WAF Signature Generation from Real-Time Information on the Web using Similarity to CVE

1st Masahito Kumazaki
Graduate School of Informatics
Nagoya University
Nagoya, Japan
Email: kumazaki@net.itc.nagoya-u.ac.jp

2nd Yukiko Yamaguchi
Information Technology Center
Nagoya University
Nagoya, Japan
Email: yamaguchi@itc.nagoya-u.ac.jp

3rd Hajime Shimada
Information Technology Center
Nagoya University
Nagoya, Japan
Email: shimada@itc.nagoya-u.ac.jp

4th Hirokazu Hasegawa
Information Security Office
Nagoya University
Nagoya, Japan
Email: hasegawa@icts.nagoya-u.ac.jp

***Abstract*—Zero-day attacks and attacks based on publicly disclosed vulnerability information are major threats to network security. To cope with such attacks, it is important to collect related information and deal with vulnerabilities as soon as possible. We have developed a system that collects vulnerability information related to web applications from real-time open information on the web, such as that found on Twitter and other discussion-style web sites, and generates web application firewall (WAF) signatures for them. In this study, first, we collected vulnerability information containing a specified keyword from the National Vulnerability Database (NVD) data feed and generated WAF signatures automatically. Then, we examined the suitability of the WAF signature generation from one tweet. Finally, we extracted tweets that might contain vulnerability information and labeled them using a filtering algorithm. We then further experimented on gathering and extracting vulnerability information for a target web application. First, we gathered past Common Vulnerabilities and Exposures (CVE) descriptions of the target web application and used them to generate a Doc2Vec model and word vectors. We also used the trained Doc2Vec model to generate word vectors gathered from open information sources such as Twitter, Stack Overflow, and Security StackExchange. After that, we extracted vulnerability information for the target web application by calculating the cosine similarity between the word vectors of the open information and the CVE descriptions. Experimental results demonstrated that our Doc2Vec-based extraction process can be easily adapted to individual web applications.**

Keywords–Web Application Firewall(WAF); Zero-day Attack; Vulnerability Information; Real-time Information.

I. INTRODUCTION

We first review the results of the vulnerability information filtering method based on pattern matching that we introduced in our previous work presented in SECURWARE 2020 [1].

Web applications are recognized as an important part of the social infrastructure, and the average person uses a variety of such applications every day. At the same time, cyberattacks are increasing year by year and are now widely recognized as a significant threat to the social infrastructure. There are many cases of serious damage caused by such attacks, such as classified information leakage by unauthorized access and attacks that exploit vulnerabilities [2] [3].

Among cyberattacks, attacks that exploit published vulnerabilities are particularly on the increase [4] [5]. In a recent case, the number of detected attacks on Apache Struts 2 increased immediately after the announcement of its vulnerability [6], some of which resulted in personal information leakage due to a lack of necessary countermeasures such as timely system updates [7] [8]. Another major information security issue is zero-day attacks, which occur before the release of vulnerability information or the provision of patches [20]. Google Chrome suffered such a zero-day attack in 2019 [10].

The generally recommended countermeasure against such cyberattacks is to apply fixed patches distributed by the vendor in a timely manner. However, in the case of zero-day attacks, there is an unprotected period before the patch is released.

We previously developed a web application firewall (WAF) signature generation system using real-time information on the web to mitigate zero-day attack problems [1]. Real-time information sources such as Twitter are used for a variety of purposes [11] [12], and they may contain the latest vulnerabilities and emergency plans, which are useful for mitigating the problem before any formal vulnerability information or patches are released. Our system automatically collects vulnerability information to construct WAF signatures to mitigate the problems until the release of formal countermeasures. Although there are previous studies on the automatic generation of signatures for intrusion detection systems (IDSs) [13] [14] [15], we focus on WAF in this study because our target is web applications. To acquire the latest vulnerability information from the web, our system collects vulnerability information from real-time data feeds on social networking services (SNSs), web forums for security technologies discussion, and similar sites. After cleansing the collected data, the system checks for associated vulnerable web applications and generates WAF signatures for them.

In this paper, we demonstrate the effectiveness of our proposal on a practical level as an extension of the basic proof of concept provided in our previous work [1]. We first discuss the results of the vulnerability information filtering method based on pattern matching for multiple information sources. Further, as a method for generating an extraction model for individual web applications, we introduce a method using the similarity of word vectors generated by a Doc2Vec model trained with past Common Vulnerabilities and Exposures (CVE) descriptions.

To evaluate the WAF signature generation from the vulnerability information part, we performed an experiment with vulnerability information provided by the National Vulnerability Database (NVD) [16] and a specific tweet. First, we extracted the vulnerability information including a specified keyword related to the target web application and performed automatic generation of the WAF signature, as a proof of

concept. Then, we collected vulnerability information from Twitter, a well-known SNS and real-time information source, and attempted to generate WAF signatures from the tweets. Finally, we examined the extraction of tweets that included vulnerability information of the target web application by means of a pattern matching approach. The results of these investigations demonstrated the efficiency of the proposed system.

We next performed a practical experiment in which vulnerability information of the target web application was extracted using a text processing framework. First, we gathered past vulnerability information of the target web application from the description section of a CVE as reference data. We used these reference data to train a Doc2Vec model to extract vulnerability information words of the target web application and then utilized the trained model to generate reference word vectors with text data from the CVE description. Then, we collected vulnerability information from open information such as SNS (e.g., Twitter) and web forums for discussion on security technologies (e.g., Stack Overflow). We generated word vectors from the text part of the open information gathered with the trained Doc2Vec model and treated them as open information vectors. Finally, we performed similarity calculations between the reference vectors and the open information vectors. If an open information vector had a large similarity to the reference vectors, we treated the text correlated to the open information vector as containing vulnerability-related information.

In section II of this paper, we describe the background of our study, namely, the existing countermeasures. In section III, we present our proposed system and the architectures with which it is implemented. In section IV, we describe the experiments we performed to evaluate the implemented system. In section V, we discuss additional real-time information sources. Section VI presents the pattern matching-based extraction for target web applications and section VII shows the word vector similarity-based extraction with the Doc2Vec model trained using past CVE descriptions. We conclude in section VIII with a brief summary.

II. BACKGROUND

Existing countermeasures against zero-day attacks include defense-in-depth solutions that combine multiple security appliances such as firewalls, IDS/intrusion prevention system (IPS)-based allow/deny lists, and so on. However, if they are based on static rules, these countermeasures may result in an unprotected period against attacks. To mitigate the damage caused by zero-day attacks, we previously proposed a WAF signature generation system that uses real-time information on the web. Specifically, the proposed system generates a WAF signature for blocking access to a vulnerable web application when it discovers from real-time information on the Internet that the application contains vulnerability information. As a result, the unprotected time against attacks is shortened and the damage will ideally be mitigated.

