2. N-Gram

It is a contiguous sequence of N items from a text. N-Gram is used as abasis for operating an N-Gram based model, which is beneficial in NLP as a method of predicting upcoming text. Cavnar and Trenkle [35] have detailed explanation of N-Gram. Hvitfeldt and Silge [34] also provide a code explanation of it (Figure 4).

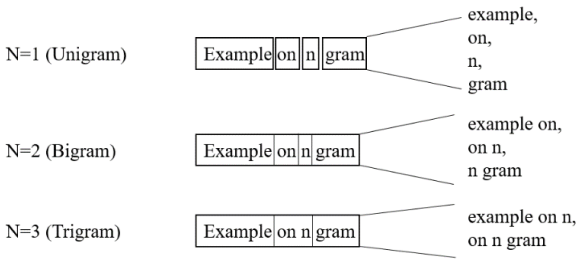

Figure 4. N-Gram model [34]

4.5. Experimental Setting

In this paper, features for the dataset are extracted, and different machine-learning models are used. The experiments are completed using the machine-learning Weka tool [36]. Weka is an open source, platform-independent and publicly available tool, which includes the implementation of different machine learning algorithms for data mining and machine learning experiments [37].

To evaluate the performance of different approaches, a 10-fold cross-validation methodology is applied, and the experiment is repeated 10 times. Each time, one part of the files is used as test and the remainder parts are used as training. The final performance is reported by the weighted average of the accuracy.

4.6. Performance Measure

As a prediction approach, some performance measures are used in order to evaluate the performance of the experiments. These measures are illustrated in Table 2 and described as follows [38]:

1) True Positive Rate (TPR): it represents the number of actual vulnerable files correctly predicted as vulnerable by this predictive model.
2) False Positive Rate (FPR): it represents the number of actual invulnerable files wrongly predicated as vulnerable by this predictive model.
3) True Negative Rate (TNR): it represents the number of actual invulnerable files correctly predicted as invulnerable.
4) False Negative Rate (FNR): it represents the number of actual vulnerable files wrongly predicated as invulnerable.

The most effective predictive model should have the highest possible values of TPR and TNR, and the lowest possible values of FNR and FPR.

Table 2. Confusion matrix

Actual \ Predicted	vulnerable	Invulnerable
vulnerable	TP	FN
Invulnerable	FP	TN

I have also assessed the model performance by the following indicators [38]:

1) Accuracy: it is the fraction of accurately determined vulnerable or invulnerable files to total number of files. It indicates the percentage of correct results.

$$Accuracy = \frac{(TP + TN)}{(TP + TN + FP + FN)} \quad (1)$$

2) Precision: it is the fraction of accurately determined vulnerable files to total number of files that are predicted as vulnerable. It is the probability that a file classified as vulnerable is indeed vulnerable.

$$Precision = \frac{TP}{(TP + FP)} \quad (2)$$

3) Recall: it is the fraction of the number of accurately determined vulnerable files to the actual number of vulnerable files.

$$Recall = \frac{\mathrm{TP}}{(\mathrm{TP} + \mathrm{FN})} \quad (3)$$

4) F-measure: it is a measure of a test's accuracy. It is calculated from the precision and recall of the test. The traditional F-measure [39] is the harmonic mean of precision and recall:

$$F_1 = 2\,\frac{Precision \cdot Recall}{Precision + Recall} = \frac{2TP}{(2TP + FP + FN)} \quad (4)$$

A more general F-measure, F_β [40], that uses positive real factor β, where β is chosen such that recall is considered β times as important as precision, is:

$$F_\beta = (1 + \beta^2) \cdot \frac{Precision \,.Recall}{(\beta^2 \,.\, Precision) + Recall} \quad (5)$$

5. RESULTS

Table 3 illustrates the results of the evaluation measures, i.e. TPR, FPR, precision, recall and F-measure, using word tokenization in machine learning algorithms. The best results can be obtained when using Decision Stump classifier.

