논문 2022-1-13 http://dx.doi.org/10.29056/jsav.2022.06.13

An Analysis of Machine-Learning Feature-Extraction Techniques using Syntactic Tagging for Cross-site Scripting Detection

Nurul Atiqah Abu Talib*, Kyung-Goo Doh*†

Abstract

Working for a strategy to ensure web application security has become more complex as it is not only to protect against the more challenging cross-site scripting (XSS) attacks but also to assist the open expressiveness of web applications to provide users with interactive services. The feature extractions used in supervised machine learning to detect XSS as an approach to the strategy are now in question of their effective classification. Their lack of preserving structural information may not uphold the property of structured data in input payloads. We apply the concept of syntactic n-grams to a payload text representation. The study of different feature extractions on the representation is to see if syntactic information is maintained. Our purpose is to determine the more effective approach to correctly classify benign and malicious payloads on a real-world dataset. The use of sn-grams that produces the most favourable results of accuracy and precision would only indicate that the extraction is reasonably able to minimize false reports during classification.

keywords : Cross-site Scripting, Supervised Machine Learning, Syntactic N-gram Features, Text Classification, Web Security

1. Introduction

Cross-site scripting (XSS) vulnerability in web applications remains a teething problem in web technology [1][2]. The insufficient validation on user inputs can cause vulnerability in the applications, making them susceptible to XSS attacks. These client-based attacks are the results of unwarranted execution in a victim's browser by the injection of malicious scripts into a vulnerable web application. Unfortunately, the validation process to help circumvent attacks has now become more complex. This is because the advanced features of incorporating user-supplied input to expand the usability of web facilities are continuously introduced to web applications, and attackers keep finding new ways of attacking. The legitimate user inputs are often mistaken as malicious, thus, annulling the application's functionality.

Static and dynamic analyses are among the most popular practices to detect or prevent XSS in web applications. They are often used

* Department of Computer Science and Engineering, Hanyang University
† Corresponding Author : Kyung-Goo Doh (email: doh@hanyang.ac.kr)
Submitted: 2022.05.06. Accepted: 2022.06.04. Confirmed: 2022.06.20.

www.kci.go.kr

together with validation procedures, such as XSS filters, to further protect the applications from XSS. Static analysis in which source codes are analyzed without running the application is an approach that requires not only enormous security knowledge to conduct analyses but also a code access which is not always publicly available for security reasons. Whereas, dynamic analysis in which the information is obtained during program execution to detect vulnerabilities relies on the completeness of attack vectors and thus often results in false negatives. In short, what is required from a detection device is not only its comprehensive tasks but also its precise identification of payload classes.

Machine-learning classification techniques are now in growing use in the field of software security, particularly for vulnerability detection, malware analysis, and other web anomalies [3]. The advantages of machine learning include its ability to automatically: (a) categorize data into appropriate categories based on past experience [4]; (b) classify recurring of similar data [5], and (c) predict new data [6]. Having learned that machine learning classification can aid and facilitate the task in other related fields, we are extending the use of similar idea to identifying XSS.

In machine learning, there is a task of selecting features that best represent data, called *feature engineering* [7]. By this task, the selected features are used to transform documents into *feature vectors* through the technique of *feature extraction*.

In the use of machine learning in the detection for XSS, the input *payload* strings can be treated as HTML strings [8] as they are often included in the output to the browser. In HTML, element structure and the order between elements are important. For this reason, it is necessary to use a feature extraction technique that can maintain information about the correct order and relationships of sub-strings.

