

Artificial Intelligence and Dynamic Analysis-Based Web Application Vulnerability Scanner

Mehmet Ali Yalçınkaya^{1,*} and Ecir Uğur Küçükşille²

¹Computer Engineering Department, Kırşehir Ahi Evran University, Kırşehir, Turkey

²Computer Engineering Department, Süleyman Demirel University, Isparta, Turkey

ARTICLE INFO.

Article history:

Received: October 30, 2022

Revised: July 25, 2023

Accepted: August 7, 2023

Published Online: November 20, 2023

Keywords:

Data Mining, Machine Learning,
Web Application Penetration Tests,
Web Application Vulnerabilities

Type: Research Article

doi: 10.22042/isecure.2023.
367746.847

doi: 20.1001.1.20082045.2024.
16.1.4.8

ABSTRACT

The widespread use of web applications and running on sensitive data has made them one of the most significant targets of cyber attackers. One of the most crucial security measures that can be taken is detecting and closing vulnerabilities on web applications before attackers. This study developed a web application vulnerability scanner based on dynamic analysis and artificial intelligence, which could test web applications using GET and POST methods and had test classes for 21 different vulnerability types. The developed vulnerability scanner was tested on a web application test laboratory, created within this study's scope and had 262 different web applications. A data set was created from the tests performed using the developed vulnerability scanner. In this study, web page classification was made using the mentioned data set as a first stage. The highest success rate in the page classification process was determined by 95.39% using the Random Forest Algorithm. The second operation performed using the dataset was the association analysis between vulnerabilities. The proposed model saved 21% more time than the standard scanning model. The page classification process was also used in crawling the web application in this study.

© 2024 ISC. All rights reserved.

1 Introduction

Nowadays, users can perform many operations, such as shopping, research, file transfer, and communication with social media using web applications. Such widespread use of web applications and the processing of very sensitive data using these applications have made them the number one target of cyber attackers.

Security breach reports informed by various organizations emphasize the significance of ensuring the

security of web applications. According to Verizon's research report, 73% of security incidents in 2022 were due to attacks on web applications. When the report is examined, most attacks after web applications have been carried out to users' emails using phishing. Nevertheless, this attack rate is about 35%. When these results are examined, it is seen that the targeting rate of web applications is almost twice that of emails. [1]. According to the 2022 report published by the ITRC, data of 422,143,312 users were stolen due to attacks on various websites last year. The reason for such high rates of attacks is various security vulnerabilities in the developed web applications [2].

One of the methods used to identify security vulnerabilities in a web application is the dynamic analysis

* Corresponding author.

Email addresses: mehmetyalcinkaya@ahievran.edu.tr,
ecirkucuksille@sdu.edu.tr

ISSN: 2008-2045 © 2024 ISC. All rights reserved.

method. In the dynamic analysis method, harmful inputs are sent to a web application that is actively serving, the application's response is analyzed, and security vulnerabilities are identified according to the responses obtained [3]. Since detecting vulnerabilities on a web application with the dynamic analysis method takes a very long time, and there is the possibility of overlooking the pages and points that should be tested, automated web application vulnerability scanners have gained significant importance.

In this study, a dynamic analysis and artificial intelligence-based web application vulnerability scanner named Sansar was developed. Sansar performs a classification process in the web application crawling and testing the vulnerabilities. The following chronological order was followed in the development of the Sansar. The first version of Sansar, which had crawling and vulnerability scanning modules and did not have artificial intelligence, was developed. While collecting URL addresses from the page HTML codes, the crawling module of first Sansar filled in the form and input fields on the web pages with default values and enabled the page to pass to the next stage. The mentioned tool also could test 21 different vulnerabilities, including vulnerabilities such as SQL injection, XSS, and LFI.

Then, to test the adequacy of Sansar, 250 different web applications developed in ASP and PHP programming languages were installed on both the local and remote servers. With the establishment of different types of web applications, such as shopping, news, sports, school automation, hospital automation, and library automation, which have been put into use by their developers on repositories such as Github, Gitlab, and Sourceforge, the broadest and most diverse web application penetration testing laboratory in the literature has been established. The Sansar tool also included a feature extraction module that allowed recording various data on the web page, form, and input where the vulnerability was tested. With this module, various features belonging to security vulnerabilities detected in web applications in the laboratory mentioned in the previous paragraph were recorded, and a data set was created. To our knowledge, the data set obtained from the data collected within the scope of this study is again the first in the literature on subjects such as features the data set had, the number of samples, and the variety of vulnerabilities.

Using the data set obtained in this study, firstly, the page classification process was performed according to the data obtained. The highest accuracy rate was obtained from the Random Forest Algorithm, with 95.39% in the tagging classification of page types.

Page classification was first used in the crawler module. After this step, the page classification function was added to Sansar. Page classification was first used in the crawler module. During crawling, Sansar classified the pages it visits. If the page is a "login page," it uses the valid username and password information and successfully performs the login process. This feature created a unique value that none of the existing web application vulnerability scanners had. Page classification was also used when associating the detected vulnerabilities with the page types was initiated. This stage corresponded to another original part of this study. The association rules obtained using the Apriori algorithm on the dataset enabled the establishment of an association between page types and vulnerability types. Thanks to this developed technique, the page type in question was determined when Sansar performed a vulnerability test on a page. Then, the tests that must be carried out 'primarily' on the related page were conducted according to the association rules.

The contributions of the presented study can be summarized as follows:

- Sansar includes a comprehensive set of 21 different vulnerability testing modules.
- To our knowledge, this study established the largest web application security laboratory documented in the literature.
- The uniqueness of the laboratory used for the tests, along with the original vulnerability scanner, allowed for the creating of a distinctive dataset from the conducted tests. This original dataset was utilized to establish relationships between vulnerabilities and classify page types through labelling.
- Also, association analysis was applied for the first time to determine the relationships between web application vulnerabilities.
- Additionally, for the first time, this study conducted the process of directly detecting login pages in web applications without any input from the user.
- The findings indicate that the newly developed model offers a 21% speed improvement compared to the standard scanning model.