Table 3. Average results of the obtained TPR, FPR, Precision, Recall and F-measure using word tokenizer

Measure \ ML	TPR	FPR	Precision	Recall	F-Measure
IBk	0.943	0.057	0.944	0.943	0.943
Random Tree	0.985	0.015	0.985	0.985	0.985
Decision Stump	1.000	0.000	1.000	1.000	1.000

However, the approach performs even better when using N-Gram tokenization over word tokenization in various machine learning algorithms. For example, the false positive rate is 0.014 and the F-measure is 0.986 using IBk classifier, which is better than the best performance of the word tokenization that produces a false positive of 0.057 and an F-measure of 0.943. Even though the range between both tokenizers is not significant, but it may go up using larger N. In this experiment, the N values are selected from 1 to 3. The evaluation measures obtained as a result of using N-Gram tokenizer are depicted in Table 4.

Table 4. Average results of the obtained TPR, FPR, Precision, Recall and F-measure using N-Gram tokenizer, where N=[1-3]

Measure / ML	TPR	FPR	Precision	Recall	F-Measure
IBk	0.986	0.014	0.986	0.986	0.986
Random Tree	0.993	0.007	0.993	0.993	0.993
Decision Stump	1.000	0.000	1.000	1.000	1.000

In order to extract the best features, Attribute Selected Classifier for feature extraction is applied, it selects attributes based on the training data only. The Correlation Attribute Evaluation with Ranker search is then used to evaluate the worth of the attributes. The predictive model is prepared using various algorithms, which are IBk, Random Tree and Decision Stump. Experimental results, as shown in Figure 5, indicate that Decision Stump outperforms the other algorithms used in this experiment, even though it is closed to their results.

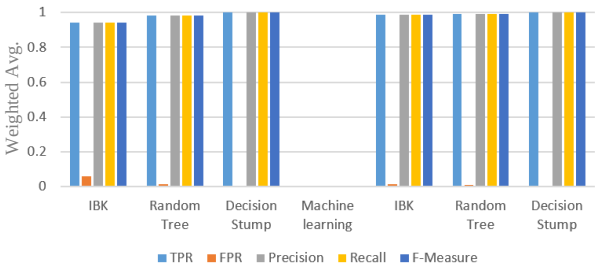

Figure 5. Weighted Avg. Vs Machine Learning Classifier for two different tokenizers

Decision Stump is a machine learning model that consists of a one-level decision tree [41], which uses part of training data for testing. During training time, it acquires knowledge about that data, so if the same data is given to it again, it produces the same results. Thus, this classifier produces better results than the other machine learning classifiers.

Another significant thing to note is that N-Gram takes much more time to build the vulnerability prediction model. For instance, the N-Gram tokenizer takes 25.61 seconds to build the model using the Decision Stump, but it only takes 5.69 seconds to build it with word tokenization. N-Gram takes more time to build the model, but it definitely produces better values than word tokenization. Figure 6 demonstrates the time spent by both tokenizers to build the model.

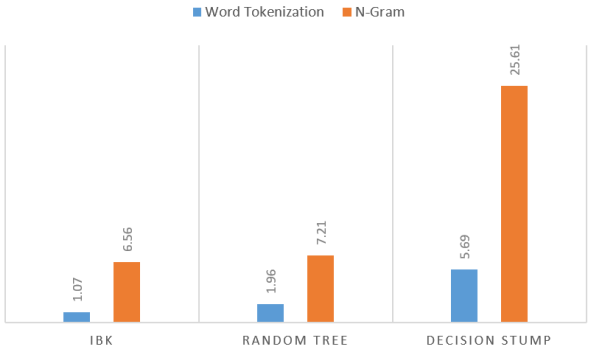

Figure 6. Time (in seconds) taken to build the model using different tokenizers.

It is important to analyze the performance measurement in machine learning. Thus, when it comes to checking or visualizing the performance of a classification model, ROC Curve and AUC can be used to achieve this purpose [42].

The standard ROC curves are presented in Figure 7-8, with the false positive rates on the x-axes and true positive rates on the y-axes. The white area under the curves represents the AUC [43]. The higher the AUC, the better the model is at distinguishing between classes [42]. As of this concept, the AUC is better using N-Gram tokenization over word tokenization.