A. WAF (Web Application Firewall)

Web applications are becoming more complicated year by year, and as such it is getting harder to detect vulnerabilities in their implementations. Therefore, WAFs are often used as a security measure to protect web applications from ingress traffic and to mitigate attacks that exploit vulnerabilities [17]. A WAF can be installed in multiple locations, such as a host type installed on a web server, a network type installed on a communication path to the web server, or a cloud type using WAF services on a cloud provided by the cloud service provider. By setting rules to prevent attacks aimed at typical web application vulnerabilities, we can protect the web server from attacks that target a specific vulnerability. Some WAFs enable users to set their own rules for specific attacks, which makes it possible to prevent zero-day attacks by applying custom signatures. In general, a WAF use the following basic functions to prevent external attacks and notify the administrator.

Analyzing
: Analyze Hypertext Transfer Protocol (HTTP) communication based on a detection pattern defined in an allow/deny list.

Processing
: Perform pass-through processing, error processing, replacement processing, blocking processing, and so on. Judgement is based on the result of the analysis function.

Logging
: Record WAF activity. An audit log records any detected unauthorized HTTP communication and its processing method. An operation log records WAF operation information and error information.

B. ModSecurity

ModSecurity is an open-source host type WAF software provided by Trustwave. It has the following functions.

- Recording and auditing of whole HTTP traffic
- Real-time monitoring of HTTP traffic
- Flexible enough rule engine to act as an external patch for web applications
- Can be embedded as a module of web server software Apache, IIS, and NGINX

Also, the Open web application Security Project (OWASP) [18] provides a Core Rule Set (CRS) that includes signatures for typical cyberattacks for ModSecurity. In the experiments later in this study, we use ModSecurity as the WAF for our proposed system to take advantage of the flexibility of the rule engine and the versatility of the module.

C. Doc2Vec

Doc2Vec is an advanced implementation of the Paragraph Vector proposed by Mikolov et al. [19]. It can vectorize a variety of different-length texts without losing the word order or semantics. However, with some text it is not preferable to keep the word order, so in the current Doc2Vec framework the user can choose either a mode that does not keep the word order, called the distributed memory model of paragraph vector (PV-DM), or a mode that does keep the word order, called the distributed bag-of-words paragraph vector (PV-DBOW). Before vectorizing text to prepare it for processing, we can train the Doc2Vec model with training samples. This is one of the advantages of Doc2Vec compared to prior neural network approaches for word embedding.

One of the applications for Doc2Vec is a similarity calculation between documents. With this application, by applying

TABLE I. Real-time information sources

	API	Identification	Timestamp
Twitter [20]	✓	✓	✓
Stack Overflow [21]	✓	✓	✓
Reddit [22]	✓	✓	✓
teratail [23]	×	✓	✓
Security StackExchange [24]	✓	✓	✓

a vector similarity calculation method (e.g., cosine distance) to vectors obtained from documents, we can obtain a similarity value between documents.

III. Proposed system

The proposed system consists of the following four modules:

(1) Collection module
(2) Cleansing module
(3) Signature generation module
(4) Notification generation module.

Figure 1 shows the architecture of the proposed system and its data flow.

First, the collection module collects vulnerability information from real-time information sources such as SNS (Fig. 1 (1)). After that, the proposed system performs data cleansing on the collected data (Fig. 1 (2)). Finally, the proposed system generates WAF signatures and set them (Fig. 1 (3)). At the same time, the proposed system generates a notification file for the administrator (Fig. 1 (4)).

In this system, it is assumed that the administrator registers the names of the web applications and their version information into the system beforehand. It is on the basis of this registered information that the system extracts vulnerability information from the Internet.

A. Collection module

The proposed system generates WAF signatures from the real-time information sources shown in Table I. All these sources are equipped with IDs and timestamps. The collection module collects vulnerability information from the sources and saves their ID, timestamp, and body. Subsequent processes will require these IDs and timestamps to identify the articles and the date of publication.

B. Cleansing module

The proposed system performs data cleansing on collected data to remove duplicate information and get the necessary information. The cleansing module extracts the following attributes from text and web page. The system uses the following attributes for generating WAF signatures.

- Application name
- Vulnerability type
- Version information
- Vulnerability identification information such as a Common Vulnerabilities and Exposures (CVE)-ID

C. Signature generation module

If the web application name that is used for operating web application system is included among the attributes extracted by the cleansing module, the system generates a WAF signature to block HTTP requests to that application and applies it to the WAF. It also notifies the notification generation module that the signature has been generated.

D. Notification generation module

The proposed system generates a notification and sends it to the administrator. This notification includes information such as the name and version of the vulnerable web application. We expect that when the vulnerability is resolved (e.g., applying a patch), this notification will remind the administrator to remove the signature.

IV. Initial Experiments

We conducted the following initial evaluation experiments. For the target web application to collect vulnerability information, we used WordPress as a keyword for extracting vulnerability information due to its known history of many vulnerabilities.

A. Experiment 1: WAF signature generation using CVE information

First, as a proof of concept, we implemented the proposed system shown in Figure 1 with Python 3.6.8 and AWK scripts and examined the automatic generation of WAF signatures using NVD data feeds instead of real-time information.

1) Processing Method: The collection module (Fig. 1 (1)) obtained information from the NVD data feed, which contains CVEs, on a daily basis. The cleansing module (Fig .1 (2)) checked whether the keyword was included in the Common Platform Enumeration (CPE) name for each vulnerability information on the data feed. The following data were then extracted.

1) CVE-ID
2) CPE name
   The CPE name is a name that identifies the platform [25].

   ```
   cpe:2.3:[Part]:[Vendor]:[Product]:[Version]
   :[Update]:[Edition]:[SW_Edition]:[Target_SW]
   :[Target_HW]:[Language]:[Other]
   ```

   In this experiment, we used [Part] and [Product] from the CPE name. [Part] represents a product type with one character, where 'a' is an application, 'o' is an operating system, and 'h' is hardware. [Product] is the product name.
3) Version information

We extracted these data and stored them into JavaScript Object Notation (JSON) format.