In the second section of this study, a summary of the related studies in the literature is presented. In the third section of this study, web application security, web application penetration tests, the random forest algorithm, and the Apriori algorithm, which form the basis of the present study, were addressed. The fourth section discusses the modules belonging to the developed web application vulnerability scanner, the performed page classification, and association analysis

processes. In the fifth section, the evaluation of this study and the tool developed was performed by comparing them with previous studies in the literature.

2 Related Work

When we review the studies in the field of web application vulnerability scanners in the literature, it has been seen that the studies have mostly focused on detecting specific vulnerabilities. The main vulnerability scanners in the literature and the types of vulnerabilities they can detect are shown in Table 1 below.

When the page classification processes in the literature are examined, it is observed that web pages have been classified into seven different categories, such as art, technology, engineering, and health, in a study conducted in 2017 [15]. As mentioned earlier, seven different classification algorithms were tested in the study, and the highest 92% accuracy rate was achieved. Another study on page classification was conducted in 2015 [16]. In this study, web pages were classified using words and word sets in HTML tags, and an 80% accuracy rate on average was achieved. In the study conducted in 2012 [17], the classification process was performed using the SVM, KNN, and GIS algorithms on the data set formed from pages collected from a news site. The highest success rate was obtained at 84% in the study in question. In Klassen's [18] study, search forms were classified using text data in HTML tags. A 91% success rate was achieved in the classification process performed using artificial neural networks. Rui and Horovitz used an artificial neural network to classify HTML forms and achieved a 94.7% success rate [19].

Sansar's contribution to the literature: It uses artificial intelligence in vulnerability tests and crawling steps. When Sansar visits a page for testing or crawling, it first classifies the page and then acts according to the classification result. It is the only known vulnerability scanner in the literature with this function.

3 Materials And Methods

This section examined web application vulnerabilities, the dynamic analysis method, the Random Forest Algorithm, and the Apriori Algorithm, which formed this study's outlines.

3.1 Web Application Security

Nowadays, web applications consist of HTML content that runs in a static or dynamic structure. In static web applications, any queries are not executed in the database according to the inputs received from the user. These web applications are generally used for promotion and information delivery, and the content

of which does not change according to user preferences [20].

Dynamic web applications are web applications whose content and behaviour change according to user requests. In dynamic web applications, the inputs received from users are interpreted by the web server and added to the queries run on the database server. The user-oriented flexible operation of dynamic web applications also brings about many security vulnerabilities. Nowadays, the vast majority of web application vulnerabilities arise because the input statements given to the web application by the user do not pass through any control or supervision. A list of the most critical vulnerabilities in web applications is published by OWASP [21]. The web application vulnerability scanner developed within the scope of this study tests 21 different web vulnerabilities dynamically using the GET and POST methods. These vulnerabilities are as follows:

- SQL Injection (Error-Based, Time-Based, HTTP Header-Based)
- Cross-site scripting (XSS)
- HTML injection
- Bash command injection (Shellshock)
- LDAP injection
- PHP code injection
- OS command injection
- File injection vulnerabilities (LFI, RFI)
- XPATH injection
- SSI injection
- Server-side template injection
- Directory traversal (for Linux and Windows Systems)
- CRLF injection
- Buffer overflow
- Open redirect
- XML External Entity (XEE)
- Cross-Site Tracing (XST)

3.2 Web Application Penetration Tests

Detecting vulnerabilities in a web application was carried out by two methods: static and dynamic analysis. In the static analysis, also known as the white box analysis method, the source code of the web application is examined, and all program branchings are tried to detect possible security vulnerabilities. However, the disadvantage of static code analysis is that it remains insufficient in determining security vulnerabilities that can only be detected during the execution of the application. Besides, since the software development process is a cycle, the static analysis method remains insufficient in this cycle.

Another method used to detect vulnerabilities in web applications is dynamic analysis, also known as

Table 1. The current studies on web application vulnerability scanners

Studies	The Name of Developed Tool	Vulnerabilities it can Detect
Kals <i>et al.</i> [4]	Secubat	SQL Injection (SQLi), Cross Site Scripting (XSS)
Kosuga <i>et al.</i> [5]	SANIA	SQL Injection (SQLi)
Halfond [6]	SDAPT	SQL Injection (SQLi)
Artunes and Vieira [7]	-	SQL Injection (SQLi)
Chen and Wu [8]	-	SQL Injection (SQLi), Cross Site Scripting (XSS)
Galan <i>et al.</i> [9]	-	Cross Site Scripting (XSS)
Ali <i>et al.</i> [10]	MySQLInjector	SQL Injection (SQLi)
Singh and Roy [11]	NVS	SQL Injection (SQLi)
Djuric [12]	SQLIVDT	SQL Injection (SQLi)
Lee <i>et al.</i> [13]	Link	Cross Site Scripting (XSS)
Xiaopeng and Di [14]	-	SQL Injection (SQLi), Cross Site Scripting (XSS), Local and Remote File Inclusion (LFI- RFI), Cross Site Request Forgery (CSRF)

black-box analysis. In the dynamic analysis method, harmful inputs are sent to a web application, which is actively serving, the application's response is analyzed, and security vulnerabilities are detected according to the obtained responses. Since the results obtained with the dynamic analysis method are produced based on the runtime behaviours of the application, they are more reliable than the static analysis method, which produces a high rate of false-positive results. Furthermore, since all possible areas of the application cannot be tested, fewer results may emerge compared to the static analysis [3].

The first process to be performed in the dynamic analysis-based web application penetration test to detect vulnerabilities in a web application is to determine the scope of the test. In some cases, it is desired to test only the home page, which is the most frequently visited place by users, while in some cases, it is desired to test only subdomains. To perform a full web application penetration test, including all web pages belonging to the application in the scope will provide the most reliable results.

Another issue that should be determined in web application penetration tests is whether to use the HTTP GET method over the URL in the tests, the HTTP POST method through the forms in the pages, or both them. In the tests carried out over the URL, the parameters within the URL address are targeted.