6. DISCUSSION AND CONCLUSION

In information security, it is important to detect vulnerabilities in web applications. Insecure web applications may cause of stealing personal information (e.g. user cookies) as described earlier in this paper. This paper presented an approach for detecting XSS attacks through text-mining based predictive model. A publicly available dataset is used during the experiment. Experiment results demonstrate that the proposed features can detect vulnerable scripts in web applications with high TPR and low FPR. Experimental results indicate that Decision Stump outperforms the other classifiers used in this experiment, and great results are demonstrated in word tokenization and N-Gram using it. The results also prove that N-Gram tokenizer obtains a higher accuracy compared to word tokenization, although it takes much more time.

A certain limitation was that even though the proposed approach performs well with certain types of machine learning algorithms, it exhausts computer components (e.g. it consumes too much memory and sometimes the program crashes) when using N-Gram with different types of algorithms, and probably without getting results. One way to overcome this issue is to use a large memory size, and the other algorithms are left for future testing. In the future, the same N-Gram tokenizer will be used to test the effect of using higher N (i.e. N>3) with various machine learning algorithms on attack detection results.

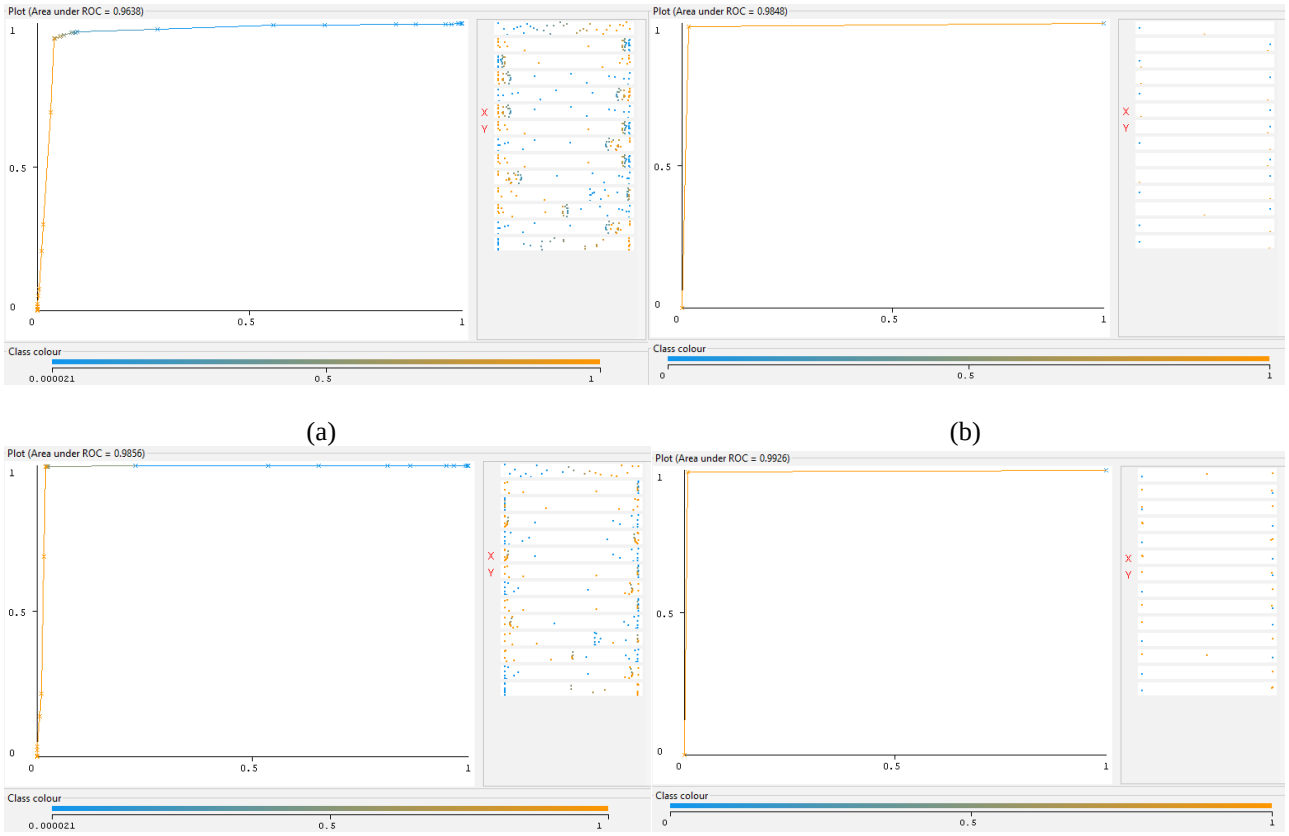

Figure 7. AUC-ROC Curve using Word Tokenization with (a) IBk, (b) Random Tree, and N-Gram with (c) IBk, (d) Random Tree.

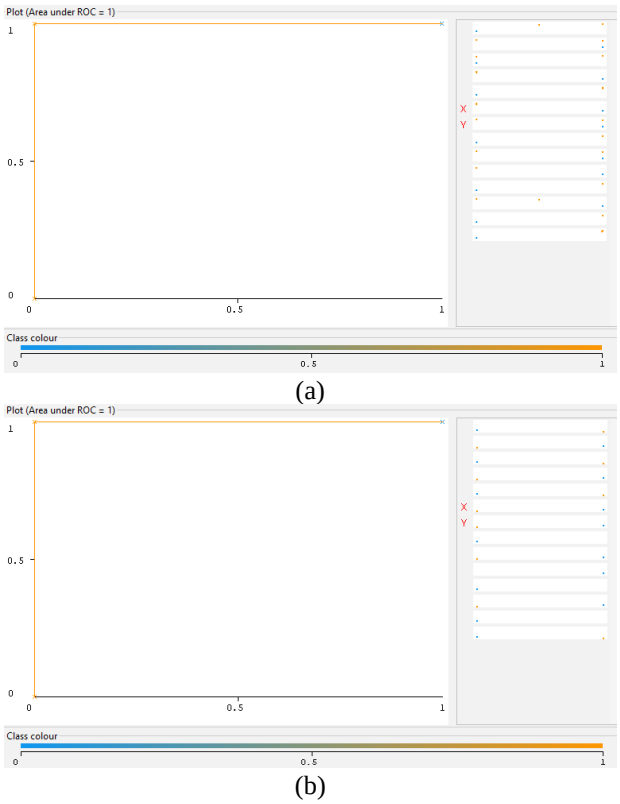

Figure 8. AUC-ROC Curve with Decision Stump in case of using: (a) Word Tokenization, (b) N-Gram

Acknowledgment

First and foremost, praises and thanks to Allah, the Almighty, for His showers of blessings throughout my research work to complete the research successfully.

I am indebted to Prof. Dr. Eyyüp GÜLBANDILAR, professor in computer engineering department at Eskişehir Osmangazi University, and Asst. Prof. İlker ÖZÇELİK, assistant professor in software engineering department at Eskişehir Osmangazi University.

My deep and sincere gratitude for them for giving me the opportunity to do research and providing invaluable guidance throughout this research. They also provided me personal and professional guidance and taught me a great deal about scientific research.

REFERENCES

[1] Ying, M., Li, S. Q. 2016. CSP adoption: current status and future prospects. Security and Communication Networks, 9(17), 4557-4573.

[2] sucuri.net. 2022. Sucuri Security. sitecheck.sucuri.net.

[3] Hearst, M. A. 1999. Untangling text data mining. In Proceedings of the 37th Annual meeting of the

Association for Computational Linguistics (pp. 3-10).

[4] Feldman, R., Dagan, I. 1995. Knowledge Discovery in Textual Databases (KDT). In KDD, Vol. 95, pp. 112-117.