Among the feature extraction techniques used to maintain the information are (1) vectorization through weighted terms of n-gram features, such as Term Frequency-Inverse Document Frequency (TF-IDF), and (2) word or document embeddings such as Word2Vec and Doc2Vec. It is now a question of whether these current techniques are contributive enough to a good classification model. By this question, we are motivated to carry out a more scrupulous examination of these extraction techniques in their use for classification. To proceed, we apply normalized-tagging n-gram features of a previous work [8] to different feature extraction techniques. Hence, we aim to offer the following contributions by way of:

- developing an alternative feature extraction approach that preserves the order and relations of elements using syntactic n-grams
- applying the use of normalized-tagging n-gram features to different feature extraction techniques
- analyzing classification models based on empirical comparison of different feature extraction techniques

www.kci.go.kr

Table 1. Example of Bag-of-words

Texts	I	Have	An	Orange	Cat	The	Likes	Dog
I have an orange cat	1	1	1	1	1	0	0	0
The cat likes the dog	0	0	0	0	1	2	1	1
The dog likes the cat	0	0	0	0	1	2	1	1

Table 2. Example of Bi-grams

Texts	the cat	cat likes	likes the	the dog	dog likes
the cat likes the dog	1	1	1	1	0
the dog likes the cat	1	0	1	1	1

To present, we divide the discussions of our work into the following sections. Section 2 is to discuss some background concepts. Section 3 outlines our experimental setup. The results of our work are discussed in details in Section 4. In Section 5, we bring in discussions on previous work as a literature review, and Section 6 contains the conclusions.

2. Background

This section provides a brief account of feature extraction techniques that preserve some syntactic information.

2.1 N-grams

N-grams based techniques are used predominantly for extracting features from documents in the domain of natural-language processing and information retrieval. Generally, features to represent texts in a document is a collection of unique n-grams, which are obtained from groups of n adjacent words as they appear in texts. To illustrate, for the text, "*I have an orange cat*", the *1*-gram features that would represent it are: "*I*", "*have*", "*an*", "*orange*" and "*cat*". To add, the *Bag-of-words* (BOW) technique is a representation of texts using 1-gram minus the grammatical, or syntactic, order of texts as it only describes the occurrence of words in a document. For instance, the BOW for different text examples are shown in Table 1.

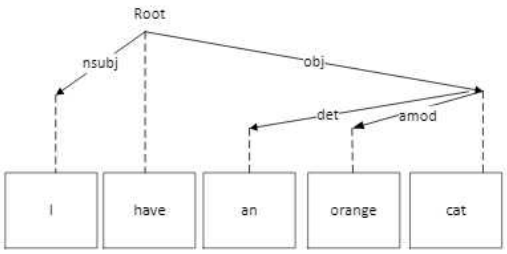
We can see that the BOW disregards the syntactical order of words and produces the same representation in the form of feature vectors for texts containing the same words of different semantics. In order to retain syntactical word order, $n > 1$ would be used. As shown in Table 2, the resulting feature vectors would be different when using the *2*-grams (bi-grams).

2.2 Syntactic N-grams

Syntactic n-grams (sn-grams) is an extension of the n-grams technique introduced by Sidorov et al. [9]. As opposed to sequencing

www.kci.go.kr

elements based on how they appear in texts, like in traditional n-grams, sn-grams are obtained based on the order of elements in syntactic dependencies or trees. Using the first text example in Table 1, its syntactic structure obtained using the Stanford parser[1] is shown in Fig. 1. The resulting syntactic bi-grams are *"have I"*, *"have cat"*, *"cat an"*, and *"cat orange"*.




```
nsubj(have-2, I-1)
root(ROOT-0, have-2)
det(cat-5, an-3)
amod(cat-5, orange-4)
obj(have-2, cat-5)
```

Fig. 1. Example of Syntactic Structure and Dependencies

2.3 Word2Vec and Doc2Vec

Word2vec [10] is a linear-based *word-to-vector* representation of texts that retains semantic meaning of words and their association. The idea is that the distance between words can determine their semantic similarity. It uses a neural network that consists of two learning models: Continuous Bag of Words (CBOW) that predicts a word based on its context, and Skip-gram that predicts the surrounding words of a given word.