Next, the signature generation module (Fig. 1 (3)), we generated a signature for ModSecurity from the extracted information to prevent access to the web application. With reference to `ModSecurity_41_xss_attacks.conf` and `ModSecurity_41_sqlinjection_attacks.conf` from ModSecurity's CRS, we added the following signatures.

- **VARIABLES:** REQUEST_COOKIES | !REQUEST_COOKIES:/__utm/ | REQUEST_COOKIES_NAMES | ARGS_NAMES | ARGS | XML:/

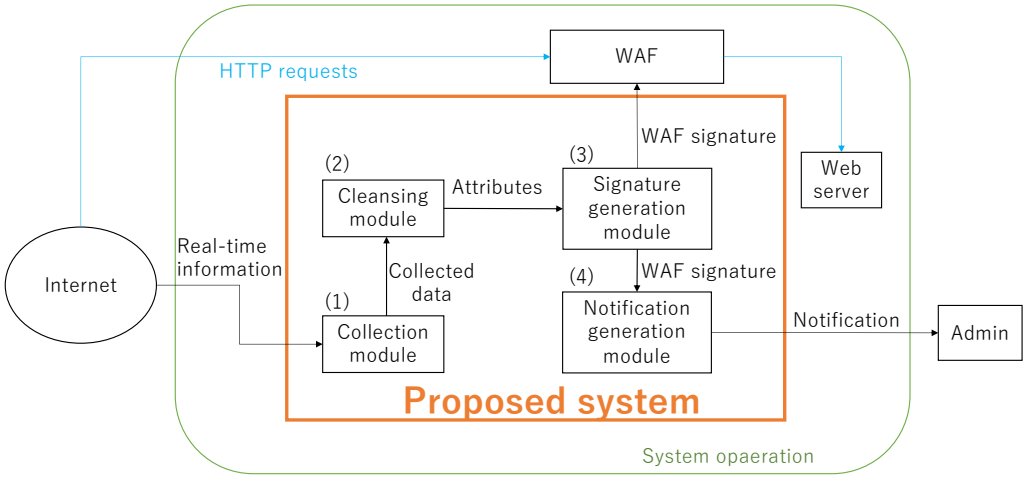

Figure 1. System architecture and data flow

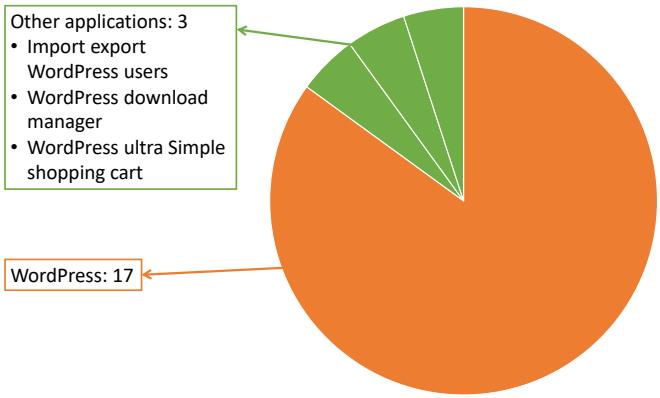

Figure 2. Extracted information in Experiment 1

```
1 SecRule REQUEST_COOKIES|!REQUEST:/__utm/|
    REQUEST_COOKIES_NAMES|ARGS_NAMES|ARGS|XML:/* "
    import_export_wordpress_users" "phase:2,block,msg:'
    WordPress injection.'severity:'2',id:'15001'"
2 SecRule REQUEST_COOKIES|!REQUEST:/__utm/|
    REQUEST_COOKIES_NAMES|ARGS_NAMES|ARGS|XML:/* "
    wordpress" "phase:2,block,msg:'WordPress injection.'
    severity:'2',id:'15002'"
3 SecRule REQUEST_COOKIES|!REQUEST:/__utm/|
    REQUEST_COOKIES_NAMES|ARGS_NAMES|ARGS|XML:/* "
    wordpress_download_manager" "phase:2,block,msg:'
    WordPress injection.'severity:'2',id:'15003'"
4 SecRule REQUEST_COOKIES|!REQUEST:/__utm/|
    REQUEST_COOKIES_NAMES|ARGS_NAMES|ARGS|XML:/* "
    wordpress_ultra_simple_paypal_shopping_cart" "phase:2,
    block,msg:'WordPress injection.'severity:'2',id
    :'15004'"
```

Figure 3. Generated WAF signature in Experiment 1

- **OPERATOR:** Regular expression using web application names
- **ACTIONS:** phase:2,block,msg:'*application name* injection.',severity:'2',id:'15000+line number'

In addition, we generated text files to provide notification about the signature generation in the notification generation module (Fig.1 (4)).

2) Results: We performed this process once daily for ten days from December 21, 2019 to December 30, 2019.

Since the results for all days during the collection period were the same, we present the result of a single day as an example in Figure 2.

The results of the automatically generated WAF signatures are shown in Figure 3. The signature for WordPress was generated from 17 cases of WordPress vulnerability information and other signatures were generated from one case corresponding to each application.

3) Consideration: The generated signatures show that, in addition to the signature for the actual WordPress application, other applications that include "wordpress" in their name have been blocked as well. These redundant rules place an additional burden on the WAF. To overcome this limitation, we need to improve the signature generation rule and fine-tune it.

TABLE II. Extracted attributes of the tweets

Key	Type	Description
id	Int64	tweet_id as an integer.
created_at	String	The time when the tweet was created.
username	String	User name who posted the tweet.
text	String	The tweet contents in UTF-8 format.
urls	List	Uniform Resource Locators (URLs) in the tweet[1].

B. Experiment 2: Generation of WAF signatures using twitter feed

Next, to investigate the possibility of WAF signature generation from real-time vulnerability information, we collected tweets from twitter feeds, chose one tweet, and generated a WAF signature from it. The tweets were collected using an Application Programming Interface (API) search by setting the query parameters to "wordpress". The chosen tweet is shown in Figure 4. The proposed system uses the information from tweets listed in Table II from tweets, so we extracted the following information.