[5] positive technologies. 2022. Threats and Vulnerabilities in Web Applications 2020–2021. www.ptsecurity.com/ww-en/analytics/web-vulnerabilities-2020-2021/

[6] OWASP. 2021. OWASP Top Ten. owasp.org/www-project-top-ten/

[7] Doupe, A., Cui, W., Jakubowski, M. H., Peinado, M., Kruegel, C., Vigna, G. 2013. deDacota: toward preventing server-side XSS via automatic code and data separation. In Proceedings of the 2013 ACM SIGSAC conference on Computer & communications Security, pp. 1205-1216.

[8] Lavrenovs, A., Melón, F. J. R. 2018. HTTP security headers analysis of top one million websites. In 2018 10th International Conference on Cyber Conflict (CyCon) (pp. 345-370). IEEE.

[9] Scholte, T., Robertson, W., Balzarotti, D., Kirda, E. 2012. Preventing input validation vulnerabilities in web applications through automated type analysis. In 2012 IEEE 36th annual computer software and applications conference (pp. 233-243). IEEE.

[10] Stock, B., Lekies, S., Mueller, T., Spiegel, P., Johns, M. 2014. Precise client-side protection against DOM-based cross-site scripting. In 23rd {USENIX} Security Symposium ({USENIX} Security 14) (pp. 655-670).

[11] Salas, M. I. P., Martins, E. 2014. Security testing methodology for vulnerabilities detection of xss in web services and ws-security. Electronic Notes in Theoretical Computer Science, 302, 133-154.

[12] Niakanlahiji, A., Jafarian, J. H. 2017. Webmtd: defeating web code injection attacks using web element attribute mutation. In Proceedings of the 2017 Workshop on Moving Target Defense, pp. 17-26.

[13] Athanasopoulos, E., Pappas, V., Krithinakis, A., Ligouras, S., Markatos, E. P., Karagiannis, T. 2010. xJS: practical XSS prevention for web application development. In Proceedings of the 2010 USENIX conference on Web application development, p. 13.

[14] Van Gundy, M., Chen, H. 2012. Noncespaces: Using randomization to defeat cross-site scripting attacks. Computers & Security, 31(4), 612-628.

[15] Gupta, B. B., Soni, H., Siwan, P., Kumar, A., Gupta, S. 2018. DOM-guard: defeating DOM-based injection of XSS worms in HTML5 web applications on Mobile-based cloud platforms. In Computer and Cyber Security, pp. 425-454. Auerbach Publications.

[16] Pelizzi, R., Sekar, R. 2012. Protection, usability and improvements in reflected XSS filters. In proceedings of the 7th ACM Symposium on Information, Computer and Communications Security, pp. 5-5.

[17] Mitropoulos, D., Stroggylos, K., Spinellis, D., Keromytis, A. D. 2016. How to train your browser: Preventing XSS attacks using contextual script fingerprints. ACM Transactions on Privacy and Security (TOPS), 19(1), 1-31.

[18] Gupta, S., Gupta, B. B., Chaudhary, P. 2018. Hunting for DOM-Based XSS vulnerabilities in mobile cloud-based online social network. Future Generation Computer Systems, 79, 319-336.

[19] Weissbacher, M., Robertson, W., Kirda, E., Kruegel, C., Vigna, G. 2015. Zigzag: Automatically hardening web applications against client-side validation vulnerabilities. In 24th {USENIX} Security Symposium ({USENIX} Security 15), pp. 737-752.

[20] Maurya, S. 2015. Positive security model based server-side solution for prevention of cross-site scripting attacks. In 2015 Annual IEEE India Conference (INDICON), pp. 1-5. IEEE.

[21] Gupta, S., Gupta, B. B. 2016. Enhanced XSS defensive framework for web applications deployed in the virtual machines of cloud computing environment. Procedia Technology, 24, 1595-1602.

[22] Guo, X., Jin, S., Zhang, Y. 2015. XSS vulnerability detection using optimized attack vector repertory. In 2015 International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery, pp. 29-36. IEEE.