- <https://www.teztest.com/sales.html?product=computer>

When the URL address shown above was examined, it was observed that the computer value was assigned to the product parameter, and the content displayed on the page was shaped according to this

assignment. In the tests to be carried out over the URL, the product parameter should be taken as a target, and by assigning payloads that will trigger various vulnerabilities to that parameter, the page's response should be measured. For example, the URL can be changed through the relevant parameter to check if the page has SQL injection vulnerability.

- <https://www.teztest.com/sales.html?product=1 ' OR ' 1 ' = ' 1>

In this case, if the statement `1 ' OR ' 1 ' = ' 1` assigned by the user to the relevant parameter is sent to the database server without any filtering to be added to the query, an SQL injection vulnerability will be detected.

Nowadays, URL rewrite engines make complex URL addresses clearer for search engines and users. Another purpose of this process is to prevent the understanding of the site's internal process and query logic. An example of a URL simplified by the URL rewrite technique is presented below.

- <https://www.teztest.com/sales.html?product=computer>
- <https://www.teztest.com/sales.html/computer/>

In testing, URL addresses are simplified by the URL rewrite technique; it is placed between the test payloads/characters. As mentioned earlier, the URL testing that addresses the SQL injection vulnerability should be administered as follows.

- <https://www.teztest.com/sales.html /1 ' OR ' 1 '= ' 1/>

When a request is made to the URL address presented above, the rewrite engine will parse the components in the URL and use them in the query process. As in

the previous example, the web page containing SQL injection will react to the related payload.

Another attack vector used in web application penetration tests is input; in other words, user input fields are present in forms on web pages. The process to be carried out while performing penetration tests on web applications is to test the application's reaction by entering harmful payloads that will trigger various vulnerabilities into each input field on the page. Each form element should be addressed separately in the tests to be performed on forms. Thus, it can be observed in which element there is vulnerability.

3.3 Random Forest Algorithm

The random forest algorithm is a supervised machine learning algorithm. As the name implies, a random forest, a collection of trained decision trees, is created in the random forest algorithm. In the random forest algorithm, the results from different learning models are combined to obtain a more accurate and stable prediction. The algorithm in question is a bagging version to which the randomness feature is added and which is improved. Instead of choosing the best branch among all the available variables, the random forest algorithm divides each node into branches using the best one among randomly selected variables in each node. All datasets in the random forest algorithm are produced by replacing the original dataset. After producing data sets, trees are developed by selecting a random feature. Given that the developed trees are not pruned, it increases the accuracy rate of the random forest algorithm. Another advantage of the random forest algorithm is that it is a high-speed algorithm [22].

The user was first asked to define two parameters to run the random forest algorithm. These variables were m , which represented the number of variables to be used in each section, and N , which represented the total number of trees to be developed. The first step after this process was to determine the training and test sets. From the data set used, n samples were selected by the bootstrap method. Of each sampling data selected, $2/3$ was used to create trees. This part is called the training set, also known as the inbag. The remaining $1/3$ of the sampling data is the test set for calculating the error rate. The test set is also called out-of-bag.

Among the variables randomly selected from the created training set, the variable with the highest information gain is used as a branching variable. This method is known in the literature as the classification and regression tree (CART) algorithm. The formula used to determine the information gain is presented in Equation 1.

$$\text{InformationGainValue}(D, X) = E(D) - \sum_i^n p(D_i)E(D_i) \quad (1)$$

The components of Equation 1 refer to are listed below;

- X =The variable to be divided,
- D =Subsets belonging to the feature to be divided,
- $E(D)$ =Entropy value before division according to variable X ,
- $E(D_i)$ =Entropy value of the i -th subset after division according to variable X .

The threshold value required for branching in the random forest algorithm is determined with the Gini index. The described processes were repeated in the random forest algorithm until the terminal node was obtained.

In the random forest algorithm, the estimates formed from n samples and produced by n trees are combined to estimate which class a new datum belonged to. The class with the most votes at the end of the combination becomes the class estimated for the data. The error rate of the test set determined this vote.

This algorithm calculated the error rate by applying the test data set on each tree created using the training data set. This error rate was calculated depending on the error rate of each tree in the forest and the correlation between them. In the developed model, the error rate obtained was low since the data and variables differed in each tree. As the error rate obtained decreases, the performance of the classification algorithm used increases [23].

When the random forest algorithm is used as a classification model for many data sets, it provides more precise results than the Adaboost and Support Vector Machine algorithms. Furthermore, the algorithm in question gives results in a very short time. Another advantage of the random forest algorithm is that it produces results with high accuracy, even in data sets with an unbalanced distribution containing thousands of variables and many class tags [24].

3.4 Apriori Algorithm

Although various algorithms have been proposed in the literature to create association rules, the most famous and widely used algorithm is the Apriori algorithm [25]. The Apriori algorithm is an association inference algorithm developed by Agrawal and Srikant in 1994. The term apriori, the algorithm's

name, comes from the expression of “previous” - “prior” since the algorithm uses the a priori knowledge of common cluster elements. In other words, it takes the knowledge from the previous step [26].

According to the basic logic of the Apriori algorithm, if the k-item set exceeds the specified minimum support threshold, the subsets of that set also provide the related minimum threshold value. The set expression mentioned in this expression consists of one or more elements. The k-item set’s expression means the set contains k elements [27].

The concepts of support and confidence explain each of the rules obtained after executing the apriori algorithm on data, namely the association rule between the elements. The support expression means the frequency of association between items due to the rule. The confidence expression refers to the accuracy of the association between the two items [28].