An extension of Word2Vec is Doc2Vec [11]. It uses an instance of the Skip-gram model to build representations through the embeddings of arbitrary word sequences, such as sentences, paragraphs, or large documents. By including information for the sequence ID, it can preserve the semantic relationship between sequences.

3. Experimental Setup

Our setup follows three major steps as explained in the ensuing subsections: (1) syntactical structure representation of payloads, (2) dataset and classifier, and (3) training and testing.

3.1 Syntactical Representations of Payloads

This section describes the application of different feature extraction techniques in our experiment.

3.1.1 N-grams

We use the feature approach of a previous work [8] involving the transformation of payloads into a language of features comprising normalized tags. The tags are HTML and URI term *labels*. The transformation is to create a payload representation of feature *sentences*.

1) http://nlp.stanford.edu:8080/parser/index.jsp

www.kci.go.kr

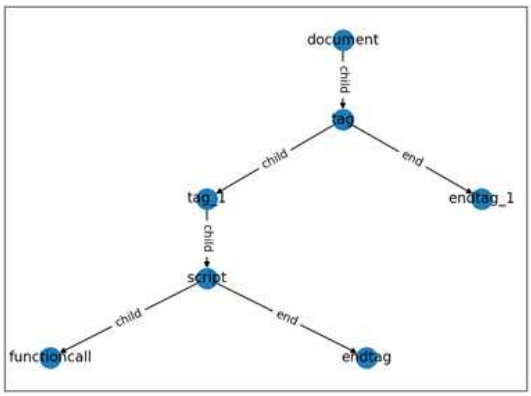


Fig. 2. Example of Syntactic Tree with Relations

The features are n-grams, specifically, bi-grams, a *linear sentence* representation of payloads, that maintain some order between sub-strings, but disregarding their syntactical relations. To further illustrate, for the payload, `<p><svg><script>doEvil()</script></p>`, its sentence is **document tag tag script functioncall endtag endtag**, whereas its features are **document tag**, **tag tag**, **tag script**, **script functioncall**, **functioncall endtag**, **endtag endtag**. The **document** merely suggests the start of the sentence. Finally, we follow the TF-IDF feature extraction approach used in the previous work to convert sentences to vectors. The TF-IDF `ngram_range` parameter is set to `(2,2)` for bi-grams.

3.1.2 Syntactic N-grams

We construct the sn-grams of payloads by extending the approach for the linear representation. To obtain the syntactical relations between labels, we modify the parser used in [8] to create a syntax tree. The relations between nodes in a tree consist of the current node, its succeeding node, and their relationship type (e.g., child, sibling, or end tag).

Fig. 2 shows the syntactic tree of the same example in Section 3.1.1. Each node consists of labels, with an identifier appended to it to retain its uniqueness in the tree. That way it will avoid the creation of two different nodes with the same label. Agian, the **document** node signifies the root of the tree, whereas the edges are the relations between the nodes. Additionally, we use a stack to store parent element of each node so as to maintain the predecessor-successor information. This is because the parser used in the approach does not directly output an abstract syntax tree. The construction of the tree, developed using acyclic-directed graphs of the Python's Networkx[2] package, is done after the parsing process.

Next, we obtain the sentence, so as to later acquire the sn-grams, by iterating the tree only once to collect node labels from each branch. The reading of the nodes begins by first following the left flow before going for the right flow, similar to depth-first traversal (DFT), but maintains the relation between each parent and their child. The resulting *structured sentence* is a sequence of sub-sentences. Specifically, it is a linear sequence of labels with the correct syntactical relations intact. The structured sentence for the payload example in Section 3.1.1 is **document tag tag_1 script functioncall**, **script endtag**, **tag endtag**. We can see that, unlike in the linear sentence, the structured sentence

2) https://networkx.org/documentation/stable/reference/algorithms/dag.html

www.kci.go.kr

preserves the syntactical relation between nodes. The sentence is engineered as a sequence of label sequences rather than a single sequence to avoid the creation of irrelevant sn-grams. We note that the relation labels and the node identifier are included in the above example sentence for clarification purposes, although they are conventionally ignored in usual sentence construction.