- **id:** 1200259525707796482

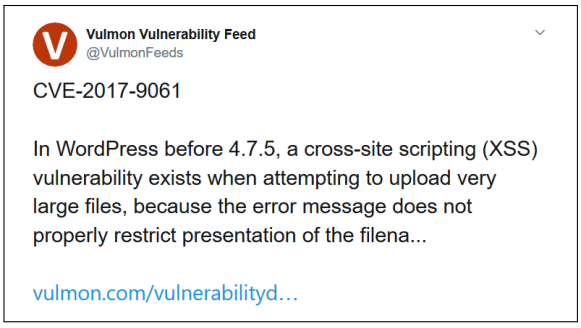

Figure 4. The tweet used in Experiment 2

```
SecRule REQUEST_COOKIES|!REQUEST:/__utm/|
    REQUEST_COOKIES_NAMES|ARGS_NAMES|ARGS|XML:/* "
    wordpress" "phase:2,brock,msg:'WordPress XSS.'severity
    :'2',id:'15001'"
```

Figure 5. Generated WAF signature in Experiment 2

- **created_at:** 2019-11-29 03:45:45
- **username:** Vulmon Vulnerability Feed
- **text:** CVE-2017-9061\n\nIn WordPress before 4.7.5, a cross-site scripting (XSS) vulnerability exists when attempting to upload very large files, because the error message does not properly restrict presentation of the filena...\n\nhttps://t.co/anaw5-QSAoa"
- **urls:** [http://vulmon.com/vulnerabilitydetails?qid=CVE-2017-9061]

1) Method and results: A visual check of the web application and vulnerability information contained in the tweet resolved the following attributes from the text.

- **Application name:** WordPress
- **Vulnerability type:** XSS
- **Version information:** 4.7.5 and earlier
- **CVE-ID:** CVE-2017-9061

We also checked the web page indicated by the URL but could not obtain any additional attributes. As in experiment 1, we generated a ModSecurity signature from these attributes, which is shown in Figure 5.

2) Consideration: This experiment confirms that a WAF signature could be generated from collected tweets. However, we still need a method to select the relevant tweet. We expect to be able to extract information such as the web application name, its version, or the type of vulnerability through the pattern matching approach.

V. Consideration of Additional Sources

Currently, the real-time information source is only Twitter, and it could be prone to be disinformation. Therefore, we explored the possibility of using other information sources, which are listed in Table I.

To analyze whether these information sources are appropriate sources or not, we reviewed discussions about the following three WordPress vulnerabilities in the corresponding communities. These vulnerabilities are registered in the NVD and have a high Common Vulnerability Scoring System (CVSS) score.

TABLE III. Number of search hits

	2018-20148	2019-17669	2019-20041
Stack Overflow	10(-)	6(-)	10(-)
Reddit	2(-)	15(-)	11(-)
teratail	6(-)	6(-)	3(-)
Security StackExchange	2(-)	0(-)	2(-)

(): Number of search hits related CVEs

TABLE IV. Number of search hits in each knowledge community

Knowledge community\Year	2017	2018	2019	total
Stack Overflow	2,166	2,072	1,837	6,075
Reddit[cybersecurity]	31	172	266	469
Reddit[security]	16	60	151	227
teratail	241	196	172	609
Security StackExchange	1,166	1,072	768	4,006

- CVE-2018-20148 Published: December 14, 2018
- CVE-2019-17669 Published: October 17, 2019
- CVE-2019-20041 Published: December 27, 2019

Since we cannot collect information from teratail via API, we searched the above vulnerabilities by means of a Google search with the queries "WordPress" and "vulnerability". The duration of the search was set to one month before and after the vulnerability announcement. The results are shown in Table III, the numbers in parentheses refer to the number of search hits. These results show that we could not obtain any information about these vulnerabilities from these communities in a timely manner.

In addition, the current system has some problems. As the results shown in Table III are not suitable for a comparison of each knowledge community, we tried additional exploration. Specifically, we compared the number of search hits per site using the Custom Search API provided by Google to see the number of discussions about vulnerabilities in each knowledge community. We set the query "vulnerability" for all sites and set the period from January 1, 2017 to December 31, 2019. Since Reddit has a lot of topics that are not security-related, we only explored two subreddits ("security" and "cybersecurity").

The results are shown in Table IV. Among the knowledge communities examined in this study, Stack Overflow and Security StackExchange had the most active in discussions about the vulnerability, so we conclude they can be used as a good information source.

VI. Filtering and Extracting Tweets through Pattern Matching Approach

A. Description

In big data processing, we generally gather data from information sources through the API of the services or perform web scraping. However, data acquired in such a manner typically contain a lot of totally unrelated information, which means we have to perform a related information extraction or an unrelated information elimination. To circumvent this issue, we propose a filtering method based on regular expression. The method first confirms the existence of specific information such as application name and version number with regular expression and then decides whether the information is related information

or not. To determine the effectiveness of this method, we performed an experiment with gathered tweets.

B. Experiment 3-1: Regular expression-based extraction with tweet

First, on the basis of the results of Experiment 2, we extracted and filtered vulnerability information from the tweets. We then used the twitter API to collect tweets every day for the following period. After eliminating the duplicates, we further processed 1,116 tweets.

- **Collection period:** From December 4, 2019 to January 8, 2020
- **Search queries:** "WordPress AND Vulnerability", "WordPress AND XSS", and "WordPress AND injection"

To check the performance of the filter, we manually assigned the following labels to the tweets based on the relevance to the WordPress vulnerability.

0: WordPress vulnerability information
1: Other information

As a result of the manual labeling, 76 tweets regarding WordPress vulnerabilities were identified. The remaining 1,040 tweets were mistakenly extracted since there URLs referred to, e.g., a web page created using WordPress.

1) Method: We collected the attributes shown in Table II from the tweet and filtered them on the basis of its text context by pattern matching using regular expressions from the tweet's text and web page body indicated by the URL. The following attributes were extracted from the tweet and stored in JSON format.

- **ID:** tweet-id
- **App_name:** If the string includes "wordpress" this element is 1, otherwise 0.
- **CVE:** Extract the string "CVE-\d{4}-\d+" and store it as a list.
- **plugin / theme:** If the string includes "plugin" or "theme" this parameter is 1, otherwise 0.

From these attributes and a list of CVEs corresponding to the target application, we automatically assigned an estimated label for tweets in accordance with the flowchart shown in Figure 6. Each label corresponds to the following categorization result.

0: WordPress vulnerability information
1: Other information
1-a: Does not included the string "wordpress"
1-b: Expected to be a WordPress plugin or theme
1-c: No WordPress vulnerability in CVE list
2: Unfiltered

2) Results: We implemented the filter in Python as per the flowchart shown in Figure 6 and ran it on the 1,116 collected tweets. The results are shown in Table V. As we can see, our method filtered 597 tweets, 98.5% of which were filtered correctly. However, 519 were unfiltered. By using the pattern matching approach, the number of objects to analyze could be reduced by half.