An example will be given to understand the Apriori algorithm’s working logic. A and B are different products sold in a market. The support threshold for product A is the ratio of baskets from which product A is purchased to all baskets. The support value for product A is shown in Equation 2:

$$\text{Support}(A) = \frac{\text{NumberOfBasketsContainingProductA}}{\text{NumberOfAllBaskets}} \quad (2)$$

The support threshold for products A and B was calculated as the ratio of the number of baskets from which both products were bought together to the total number of baskets. The calculation of this support value is shown in Equation 3:

$$\text{Support}(A, B) = \frac{\text{NumberOfBasketsContainingProductsAandB}}{\text{NumberOfAllBaskets}} \quad (3)$$

The probability that product B will be in the basket from which product A is bought is expressed with the confidence value. This confidence value can be calculated using one of the processes in Equation 4 or 5.

$$\text{Confidence}(A, B) = \frac{\text{NumberOfBasketsContainingProductsAandB}}{\text{NumberofallbasketscontainingproductA}} \quad (4)$$

$$\text{Confidence}(A, B) = \frac{\text{Support}(A, B)}{\text{Support}(A)} \quad (5)$$

While conducting an association analysis on the data with the Apriori algorithm, the user was first asked to enter the support and confidence threshold

to determine the validity of the rules to be obtained. For the association between the two products to be significant, both support and confidence criteria must be high. Determining a high support threshold will make the algorithm run faster while at the same time causing a decrease in the number of rules obtained. For example, not buying the same number of products during shopping will result in the inability to obtain the association rule regarding these products when combined with a high support threshold. A researcher who wants to obtain an association rule for each product should lower the support threshold and filter the results according to the high confidence value.

4 The Structure of Web Application Security Scanner- Sansar

Sansar web application vulnerability scanner was developed in the PyCharm environment, using the Python programming language, as a console-based. Under this heading, the modules of Sansar will be discussed first. Then, the development of the web penetration test laboratory and the data set creation will be mentioned. After explaining the page classification process and association analysis processes performed on the data set obtained, the integration of these processes into Sansar will be mentioned.

4.1 Crawler Module

The developed web application vulnerability scanner used three different methods to extend the scope of the tests to be performed as follows:

- (1) Subdomain collection process using search engines
- (2) Determination of hidden URL addresses with the brute force method
- (3) Browsing the whole web application using the links and forms given on the page

Within this study’s scope, the “domain” operator of the Bing search engine was used to collect the subdomain addresses of a domain address. To explain with an example, when the search was performed as “domain: sdu.edu.tr” in the Bing search engine, it was observed that records belonging to subdomains of “sdu.edu.tr” returned. The URL address generated at the end of the search is as follows.:

- <https://www.bing.com/search?q=domain%3Asdu.edu.tr&first=10>

The developed tool sends a request by adding the scanned domain address to the dark part of the URL address above and collects the sub-domain addresses from the response page. The process continues by moving to the next page with the “first” variable in the URL pattern. The process in question will con-

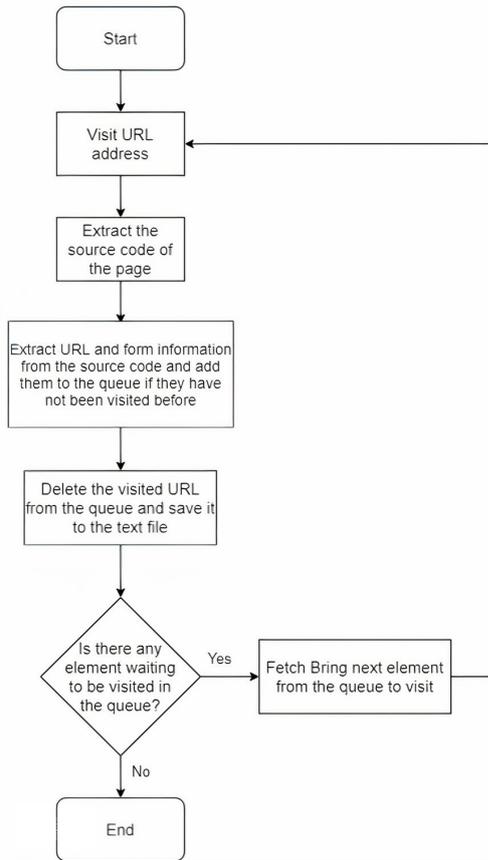


Figure 1. The algorithm followed in determining URL addresses from the page source

tinue until no more results are displayed. Pages, such as robots.txt or admin panels, hidden by application developers, are significant for attackers.

A module was added to Sansar’s crawler module that detected hidden pages not linked in the site by brute force technique. The module in question added the URL extensions. It takes from a dictionary to the end of the domain and subdomain addresses obtained from the web application and creates a new URL address. The tool then tried to detect hidden addresses by testing whether the new URL address it had created was valid.

Another method that the crawler module used to collect the web application addresses was the URL addresses detected on the page sources and the forms on the page. The algorithm followed in this process is shown in Figure 1. The developed tool started crawling first by visiting the URL address sent to it. Then, “script,” “iframe,” “a,” “frame,” “iframe,” “button,” and “meta” HTML tags included in the page source to the visited URL are extracted. After this step, URL addresses assigned to the “src,” “href,” “action,” and “help-equiv” fields included in the mentioned tags were collected. The collected URL addresses were added to

the list of URL addresses to be visited after checking whether they belong to the page domain navigated. Afterwards, the developed tool detects the forms on the visited web page. After the forms on the web page were detected, each was submitted by filling in the default values by the tool. Thus, if a redirection process were performed after submitting the form in the web application, that URL address would also be added to the list. Thanks to all the components it included, the developed crawler module produces a completed list of URLs.

4.2 Vulnerability Test Module

A list containing URL addresses collected by the crawler module is taken as a parameter by the test module of Sansar. The developed module performs vulnerability scanning on all URL addresses in the said list through HTTP GET and POST methods. The test module uses 21 different vulnerability test classes, as mentioned earlier.

Among the vulnerability test classes developed within the scope of this study, there are the following methods:

- a constructor method
- a method called get, which enables testing over GET
- a method called a post, which enables testing over POST
- a method named login process, which goes to the login page and performs the login process again in the case of a session break

Thanks to this structure created, when a new web vulnerability test class is added to the application, it will be sufficient to determine the payload list that will trigger the vulnerability in question and to determine the reaction (e.g., error messages and late viewing) that the application will give if the vulnerability is triggered. The operation steps of the test class belonging to the error-based SQL injection class from the created test classes are shown in Table 2.

When Sansar detected vulnerability while scanning a web page, various feature data belonging to the web page on which the vulnerability was detected, form, and inputs were collected. A unique dataset was created due to the collection of these data for the detected vulnerability. This data set was used in the following sections of this study to perform page classification and association analysis between vulnerabilities.