To construct the sn-gram features, we use a customized function to obtain a sequence of *n* words in each sub-sentence minus the identifiers. The resulting unique sn-gram (s2-gram) features obtained from the structured sentence is **document tag**, **tag tag**, **tag script**, **script functioncall**, **script endtag**, and **tag endtag**. Finally, similar to the linear approach, we use the TF-IDF vectorizer with the sn-gram features as its vocabulary to convert sentences into feature vectors.

3.1.3 Word2Vec

We use the Word2Vec module from Python's Gensim[3] library. As its purpose is to learn word association, we supply the module with 1-gram labels of the n-gram sentences. We also set the module parameters of `min_count` and `workers` to `1` and `4`, each, and use the default values for other parameters.

3.1.4 Doc2Vec

Similarly, we use the Doc2Vec module from the same library as Word2Vec. As a procedure, we first create tagged documents of linear sentences and their corresponding payload ID and output values. The tagged documents are supplied as input to the module. The parameters are set following those for the Word2Vec module.

3.2 Dataset and Classifier

We utilize the dataset from the work of Fang et al. [12], consisting of 31,407 benign and 33,426 malicious payloads. The target value (i.e., the payload class to be predicted by the classifier) in the dataset are categorical values. To specify, the values of 0 and 1 are each representing the benign and malicious payloads, respectively. Additionally, we use Random Forest[4] as the classification model for our experiment.

3.3 Training and Testing

As a procedure, it involves training and testing a classification model. In the training process, we start with transforming a training dataset containing raw payloads using different payload representation approaches. The resulting sentences from the transformation process are fed to their respective feature extraction approaches (refer Sections 3.1.1 to 3.1.4) in order to develop a vector space (feature vectors). The vectors would then be used as an input to train the classification model. The final output is a trained classifier.

In the testing process, we utilize all the payloads in the testing set and prepare them for

3) https://radimrehurek.com/gensim/index.html

4) https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html

www.kci.go.kr

a similar transformation process as the training set. The classifier will next identify each payload's target value in the set to decide whether the payload's feature vector is either benign or malicious, based on the information it gained from the training phase.

We apply a stratified 10-fold-cross validation[5]. The dataset is divided into 10 folds, nine of which are used as the training set and one as the testing set. Each fold contains approximately the same sizes of each output label. This process is iterated 10 times.

Our experimentation involves comparing the result of different feature extraction approaches using the sentences of normalized labels. The experimentation is conducted using a Mac-Book Pro machine with 8GB of memory and a 2.3 GHz Dual-Core Intel Core i5 processor.

4. Experimental Results

We analyze the results of the study based on the false positive and false negative counts as well as the accuracy, precision, recall, and F1 scoring metrics as follows:

- True positives (*TP*): The frequency of actual malicious payloads that are correctly classified as malicious
- True negatives (*TN*): The frequency of benign payloads that are correctly classified as benign
- False positives (*FP*): The frequency of benign payloads that are incorrectly classified as malicious
- False negatives (*FN*): The frequency of actual malicious payloads that are incorrectly classified as benign
- Accuracy: The ratio between correct detection and the total detection, $(TP+TN)/(TP+FP+FN+TN)$
- Precision: The ratio between the correct detection of malicious payloads and all the predicted malicious payloads, $TP/(TP+FP)$
- Recall: The ratio between the correct detection of malicious payloads and all the actual malicious payloads, $TP/(TP+FN)$

Additionally, we calculate the execution time (Time) for classification, which is measured as the amount of time spent by a classifier to make a decision on the testing set. Time is measured before the start and after the classification procedure, which includes the construction of payload sentences and their vectorization.

Next, we compare the results of alternative feature extraction approaches through experiment using the real-world dataset previously used by other work.