Out of the seven tweets that were not correctly filtered, six were supposed to be labeled as 0 but were mistakenly labeled as 1-b. These were all weekly summaries of vulnerability information for WordPress and related modules and contained information about both WordPress and its plugins. This is why the filter misjudged them as information about WordPress plugins.

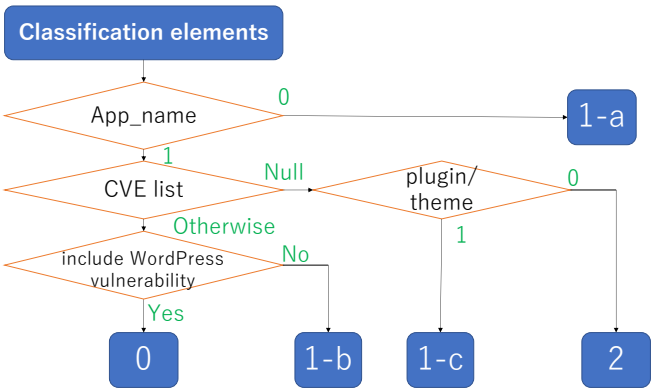

Figure 6. Flowchart of estimated label setting in Experiments 3-1 and 3-2

TABLE V. Filtering results of Experiment 3-1

		Estimated label					
		0	1-a	1-b	1-c	2	total
Correct label	0	13	0	6	0	57	76
	1	1	94	292	191	462	1,040
	total	14	94	298	191	519	1,116

3) Consideration: The information required for the proposed system is vulnerability information that is up-to-date or has not been officially announced, which is included in the tweets labeled as 2 in this experiment. Therefore, we need to provide a way to extract the required information from these tweets.

C. Experiment 3-2: Regular expression-based extraction with other data

To determine the difference among data collection periods and among data sources, we applied the regular expression-based filtering procedure discussed in Section VI-B to the other data. Table VI shows data sources and collection periods.

We performed regular expression-based filtering on the main text of tweet and web page content pointed to by the URL contained in the tweet. We also performed filtering on question texts for Stack Overflow and Security StackExchange data. We manually applied labels to Security StackExchange data and compared these estimated labels provided by the flow in Figure 6.

Table VII shows the filtering results with estimated labels. The label descriptions are the same as those in Section VI-B.

TABLE VI. Source and collection period of Experiment 3-2

Source site	Collection period	Number
Tweet-2021	from February 19, 2021 to June 30, 2021	5,843
Tweet-2020	from March 2, 2020 to June 2, 2020	4,702
Stack Overflow	from January 1, 2018 to December 31, 2020	48,377
Security StackExchange	from January 1, 2018 to December 31, 2020	151

TABLE VII. Filtering results of Experiment 3-2

	0	1-a	1-b	1-c	2	total
Tweet-2021	224	572	1,165	1,276	2,606	5,843
Tweet-2020	33	565	1,615	601	1,888	4,702
Stack OverFrow	1	17,328	3,843	0	27,165	48,337
Security StackExchange	1	8	12	1	129	151

When we compared Tweet-2020 and Tweet-2021, we found that we could filter and classify more than 60% of the data regardless of the data gathering period. Tweet-2021 contained seven times more WordPress vulnerability results (label 0) than Tweet-2020. This difference comes from the fact that a danger vulnerability CVE-2020-36326 (CVSS score: 9.8) was announced in the Tweet-2021 gathering period, which affected the number of WordPress vulnerability results.

We could filter and classify around 45% of the data from Stack Overflow, and there was only one WordPress vulnerability result. We speculate that community members of Stack Overflow who have an interest in vulnerability may also be members of Security StackExchange and asked about it on the Security StackExchange side.

We could only filter and classify around 15% of the data from Security StackExchange. This is presumably because discussions on Security StackExchange are complicated or higher level ones, such as "How can I operate WordPress to avoid security issues as much as possible?" and such discussions do not contain specific version numbers or specific plugins, which are required in the Figure 6 flow.

Table VII shows the relationship between manually applied correct labels and estimated labels. As mentioned, only 15% were classified, so we cannot include the remaining 85% in this discussion. Eight CVEs were issued in the data gathering period, but we only obtained one WordPress vulnerability result (label 0). However, the version numbers listed in the results were not a WordPress versions (i.e., they belonged to the other application), so the results lacked version number information. We clarify why these results were poor through a detailed analysis of the Security StackExchange text data. First, the texts here contained a lot more information than a tweet, which means that one text data frequently contained vulnerability discussions on multiple applications. Furthermore, we found that many text data did not contain key phrases or version numbers compared to tweets. On the other hand, we found that some text contained detailed information such as "problem of illegal communication occurrence" and appeal reminders about various problems. Such information might sometimes include the first report of a zero-day vulnerability, so it may be applicable to the WAF rule for zero-day vulnerability if we can treat such information properly.

To conclude of this subsection, our findings indicate that the filtering method we propose works well for tweets but less well for Security StackExchange, which features longer texts. We also found that the characteristics of the text data in Security StackExchange differ significantly from those of tweets, which may stem from the differing styles of the knowledge communities.

TABLE VIII. Correct/estimated labels of Security StackExchange in Experiment 3-2

		estimated_label					
		0	1-a	1-b	1-c	2	total
correct_label	0	1	0	2	0	5	8
	1	0	8	10	1	124	143
	total	1	8	12	1	129	151

TABLE IX. Learning sources of Experiments 4-1 and 4-2

Data form	Collection period	No.
CVE_WordPress	From August 17, 2005 to June 15, 2021	564
CVE_ALL1	From January 1, 2018 to December 31, 2018	16,511

VII. Narrowing down using Doc2Vec

A. Description

In this section, to automate the target web application vulnerability extraction, we propose a Doc2Vec-based method that trains a Doc2Vec model with past vulnerability descriptions and extracts vulnerability information from the latest open information (e.g., tweets). We use the description sections of CVEs as past vulnerability information sources. We first gathered past CVE descriptions of the target web application and used them to generate the Doc2Vec model and word vectors. We also generated word vectors from large amounts of open information gathered with the trained Doc2Vec model. We then calculated the cosine similarity between individual word vectors of open information and individual word vectors of CVE descriptions. If a word vector of an open information has a large similarity to word vectors of the past CVE vectors of a target web application, we consider the open information to contain vulnerability information of that application. To eliminate the vulnerability information of other web applications, we also created a Doc2Vec model from entire CVE descriptions and related word vectors. The final decision is made on the basis of both similarities to the CVE description of the target web application (high similarity) and similarities to the whole CVE descriptions (low similarity).