In the extraction of the features of the page with vulnerability, firstly, the Wappalyzer library was used. The Wappalyzer library methods were developed to obtain the feature values of the page, form, and input.

Table 2. The functioning of the test class of error-based SQL injection vulnerability

No	Testing steps of error-based SQL vulnerability
1	The values of the URL, session, login page address, username, and password sent as parameters are assigned to the class variables.
2	The GET and POST methods are run according to the user's request. (Since the operation logic of both methods is the same, it will be explained as a single process.)
3	It has tried to reach the page to be tested. If the page which is desired to be accessed directs the user to the login page, test processes are started by logging in.
4	SQL injection payloads are taken from the Payloads.py file.
5	The list of errors likely to be displayed as a result of injection is taken from the Errors folder.
6	The payloads taken are injected into the URL patterns prepared by the 'getParser' method if the GET method is run and into the POST patterns prepared by the 'FormScraper' method if the POST method is run.
7	The request is sent by using the created URL or POST patterns.
8	It is checked whether error messages prove vulnerability in the response corresponding to the sent request.
9	If an error message is encountered in the returned response, data, such as the name of the vulnerability, its address, and the session, are sent to the feature extraction module.
10	The features sent from the feature extraction module are sent to the reporting class.

After running all the developed methods, 70 features belonging to the page, form, and input where web vulnerability was detected were obtained. The list of the features collected by the developed tool is shown in Table 3.

When vulnerability is detected on the tested web page, after extracting the features shown in Table 3, the collected feature data are sent to the reporting module to add to the test report. When the test processes were completed, the reporting module sent the data together and wrote them into an Excel file.

4.3 Creation of the Web Application Penetration Testing Laboratory and Realization of the Test Processes

A web application penetration testing laboratory had to be established to measure Sansar's adequacy in testing vulnerabilities during the development and to form a data set by collecting data with the tests to be performed after the development. Thanks to the created laboratory, web applications can be tested with different types of structures prepared for different purposes. PHP and ASP-based web applications were used in the web application penetration testing laboratory created within the scope of this study. Web applications used to create the laboratory are open-source projects developers share on the Github, Gitlab, and Sourceforge platforms. There are 262 different web applications in total in the web laboratory. Two hundred seven of these web applications are PHP-based, and 55 of them are ASP-based web projects. In this study, PHP-based projects were installed on Apache and MYSQL servers in the XAMMP and WAMP environments. IIS and MSSQL were used

Ad	Değiştirme tarihi	Tür	Boyut
09-06-2019-11-57	6.09.2019 11:57	Microsoft Excel Ç...	17 KB
09-06-2019-12-09	6.09.2019 17:33	Microsoft Excel Ç...	17 KB
09-06-2019-14-10	11.09.2019 22:59	Microsoft Excel Ç...	10 KB
09-06-2019-14-33	11.09.2019 22:59	Microsoft Excel Ç...	18 KB
09-06-2019-17-24	6.09.2019 17:24	Microsoft Excel Ç...	8 KB
09-08-2019-22-14	11.09.2019 23:00	Microsoft Excel Ç...	10 KB
09-08-2019-22-17	11.09.2019 23:00	Microsoft Excel Ç...	19 KB
09-08-2019-22-39	11.09.2019 23:00	Microsoft Excel Ç...	15 KB
09-08-2019-22-45	9.09.2019 02:24	Microsoft Excel Ç...	24 KB
09-08-2019-23-45	11.09.2019 23:01	Microsoft Excel Ç...	11 KB
09-10-2019-01-59	11.09.2019 23:02	Microsoft Excel Ç...	15 KB
09-11-2019-12-56	12.09.2019 21:07	Microsoft Excel Ç...	11 KB

Figure 2. Scan reports of the tests performed

for the installation of ASP-based projects.

After the development of the Sansar and the completion of the web application penetration testing laboratory, the vulnerability testing processes were started. Vulnerability tests were completed by testing all web applications installed in the laboratory. The results of each test were recorded in an Excel file with the day, month, year, hour, and minute information of the completion date of the test. The screen image showing some of the results obtained after the test processes were performed is given in Figure 2. A data set was created by combining the results obtained after the test. The obtained data set had a unique value since it was created using a vulnerability scanner we developed in a laboratory we have established.

4.4 Realization of Classification Processes Using the Data Set Created

Before mentioning the classification studies conducted using the data set created, it is necessary to mention the data set used. There were 7781 samples, and 70 were attributed to each sample in the unique data

Table 3. Features collected by the vulnerability scanner

Feature Type	Features
Wappalyzer library features	'Payload', 'CMS', 'Message Boards', 'Database Managers', 'Documentation Tools', 'Widgets', 'Ecommerce', 'Hosting Panels', 'Javascript Frameworks', 'Comment Systems', 'Font Scripts', 'Web Frameworks', 'Editors', 'LMS', 'Web Servers', 'Rich Text Editors', 'Javascript Graphics', 'Mobile Frameworks', 'Programming Languages', 'Operating Systems', 'Search Engines', 'Web Mail', 'Marketing Automation', 'Web Server Extensions', 'Databases', 'Payment Processors', 'Dev Tools', 'Live Chat'
Page features	'description', 'keywords', 'generator', 'content-type', 'title', 'Num Of Forms', 'Num Of Inputs', 'Num Of Password Areas', 'Num Of Search Areas', 'Num Of Select', 'Num Of Radio', 'Num Of Option', 'Num Of Checkbox', 'Num Of Text', 'Num Of Email', 'Num Of Text Areas', 'Num Of File', 'Num Of Buttons'
Form features	'Num Of Inputs In Vulnerable Form', 'Types of Inputs In Vulnerable Form', 'Num Of Password Areas In Vulnerable Form', 'Num Of Search Areas In Vulnerable Form', 'Num Of Text Areas In Vulnerable Form', 'Num Of Select In Vulnerable Form', 'Num Of Radio In Vulnerable Form', 'Num Of Option In Vulnerable Form', 'Num Of Checkbox In Vulnerable Form', 'Num Of Buttons In Vulnerable Form', 'Action Of Form', 'Text of Form'
Input features	'name', 'type', 'src', 'size', 'required', 'value', 'max', 'min', 'maxlength', 'accept'