Table 3 shows the comparison results of classification models using four different feature extraction approaches: *N-grams* and TF-IDF, *Sn-grams* and TF-IDF, Word2Vec, and Doc2Vec. We observe that the use of our sn-gram features records the best result in accuracy, and precision, followed by n-grams with only a slight difference of 0.01% and 0.03%, respectively. In terms of recall, n-grams

5) https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.StratifiedKFold.html

www.kci.go.kr

Table 3. Classification results and execution time for each feature extraction approaches

Feature Extraction Approaches	Performances					Time(sec.)
	FP	FN	Accuracy (%)	Precision (%)	Recall (%)	
N-grams and TF-IDF	7	42	99.25	99.80	98.74	1.64
Sn-grams and TF-IDF	6	43	99.26	99.83	98.73	4.86
Word2Vec	14	48	99.05	99.58	98.56	3.58
Doc2Vec	186	214	93.84	94.40	93.61	3.39

Table 4. Payload type of false positives and false negatives reported by n-gram and sn-gram approaches

Payload Type	N-grams		Sn-grams	
	FP	FN	FP	FN
Alphanumeric texts with or without special character	√	√	√	√
Integers with or without special characters		√		√
Query names with special characters		√		√
URI of file path values	√	√	√	√
Semicolon-separated URI queries	√		√	
End tags		√		
Malformed script tags		√		
Function call query values		√		
Tautology statements		√		√
Query names with empty or unclosed tag values		√		√

shows the best with a score of 98.74%, just of 0.01% ahead of normalized sn-grams. Next, Word2Vec stands in third best with 99.1%, 99.6%, and 98.6% score in accuracy, precision, and recall, respectively. Lastly, although not as high as its counterparts, Doc2Vec shows good results for all scores of above 93.0%.

In elaboration, the precision to account for the false positives and false negatives are evident as follows. Using the above formula, the counts are made in reference to the number of detections by classification of each approach. As Table 3 shows our approach of sn-grams come up with the least false positive counts. The n-grams approach is tailing just by a difference of 1 count. Next is Word2Vec come as third with the record of 14 counts. The report of false positives by Doc2Vec is recorded with the highest of 186 counts. As for the record of false negatives, n-grams approach report the least count of 42. Our approach of sn-gram reports 43 exceeding just by 1 count. The Word2Vec approach records 48 counts of false negatives, while Doc2Vec reports the most of 214 counts. To compare, we present in Table 4 the report of different false positives and false negatives by n-gram and sn-gram approaches classified in terms of payload types.

Table 3 also shows the execution time for classification. It is measured as the amount of time spent to translate, vectorize, and classify

www.kci.go.kr

the testing set into the category of malicious or benign payloads. The results show that the n-grams feature approach takes the shortest 1.6 seconds on average to complete an execution. It is followed by the Doc2Vec feature approach of 3.4 seconds, almost twice as long compared to n-gram features. The Word2Vec feature approach comes as the third fastest with more than 3.6 seconds of the running time to complete the execution. The sn-grams takes the longest execution time of 4.9 seconds. However, it is noted that the differences of time score of the approaches are practically negligible, showing the most difference of only about 3 seconds between n-grams and sn-grams.

5. Literature Review

The use of machine learning approach for purpose of network security using classification technique is various. Rathore et al. proposed an XSS attack detection approach for social networking services (SNS) web applications [5]. The approach involves the use of two classifiers, ADTree and Random Forest, to classify whether a web page is XSS-infected or not. The authors compared the performance between the two classifiers and found that the ADTree outperforms Random Forest classifier.

Similarly, Nunan et al. proposed an approach to identify document and URL-based features to classify XSS webpages [4]. The features are based on obfuscated code, suspicious patterns and HTML and JavaScript schemes. Fang et al. proposed a detection approach called DeepXSS via word2vec embedding and Long Short-Term Memory (LSTM) recurrent neural networks [12]. Mereani and Howe attempted the classification of scripts to detect persistent XSS using SVM, K-NN and Random Forests [13]. The features are based on the presence of a single or combined non-alphanumeric characters (structural features) or the presence of certain code (behavioral features) in scripts. These works are the closest to ours as our work also targets to prevent incoming XSS and analyze the different feature extraction approaches to analyze classification performance.