B. Experimental setup

We performed an experiment to investigate the relationship between the estimated label in Section VI and the similarity obtained by the Doc2Vec-based method. Table IX lists the data we used for training the Doc2Vec models. We gathered CVEs of the target application (WordPress) that were announced from August 17, 2005 to June 15, 2021 (CVE WordPress). We also gathered all of the CVEs that were announced in 2018 (CVE ALL). We collected the above CVEs using NVD's REST API [25].

We trained and created two Doc2Vec models: The one is WP_R_model, which uses the description section of individual CVE_WordPress data, and ALL_R_model, which uses the description section of individual CVE_ALL data. The Doc2Vec parameters used for training are itemized below. A parameter

TABLE X. Verification data for Experiments 4-1 and 4-2

Data form	Correction period	No.
Tweet	From December 4, 2019 to January 8, 2020	1,116
Security StackExchange	From January 1, 2018 to December 31, 2020	151

value existing at the top is a default value and values existing in the cases are varied during evaluations.

- DM: 1 (learn with PV-DM)
- Learning Rate: 0.025 (0.0012, 0.0025)
- Vector Size: 300 (30, 100, 450, 500, 600, 800, 1000)
- Min_count: 10 (1, 2, 3, 5, 8, 15)

We performed word vector generation for the description section of individual CVE_ALL data with the ALL_R_model and treat it as a summary of the description of individual CVE_ALL data. This word vector set is named V_ALL. Similarly, We performed word vector generation for the description section of individual CVE_WordPress data with the WP_R_model and treat it as a summary of the description of individual CVE_WordPress data. This word vector set is named V_WP.

As test data, we decided to use manually labeled tweet data and Security StackExchange data. We did not used Stack Overflow data because it contains quite a few vulnerability discussion entries (as described in Section VI-C). Table X lists the test data. We performed word vector generation on individual tweet data for both Doc2Vec models and treat them as a summaries of the individual tweet data. These word vector sets are named V_TW_WP (generated with WP_R_model) and V_TW_ALL (generated with ALL_R_model). Similarly, we performed word vector generation for individual Security StackExchange data for both Doc2Vec models and treat them as a summaries of the individual Security StackExchange data. These word vector sets are named as V_SSE_WP (generated with WP_R_model) and V_SEE_ALL (generated with ALL_R_model).

Then, we calculated Similarity_App and Similarity_All values for individual tweet data with the following two steps. First, we chose a tweet data and calculated the cosine distance between the V_TW_WP of the tweet data and an individual V_WP. Second, we calculated the average of the top-5 cosine distance values and used it as the Similarity_App value for that tweet data. This procedure was performed for all tweet data to obtain their Similarity_App values. The same procedure was also performed with V_TW_ALL and V_ALL to obtain Similarity_All values for all tweet data. Similarity_App and Similarity_All for Security Stack Exchange data were generated using the same procedure with V_WP, V_ALL, V_SSE_WP, and V_SSE_ALL.

Finally, we set the threshold value for both Similarity_App and Similarity_All with the heuristic method and calculated the below evaluation index, which is widely used in classification experiment.

- True Positive (TP)
- False Positive (FP)
- True Negative (TN)
- False Negative (FN)
- Accuracy (Acc)
- Precision (Pre)
- True Positive Rate (TPR)
- False Positive Rate (FPR)
- True Negative Rate (TNR)
- False Negative Rate (FNR)

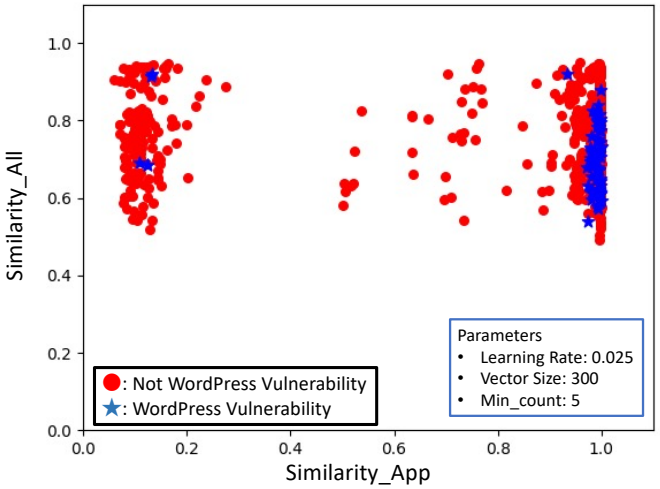

Figure 7. Experriment 4-1: Scatter plot of similarity between tweets and CVEs

C. Experiment 4-1: Results with tweets

Table XI shows the evaluation indices of tweet data with the default Doc2Vec parameters and the parameters with top 2 accuracy. Tweet data contains a comparatively small number words, so it adopts a smaller Vector Size value and smaller Min_count value. The best parameter in this experiment was Vector Size = 300 and Min_count = 5 with Similarity_App $\geq$ 0.95 and Similarity_All $<$ 0.8 threshold values. The result did not change even in the short Vector Size value, which means we can reduce the calculation cost by urilizing the shorter Vector Size value (e.g., Vector Size = 30). When we reduced the Min_count to less than 5, the result became dramatically worse. This result suggests that too small Min_count will include words that are unnecessary for the Doc2Vec model and thereby lead to bad results. The learning rate parameters affected quite a few of the results in this manner, so we had to omit them.

Figure 7 shows the distribution of Similarity_App and Similarity_All with the WordPress Vulnerability or Not WordPress Vulnerability label in the best parameter. The horizontal axis denotes Similarity_App and the vertical axis denotes Similarity_All. The blue star and red circle indicate that a tweet data does or does not contain a WordPress vulnerability, respectively. We can see that a large number of WordPress Vulnerability data are gathered near the Similarity_App = 1.0 area. However, a significant amount of Not WordPress Vulnerability data is also gathered there, which indicates a high number of false positive results. Similarity_All can eliminate some of the Not WordPress Vulnerability data. However, since V_ALL is generated from all of the CVEs, and as such contains many security and vulnerability related words, WordPress data has a comparatively large similarity to V_ALL values, which reduces the separation ability with Similarity_All.

D. Experiment 4-2: Results with Security StackExchange

Table XII shows the evaluation indices of Security Stack Exchange data with the default Doc2Vec parameters and parameters with top-2 accuracy. Security StackExchange data contains a comparatively large number words, so it adopts

TABLE XI. Metrics of Experiment 4-1: Difference by Vector_Size and Min_count.