Table 4. Sample numbers according to vulnerabilities in the data set

Type of Vulnerability	Number
Blind SQL Injection	1747
Buffer Overflow	46
Command Execution	1119
Header SQL Injection	58
Carriage Return and Line Feed	3
Directory Traversal	850
HTML Injection	352
Local File Injection	750
Open Redirect	360
PHP-Code Injection	952
Header Based Cross-Site Scripting	2
Remote File Inclusion	9
Shellshock	83
Error Based SQL Injection	596
Server-side Includes	166
Server-Side Template Injection	224
XML External Entity	2
Cross-Site Scripting (XSS)	462

set. The number of samples of vulnerability types in the data set is shown in [Table 4](#).

Web vulnerability may emerge at many points according to the level of attention and knowledge of the encoder, the way of using the data, and the versions of other factors used, such as a server, media interpreter, and language. Rather than developing a model that produces direct and precise predictions between a vulnerability type and a page type, it would be

the most reasonable approach to determine the page type and then list the vulnerability types that may be encountered according to the page type. Through scans carried out using such an approach, both possible vulnerability points due to the nature of web vulnerabilities will be thoroughly tested, and vulnerabilities that are less likely to be observed will not be tested depending on the user's request.

The classification processes carried out within the scope of this study continued with the page-type classification process. In the page type classification process, the URL addresses of each sample in the data set were visited individually, and it was tagged which type the page was. When deciding on the page tag types, attention was paid to identifying the types commonly encountered on web pages and have different characteristics from other page types. In this study, eight different page types were determined under the mentioned criteria. These are as follows:

- Login Page
- Register Page
- Search Page
- List Page
- Info Page
- Forgot Password Page
- Contact Us Page
- Submit Page

Tagging was done by the first author and verified by the second author. After the tagging process on the created data set, the sample numbers of the obtained page types are shown in [Table 5](#). The number of pages of the Submit page type was high among the tagged samples, and the reason for this was that all the pages in the web applications that did not fall into other categories but were processed by taking user data were included in this group. There were

Table 5. The number of page types in the dataset after tagging

Page Type	Number
Contact Us Page	249
Forgot Password Page	102
Info Page	799
List Page	353
Login Page	932
Register Page	1209
Search Page	440
Submit Page	3707

many form pages created for different purposes in different applications. Since creating a different type for each page in question was impossible, these pages were included in the submit page.

After labelling, the required features for page classification were determined. The number of 70 features included in the data set was reduced to 13 according to various criteria, such as the effects of the features on the classification result and the correlation between features. The Weka program was used for feature selection. During feature selection, CfsSubsetEval is used as the attribute evaluator algorithm, and BestFirst is used as the search method algorithm. The features used in determining the page type in this study are listed below:

- Number of Forms
- Number of Inputs
- Number of Password Areas
- Number of Search Areas
- Number of Select
- Number of Radio Buttons
- Number of Option
- Number of Checkbox
- Number of Email Area
- Number of Textarea
- Number of Text
- Number of File
- Number of Buttons

Each listed feature shows the number of input components specified with its name on the web page to be classified. In applying each classification algorithm to the data set, the Pandas, NumPy, Sklearn, and Pickle libraries of Python were used. In the classification processes performed, 80% of the total data set was used to train the model, and 20% was used to test the trained model.

Classification processes were carried out on the data set created within the scope of this study using logistic regression, artificial neural networks, support vector machines, decision trees, and random forest

Table 6. The number of page types in the dataset after tagging

Random Forests	95.39%
Decision Tree	95.24%
Support Vector Machines	88.44%
Neural Networks	86.17%
Logistic Regression (Newton)	84.19%
Logistic Regression (Sag)	80.40%

algorithms, respectively. The prediction values obtained from the classification processes performed are shown in Table 6.

When the classification results shown in Table 6 are examined, it was observed that the highest success rate was obtained using the random forest algorithm. The decision tree algorithm obtained the closest success rate to the random forest algorithm. Sag, saga, liblinear, lbfgs, and newton-ng analyzers were used in the logistic regression algorithm's classification processes. Since the accuracy rates obtained from other analyzers were close, sag and newton-ng analyzers were included in the table. Figure 3 shows the screen images obtained by running the classes created for classification processes.

The confusion matrix for the classification process, which was performed using the random forest algorithm and in which the highest success rate was achieved, is shown in Figure 4. When the confusion matrix was examined, it was observed that the most inaccurate predictions were made on the pages of the Forgot Password Page type. Web applications have various types of "I forgot my password" pages. In some applications, there is an email input, to which the user's email address is entered, and a send button. The classifier can mix such pages with login pages. In some applications, different information about the user and the new password are requested, and if the information entered by the user is correct, the password renewal process is performed. The classifier can mix this forgot password page type, to which different user information and new passwords are entered, with the register page type, which contains similar inputs.

Precision, recall, f1 and weighted avg. Metrics are given in Table 7. As it is known, Macro-average gives equal weight to all classes regardless of the imbalance in the data set. It is, therefore, more susceptible to low scores than underrepresented classes. For this reason, we included weighted average measurements in our study.