Khabia and Chandak addressed the problem of classifying datasets with high dimensions and proposed a cluster-based approach using n-grams and kmeans [14]. To reduce the loss of semantic information of texts, they created a mapping of two vector-space models. The first vector-space model is based on the feature representations of words, bi-grams and tri-grams. To overcome the high dimensionality problem of feature vectors, they applied a threshold on the TF-IDF values of the vector space. The second vector-space model is based on Latent Semantic Analysis (LSA). They compared the performance of their approach based on the feature representation and found that vectorization based on tri-grams provide the highest accuracy and LSA can help improve the performance of clustering.

Chernis and Verma utilized machine learning to find bugs by extracting two levels of

www.kci.go.kr

features, simple and complex, from C program functions. They further analyzed whether these functions are vulnerable or invulnerable to attacks [15]. Zolanvari et al., on the other hand, discussed how machine learning is used to prevent vulnerability exploitations in Internet of Things (IoT) technologies and to be applied to secure Industrial Internet of Things (IIOT) technologies. They made classification of traffic information into whether it is normal or it is an attack [16].

Kar et al. proposed an approach to detect SQL injection via document similarity measures using TF-IDF and n-grams [17]. They also applied a Hierarchical Agglomerative clustering algorithm to obtain injection patterns. This information is later saved as a database firewall system, called SQLiDDS, to be used for the detection process during run-time. We opt a similar approach for the payload transformation process. However, while they TF-IDF and n-grams to classify SQL queries, we use them to classify input payloads.

6. Conclusion

We present an empirical comparison between different applications of feature extraction techniques: n-grams and TF-IDF, sn-grams and TF-IDF, Word2Vec, and Doc2Vec. It is interesting to note that our sn-gram approach charts the most favorably evident results of accuracy and precision in payloads classification. In terms of recall, the n-gram approach shows the best result but with a negligible record of 0.01% surpassing our approach. Although our approach does not score the best in classification time, we take this time difference as negligible.

Most importantly, both of these approaches are reasonably superior in minimizing false positive and negative reports. Additionally, they are highly potential in usability to correctly classify payloads represented as syntactical-structure-aware documents using a normalized set of features.

This research was supported by the MISP(Ministry of Science, ICT), Korea, under the National Program for Excellence in SW supervised by the IITP(Institute of Information & Communications Technology Planning & Evaluation) (2018-0-00192)

References

[1] OWASP, “OWASP Top 10:2021”, OWASP, 2021. https://owasp.org/Top10/ (accessed Apr. 21, 2022).

[2] MITRE Corporation, “CVE Details: The Ultimate Security Vulnerability Datasource”, https://www.cvedetails.com/vulnerabilities-by-types.php (accessed Mar. 20, 2019).

[3] G. Apruzzese, M. Colajanni, L. Ferretti, A. Guido, and M. Marchetti, “On the effectiveness of machine and deep learning for cyber security”, Int. Conf. Cyber Conflict, CYCON, vol. 2018-May, pp. 371-389, 2018, doi: 10.23919/CYCON.2018.8405026.

[4] A. E. Nunan, E. Souto, E. M. Dos Santos, and E. Feitosa, “Automatic classification of cross-site scripting in web pages using

www.kci.go.kr

document-based and URL-based features", Proc. - IEEE Symp. Comput. Commun., pp. 000702-000707, 2012, doi: 10.1109/ISCC.2012.6249380.

[5] S. Rathore, P. K. Sharma, and J. H. Park, "XSSClassifier: An efficient XSS attack detection approach based on machine learning classifier on SNSs", J. Inf. Process. Syst., vol. 13, no. 4, pp. 1014-1028, 2017, doi: 10.3745/JIPS.03.0079.