	TP	FP	TN	FN		Acc	Pre	TPR	TNR	FPR	FNR
Vector: 300, Count: 10	66	722	317	11		34.3%	85.7%	8.4%	96.6%	3.4%	91.6%
Vector: 300, Count: 5	68	614	425	9		44.2%	88.3%	10.0%	97.9%	2.1%	90.0%
Vector: 30, Count: 5	70	637	402	7		42.3%	90.9%	9.9%	98.3%	1.7%	90.1%

TABLE XII. Metrics of Experiment 4-2: Difference by Vector_Size.

	TP	FP	TN	FN		Acc	Pre	TPR	TNR	FPR	FNR
Vector: 300, Count: 10	5	44	99	3		68.9%	62.4%	10.2%	97.1%	2.9%	89.8%
Vector: 450, Count: 10	5	32	111	3		76.8%	62.5%	13.5%	97.3%	2.6%	86.5%
Vector: 600, Count: 10	6	57	86	2		60.9%	75.0%	9.5%	97.7%	2.3%	90.5%

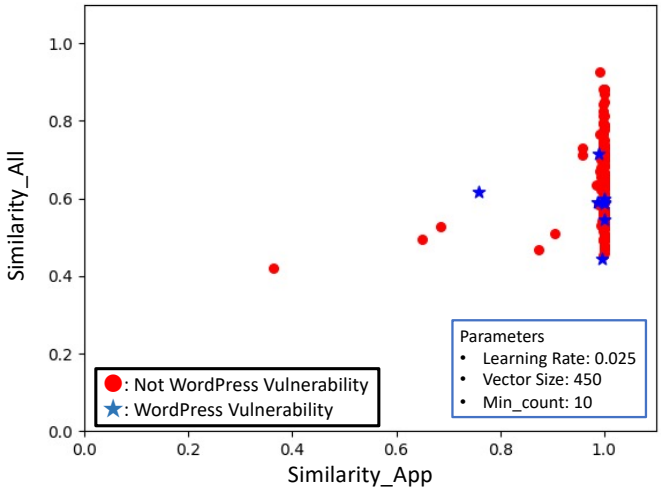

Figure 8. Experriment 4-2: Scatter plot of similarity between SSE and CVEs

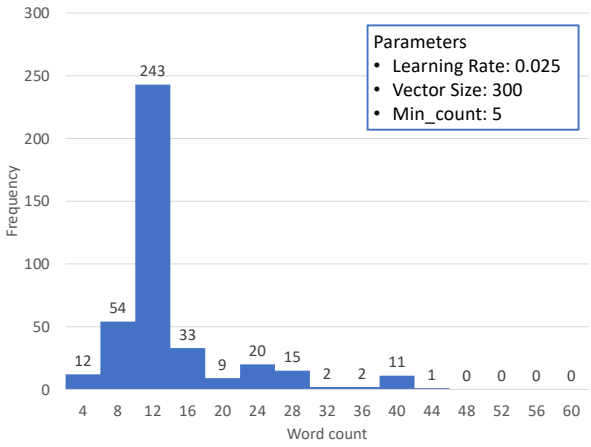

Figure 9. Experriment 4-1: Histgram of true negatives

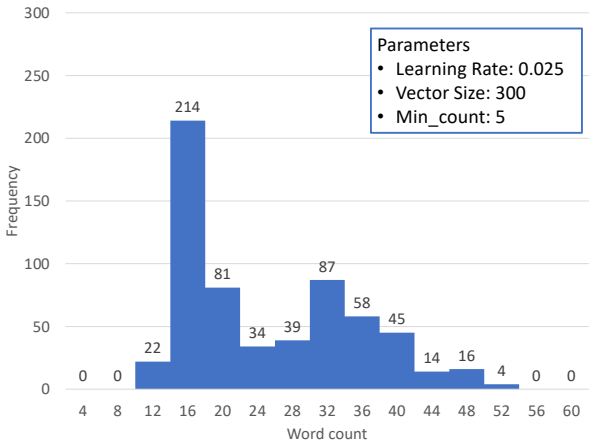

Figure 10. Experriment 4-1: Histgram of false positives

a larger Vector Size value. The best parameter in this experiment was Vector Size = 450 and Min_count = 10 with Similarity_App $\geq$ 0.98 and 0.54 $<$ Similarity_All $<$ 0.64 threshold values. We conclude that the default Vector Size is not sufficient to represent a large number words. When we reduced Min_count to less than 10, the result became worse.

Figure 8 shows the distribution of Similarity_App and Similarity_All with WordPress Vulnerability or Not WordPress Vulnerability labels in the best parameter. The organization of the figure is identical to Figure 7. Similar to the results for the tweet data, a lot of the Not WordPress Vulnerability data here is gathered near the Similarity_App = 1.0 area, which indicates a high number of false positive results. This trend was stronger than it was in the tweet data, as indicated by almost all of the Not WordPress Vulnerability data in Figure 8 being distributed close to Similarity_App = 0.1. However, the Similarity_All values of WordPress Vulnerability data were comparatively small, so we can omit the Not WordPress Vulnerability Data by reducing the Similarity_All threshold value compared to the tweet data result. This characteristics is connected to the higher accuracy result than that of the tweet data.

E. Relationship between negatives and word count in individual tweet data

As discussed in Section VII-B, since Doc2Vec-based classification could not classify negatives, it had a higher number of false positive results, especially for tweet data. To analyze why this occurred, we examined the relationship between negatives and word count in individual tweet data. Our hunch was that, since tweet data cobtain fewer words, they might have been comparatively harder to separate.

Figures 9 and 10 show the relationship between word count and true negatives and false positives, respectively. The horizontal axis denotes word count in bin size 4 and the vertical axis denotes the frequency of an individual bin.

As we can see in Figure 9, true negatives appeared more frequently in comparatively lower word count areas (less than 16). In contrast, in Figure 10, we can see that false positives were concentrated more on big word count areas (more than 16). These findings demonstrate that,contrary to our expection, the proposed method is quite capable of separating small word count data. If we can improve the classification result of the higher word count area, the current false positive rate will be mitigated. There are many approaches we could take for making such an improvement, such as using a different model with higher word count areas (e.g. utilizing a more complicated text classification model).

F. Summary of Doc2Vec results

The regular expression-based classification discussed in Section VI cannot classify large amounts of data, especially in Security StackExchange data. In contrast, we found here that Doc2Vec-based classification can perform classification on whole data with moderate performance. Furthermore, it can omit negatives with high accuracy in both datasets. Thus, we recommend combining both regular expression-based and Doc2Vec-based classifications with the following rules.