[23] Sundareswaran, S., Squicciarini, A. C. 2012. XSS-Dec: A hybrid solution to mitigate cross-site scripting attacks. In Data and Applications Security and Privacy XXVI: 26th Annual IFIP WG 11.3 Conference, DBSec 2012, Paris, France, July 11-13, 2012. Proceedings 26, pp. 223-238. Springer Berlin Heidelberg.

[24] Panja, B., Gennarelli, T., Meharia, P. 2015. Handling cross site scripting attacks using cache check to reduce webpage rendering time with elimination of sanitization and filtering in light weight mobile web browser. In 2015 First Conference on Mobile and Secure Services (MOBISECSERV), pp. 1-7. IEEE.

[25] Goswami, S., Hoque, N., Bhattacharyya, D. K., Kalita, J. 2017. An Unsupervised Method for Detection of XSS Attack. Int. J. Netw. Secur., 19(5), 761-775.

[26] Yusof, I., Pathan, A. S. K. 2016. Mitigating cross-site scripting attacks with a content security policy. Computer, 49(3), 56-63.

[27] Gupta, S., Gupta, B.B. 2017. Cross-Site Scripting (XSS) attacks and defense mechanisms: classification and state-of-the-art. International

Journal of System Assurance Engineering and Management, 8(1), 512-530.

[28] GitHub. 2023. OWASP Community Pages. github.com/OWASP/www-community/blob/master/pages/Types_of_Cross-Site_Scripting.md (Accessed 2 Apr. 2023).

[29] Stivalet, Bertrand. 2022. PHP Vulnerability Test Suite. github.com/stivalet/PHP-Vulnerability-test-suite (Accessed 2 Apr. 2023).

[30] Chetty, N., Vaisla, K. S., Sudarsan, S. D. 2015. Role of attributes selection in classification of Chronic Kidney Disease patients. In 2015 international conference on computing, communication and security (ICCCS), pp. 1-6. IEEE.

[31] Gupta, S., Abraham, S., Sugumaran, V., Amarnath, M. 2016. Fault diagnostics of a gearbox via acoustic signal using wavelet features, J48 Decision Tree and Random Tree Classifier. Indian J Sci Technol, 9, 1-8.

[32] Sammut, C. 2011. Encyclopedia of machine learning : with 78 tables. New York, Ny Springer.

[33] Wikipedia, 2023, Decision stump. en.wikipedia.org/wiki/Decision_stump (Accessed 2 Apr. 2023).

[34] Hvitfeldt, E., Silge J. 2021. Supervised Machine Learning for Text Analysis in R. Smltar.com, smltar.com/. Accessed 19 Jan. 2021.

[35] Cavnar, W. B., Trenkle, J. M. 1994. N-gram-based text categorization. In Proceedings of SDAIR-94, 3rd annual symposium on document analysis and information retrieval (Vol. 161175).

[36] Frank, E., et al. 2016. Data Mining: Practical Machine Learning Tools and Techniques. www.cs.waikato.ac.nz/ml/weka.

[37] Witten, I H, Frank, E. 2005. Data Mining : Practical Machine Learning Tools and Techniques. Amsterdam ; Boston, Ma, Morgan Kaufman.

[38] Tung, Y. H., Tseng, S. S., Shih, J. F., Shan, H. L. 2013. A cost-effective approach to evaluating security vulnerability scanner. In 2013 15th Asia-Pacific Network Operations and Management Symposium (APNOMS), pp. 1-3. IEEE.

[39] Taha, A. A., Hanbury, A. 2015. Metrics for evaluating 3D medical image segmentation: analysis, selection, and tool. BMC medical imaging, 15(1), 1-28.

[40] Sasaki, Y. 2007. The Truth of the F-Measure. 26 Oct. 2007, pp. 1–5.

[41] Iba, W., Langley, P. 1992. Induction of one-level decision trees. In Machine Learning Proceedings 1992 (pp. 233-240). Morgan Kaufmann.

[42] Narkhede, S. 2018. Understanding auc-roc curve. Towards Data Science, 26(1), 220-227.

[43] Muschelli III, J. 2020. ROC and AUC with a binary predictor: a potentially misleading metric. Journal of classification, 37(3), 696-708.