Two new classes were written for Sansar to use the classification model developed using the Random Forests Algorithm. The first one of these classes will visit the page to be classified, extract the feature

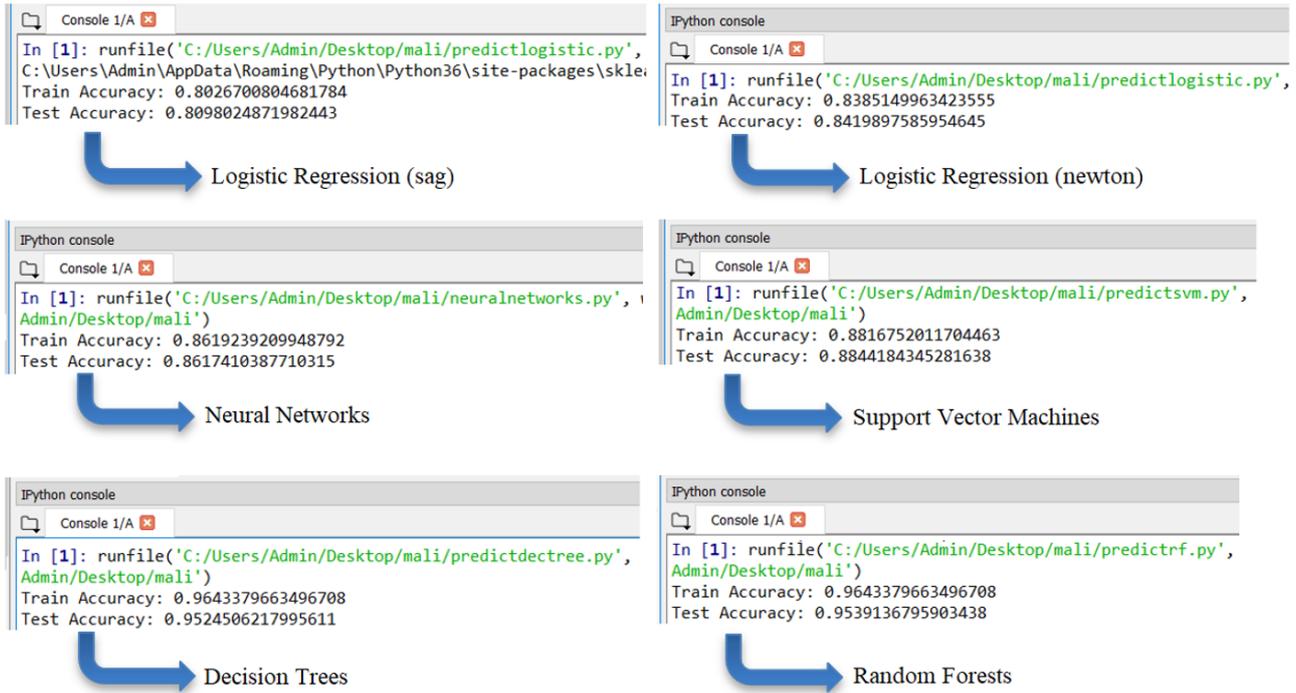


Figure 3. Accuracy rates obtained from the classes written for various algorithms

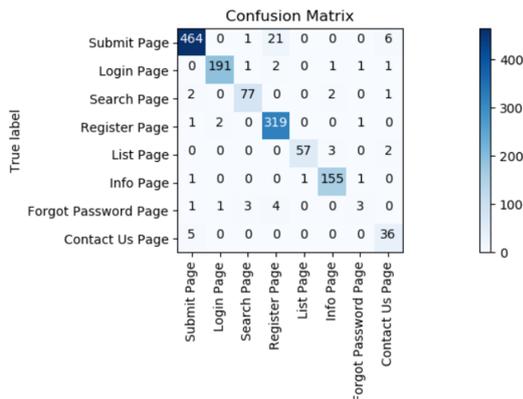


Figure 4. The confusion matrix of the classification process performed using the Random Forest Algorithm

Table 7. Metrics of classification process with random forests algorithm

	Precision	Recall	F1-Score
Contact Us Page	0.80	0.88	0.84
Forgot Password Page	0.75	0.25	0.38
Info Page	0.96	0.99	0.97
Login Page	0.98	0.97	0.98
Register Page	0.92	0.99	0.96
Search Page	0.94	0.94	0.94
Submit Page	0.98	0.95	0.96
Accuracy	*	*	0.95
Weighted Avg.	0.95	0.95	0.95

values to be used in the classification from the source codes of the page, and send them to the second class. The second class will put the data sent by the feature extractor class into the previously saved classification model and decide on the page’s class. Figure 5 shows the method which puts the values returned from the feature extraction class into the created classification model and returns the page type.

The first point at which the page classification process was used is the crawler module of Sansar. The developed model was used to recognize the login pages during the indexing of the pages by the navigation module and log in successfully. None of the existing web application vulnerability scanners in the literature can perform an artificial intelligence-based login process. After adding the page classification function, the URL addresses visited by the navigation module were first subjected to the classification process, and if the page was classified as a 'Login Page,' user login processes were initiated. A short block of code from the class where the processes in question are performed is presented in Figure 6.

Figure 7 shows the automatic detection of the login page and logging in during the crawling of http://testphp.vulnweb.com address. The vulnerability test module is another module that uses the classification function in the web application vulnerability scanner. This process will be explained after the realization of association analysis with the Apriori algorithm under the next heading.

```

def classifyPage(self):
    #load model
    self.features= pageFeatureExtractor(self.urlAdress, self.session).extractFeatures()
    for f in self.features:
        self.featuresArray.append(self.features[f])
    X_try = np.asarray([self.featuresArray])
    loaded_model = pickle.load(open('classification/rfmodel.sav', 'rb'))
    return loaded_model.predict(X_try)

```

Figure 5. The classifyPage method of the randomForestClassifier.py class

```

try:
    if Spider.isLoggendIn==False:
        typeOfPage = randomForestClassifier(partsOfString[1], Spider.session).classifyPage()
        if typeOfPage[0] == 'Login Page':
            print("[*]Login Page Detected. Starting Login Process...")
            print("[*]Login Page Address: "+partsOfString[1])
            Spider.contentOfLoginPage = Spider.session.get(partsOfString[1],headers=Spider.general_headers).content
            Spider.loginURL = partsOfString[1]
            currentUrlAdress = Spider.loginWithSelenium()