- Apply the Doc2Vec-based classification in the first stage and omit data classified as negative.
- Apply the regular expression-based classification to data classified as positive in the first stage.

VIII. CONCLUSION

In this paper, we proposed a WAF signature generation system using real-time open information (e.g., SNS) to provide early protection for web applications. To extract vulnerability information from real-time open information, we developed a regular expression-based filtering method and a Doc2Vec-based similarity calculation method evaluated them with Twitter and Security Stack Exchange data.

With the regular expression-based filtering method, we could classify 60% of the tweet data as to whether it contained information about vulnerability of the target web application. However, we could only classify 10% of the Security StackExchange data, as these data were comparatively long and sometimes contained multiple vulnerabilities of several different applications.

With the Doc2Vec-based method, we could classify both tweet data and Security StackExchange data with moderate accuracy, especially when it came to extracting true positive data (i.e., data that actually discussed the vulnerability of the target web application). However, there were still a number of false positive results, and it requireed further classification of data rated as positive.

On the basis of these findings, we proposed a sequential use of both methods. In the first stage, we perform the Doc2Vec-based filtering is applied to omit data rated as negative, as those data contain almost no actually positive data. In second stage, regular expression-based filtering is applied to perform detailed classification for extracting actually positive data.

We also examined the relationship between word count and classification accuracy in data rated as negative in the Doc2Vec-based method. We found that the proposed method could classify small word count areas well enough but struggled with larger word count areas. It should be possible to significantly reduce the number of false positives if we use a more complicated text classification algorithm or advancement of future text classification algorithm.

With the advancement of text classification algorithms expected in the future, our proposed method shows good potential for generating WAF signatures and will be beneficial for protecting web applications from attacks following official vulnerability notifications, especially in case of zero-day vulnerability.

REFERENCES

[1] M. Kumazaki, Y. Yamaguchi, H. Shimada, H. Hasegawa, "WAF Signature Generation with Real-Time Information on the Web," In Proceedings of the 14th International Conference on Emerging Security Information, Systems and Technologies (SECURWARE 2020), ISBN: 978-1-61208-821-1, pp. 40-45, November 2020.

[2] Significant Cyber Incidents — Center for Strategic and International Studies https://www.csis.org/programs/technology-policy-program/significant-cyber-incidents [retrieved: July, 2020]

[3] The 15 biggest data breaches of the 21st century | CSO Online https://www.csoonline.com/article/2130877/the-biggest-data-breaches-of-the-21st-century.html [retrieved: August, 2020]

[4] 10 Major Security Threats 2019 — Information-Technology Promotion Agency (IPA), Japan https://www.ipa.go.jp/files/000076989.pdf [retrieved: July, 2020]

[5] Think Fast: Time Between Disclosure, Patch Release and Vulnerability Exploitation — Intelligence for Vulnerability Management, Part Two | FireEye Inc https://www.fireeye.com/blog/threat-research/2020/04/time-between-disclosure-patch-release-and-vulnerability-exploitation.html [retrieved: August, 2020]

[6] Attacks Heating Up Against Apache Struts 2 Vulnerability | Threatpost https://threatpost.com/attacks-heating-up-against-apache-struts-2-vulnerability/124183/ [retrieved: July, 2020]

[7] US House of Representatives Committee on Oversight and Government Reform, "The Equifax Data Breach", Majority Staff Report 115th Congress, 2018. https://republicans-oversight.house.gov/wp-content/uploads/2018/12/Equifax-Report.pdf [retrieved: July, 2020]

[8] GMO Payment Gateway, "Apology and Report for Leak of Personal InformationDueto Unauthorized Access", 2017. https://www.gmo-pg.com/en/corp/newsroom/pdf/170310_gmo_pg_en.pdf [retrieved: July, 2020]

[9] Committee on National Security Systems, "Committee on National Security Systems(CNSS) Glossary", CNSSI No. 4009, 2015

[10] Chrome Releases: Stable Channel Update for Desktop https://chromereleases.googleblog.com/2019/03/stable-channel-update-for-desktop.html [retrieved: July, 2020]

[11] T. Sakaki, O. Makoto, and M. Yutaka, "Earthquake shakes Twitter users: real-time event detection by social sensors.", In Proceedings of the 19th international conference on World wide web, pp. 851-860, 2010.

[12] O. Oh, M. Agrawal, and H. R. Rao, "Information control and terrorism:Tracking the mumbai terrorist attack through twitter.", Information-Systems Frontiers, vol. 13, no. 1, pp. 33 – 43, 2011.

[13] S. More, M. Matthews, A. Joshi, and T. Finin, "A Knowledge-Based Approach To Intrusion Detection Modeling.", In Proceedings of 2012 IEEE Symposium on Security and Privacy Workshops, pp. 75-81, May 2012.

[14] C. Kreibich and J. Crowcroft, "Honeycomb: Creating Intrusion Detection Signatures Using Honeypots.", SIGCOMM Computer Communication Review., Vol. 34, No. 1, pp. 51-56, January 2004.

[15] S. Singh, C. Estan, G. Varghese, and S. Savage, "Automated Worm Fingerprinting.", In Proceedings of the 6th Symposium on Operating Systems Design and Implementation, pp.4-4, December 2004.

[16] NVD - Home https://nvd.nist.gov/ [retrieved: October, 2020]

[17] K. Lawrence and L. Luo, “Interactive management of web application firewall rules.”, U.S.Patent, No. 9,473,457, 2016

[18] OWASP Foundation | Open Source Foundation for Application Security https://owasp.org/ [retrieved: October, 2020]

[19] T. Mikolov, K. Chen, G. Corrado, and J. Dean, “Efficient Estimation of Word Representations in Vector Space,” arXiv:1301.3781, January 2013.

[20] Twitter https://twitter.com [retrieved: October, 2020]

[21] Stack Overflow - Where Developers Learn, Share, & Build Careers https://stackoverflow.com [retrieved: October, 2020]

[22] reddit: the front page of the internet https://www.teratail.com [retrieved: October, 2020]

[23] teratail teratail.com [retrieved: October, 2020]

[24] Information Security Stack Exchange https://security.stackexchange.com/ [retrieved: October, 2020]

[25] B. Cheikes, D. Waltermire, and K. Scarfone, “Common platform enumeration: Naming specification version 2.3”, NIST Interagency Report 7695, pp.11-13, 2011.

[26] NVD - New NVD CVE CPE API and SOAP Retirement https://nvd.nist.gov/General/News/New-NVD-CVE-CPE-API-and-SOAP-Retirement