[6] S. Ndichu, S. Kim, S. Ozawa, T. Misu, and K. Makishima, "A machine learning approach to detection of JavaScript-based attacks using AST features and paragraph vectors", Appl. Soft Comput. J., vol. 84, 2019, doi: 10.1016/j.asoc.2019.105721.

[7] A. C. Müller and S. Guido, Introduction to Machine Learning with Python: a guide for data scientists, vol. 53, no. 9. 2016.

[8] N. A. Abu Talib and K. G. Doh, "Run-time Detection of Cross-site Scripting: A Machine-Learning Approach Using Syntactic-Tagging N-Gram Features", in-press.

[9] G. Sidorov, F. Velasquez, E. Stamatatos, A. Gelbukh, and L. Chanona-Hernández, "Syntactic N-grams as machine learning features for natural language processing", Expert Syst. Appl., vol. 41, no. 3, pp. 853-860, 2014, doi: 10.1016/j.eswa.2013.08.015.

[10] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space", 1st Int. Conf. Learn. Represent. ICLR 2013 - Work. Track Proc., 2013.

[11] Q. Le and T. Mikolov, "Distributed representations of sentences and documents", 31st Int. Conf. Mach. Learn. ICML 2014, vol. 4, pp. 2931-2939, 2014.

[12] Y. Fang, Y. Li, L. Liu, and C. Huang, "DeepXSS: Cross site scripting detection based on deep learning", ACM Int. Conf. Proceeding Ser., pp. 47-51, 2018, doi: 10.1145/3194452.3194469.

[13] F. A. Mereani and J. M. Howe, "Detecting Cross-Site Scripting Attacks Using Machine Learning", Adv. Intell. Syst. Comput., vol. 723, pp. 200-210, 2018, doi: 10.1007/978-3-319-74690-6_20.

[14] A. Khabia and M. B. Chandak, "A Cluster based Approach with N-grams at Word Level for Document Classification", Int. J. Comput. Appl., vol. 117, no. 23, pp. 38-42, 2015, doi: 10.5120/20697-3599.

[15] B. Chernis and R. Verma, "Machine learning methods for software vulnerability detection", IWSPA 2018 - Proc. 4th ACM Int. Work. Secur. Priv. Anal. Co-located with CODASPY 2018, vol. 2018-Janua, pp. 31-39, 2018, doi: 10.1145/3180445.3180453.

[16] M. Zolanvari, M. A. Teixeira, L. Gupta, K. M. Khan, and R. Jain, "Machine Learning-Based Network Vulnerability Analysis of Industrial Internet of Things", IEEE Internet Things J., vol. 6, no. 4, pp. 6822-6834, 2019, doi: 10.1109/JIOT.2019.2912022.

[17] D. Kar, S. Panigrahi, and S. Sundararajan, "Sqlidds: SQL injection detection using query transformation and document similarity", Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics), vol. 8956, pp. 377-390, 2015, doi: 10.1007/978-3-319-14977-6_41.

www.kci.go.kr

Authors

Nurul Atiqah Abu Talib

2009-2013 BIT (Hons) in Computer System Security from Universiti Kuala Lumpur, Malaysian Institute of Information Technology (MIIT), Malaysia

2013.9-present Ph.D candidate in the Department of Computer Science and Engineering at Hanyang University ERICA, Gyeonggi-do, South Korea

<Research interests> Web Security, Machine Learning, Program Analysis

Kyung-Goo Doh

1980 B.S. degree in Industrial Engineering, Hanyang University

1987 M.S. degree in Computer Science, Iowa State University

1992 Ph.D. degree in Computer Science, Kansas State University

1993-1995 Assistant Professor, University of Aizu, Japan

1995-present Professor, Department of Computer Science, Hanyang University ERICA

<Research interests> Programming Languages, Program Analysis, Software Engineering, Software Security

www.kci.go.kr