```

Figure 6. Spider.py class user login processes

```

[*]Extracting features of page for classification. Url Address: http://testphp.vulnweb.com/index.php
[-]Pages Waiting To Be Visited In The Queue:: 14
[+]Total Pages Visited: 8

[*]Extracting features of page for classification. Url Address: http://testphp.vulnweb.com/AJAX/index
[-]Pages Waiting To Be Visited In The Queue:: 13
[+]Total Pages Visited: 9

[*]Extracting features of page for classification. Url Address: http://testphp.vulnweb.com/privacy.php
[-]Pages Waiting To Be Visited In The Queue:: 12
[+]Total Pages Visited: 10

[*]Extracting features of page for classification. Url Address: http://testphp.vulnweb.com/guestbook
[-]Pages Waiting To Be Visited In The Queue:: 12
[+]Total Pages Visited: 11

[*]Extracting features of page for classification. Url Address: http://testphp.vulnweb.com/Mod_Rewrit
[-]Pages Waiting To Be Visited In The Queue:: 13
[+]Total Pages Visited: 12

[*]Extracting features of page for classification. Url Address: http://testphp.vulnweb.com/login.php
[*]Login Page Detected. Starting Login Process...
[*]Login Page Address: http://testphp.vulnweb.com/login.php
[-]Pages Waiting To Be Visited In The Queue:: 14
[+]Total Pages Visited: 13

```

Figure 7. The automatic detection of the login page and logging in at the time of testing

4.5 Association Analysis with the Apriori Algorithm

This study uses the apriori algorithm to extract association rules between vulnerability types. In other words, when a vulnerability is detected on a page, it is aimed to detect other vulnerabilities that can be seen on the same page with that vulnerability.

Samples included in the unique data set created differ in terms of the page type, the vulnerability discovered, and the type of method discovered (GET and POST). One thousand seven hundred seventy-eight of the vulnerability samples in the data set were determined over the URL; in other words, using the HTTP GET method. The remaining 6003 vulnerabilities were detected using the forms on the web pages; in other words, using the POST method. After the studies were carried out, it was observed that it was not possible to associate the vulnerabilities obtained using the GET method over the URL in the tests administered with the page types because, in the classification of page types, forms on the page, user input fields in the forms, and buttons are used. In the vulnerabilities detected over the URL address, the components on the page are not important. The URL address is segmented in the tests administered, payloads are embedded in the relevant fields, and attack attempts are carried out. For this reason, in the association analysis performed using the Apriori algorithm, only vulnerabilities detected in the components on the page (vulnerabilities determined by the POST method) were used.

To perform the association analysis over the existing data set, the data set was put into a form suitable for the Apriori algorithm. The vulnerability encountered on a web page was added to the new data set to be used in the Apriori algorithm by putting a comma between them. The process in question was conducted separately for each URL address and added as a separate row to the data set. The screen image of the new data set formed after the process performed is shown in [Figure 8](#).

The association analysis process was performed after creating the data set to be used. The association analysis process was done in the Spyder environment using Python programming. There are two parameters to be considered when using the Apriori algorithm. These are support and confidence values. The default support value for the Apriori algorithm is 0.1. [Figure 9](#) shows the association rules obtained at the support values = 0.1 and confidence = 0.8.

When the rules shown in [Figure 9](#) were examined, it was observed that the rules of the most common vulnerabilities in the data set emerged. Furthermore,

	A	B	C	D
1	DTRAV,PHPI,SSTI,LFI,SSI,COMMAND EXEC			
2	DTRAV,PHPI,SSTI,LFI,SSI,COMMAND EXEC			
3	BLIND SQLI,SQLI,XSS			
4	DTRAV,PHPI,SSTI,LFI,SSI,COMMAND EXEC			
5	DTRAV,PHPI,SSTI,LFI,SSI,COMMAND EXEC			
6	BLIND SQLI,SQLI,XSS			
7	BLIND SQLI			
8	BLIND SQLI			
9	BLIND SQLI			
10	BLIND SQLI			
11	BUFFER OVERFLOW			
12	LFI,PHPI,DTRAV			

Figure 8. The data set to be used in association analysis

Table 8. The number of page types in the dataset after tagging

Type of Vulnerability	Number
Blind SQL Injection	1565
Command Execution (CE)	852
Directory Traversal (DTRAV)	774
PHP Injection	704
Local File Inclusion (LFI)	642
Basic SQL Injection	538
HTML Injection	262
Cross Site Scripting (XSS)	320
Server-Side Template Injection (SSTI)	216
Server-Side Inclusion (SSI)	143
Buffer Overflow	13
Carriage Return and Line Feed (CRLF)	2
XML External Entity (XEE)	2

a very low number of one-to-one rules between vulnerabilities was obtained. The first reason the mentioned result is so inadequate is that the support value is determined as 0.1. [Table 8](#) demonstrates the number of vulnerabilities in the data set subjected to association analysis.

When the number of vulnerabilities demonstrated in [Table 8](#) is examined, it is observed that each vulnerability is of a different number. By their nature, web vulnerabilities can be observed on any web page. In a web page type, it would be false to say that a specific kind of vulnerability cannot be observed. The reason for not being able to reach certain opinions about web vulnerabilities is that every web programmer has a different programming habit and attaches different levels of importance to secure programming. Therefore, it is necessary to create an association rule for every vulnerability in the data set being studied

Index	antecedents	consequents	ntecedent suppoi	onsequent suppoi	support	confidence
2	frozenset({'HTMLI'})	frozenset({'XSS'})	0.120924	0.180707	0.111413	0.921348
3	frozenset({'DTRAV', 'COMMAND EXEC'})	frozenset({'LFI'})	0.131793	0.307065	0.11413	0.865979
5	frozenset({'DTRAV', 'PHPI'})	frozenset({'LFI'})	0.141304	0.307065	0.119565	0.846154
0	frozenset({'SQLI'})	frozenset({'BLIND SQLI'})	0.226902	0.557065	0.186141	0.820359
1	frozenset({'PHPI'})	frozenset({'COMMAND EXEC'})	0.383152	0.456522	0.309783	0.808511
4	frozenset({'PHPI', 'LFI'})	frozenset({'COMMAND EXEC'})	0.210598	0.456522	0.169837	0.806452

Figure 9. The association rules obtained with the support = 0.1 and confidence = 0.8 values

as much as possible. The way to achieve this would be to lower the support value, which is set as 10%. The figures given in Table 8 were examined, and it was decided to exclude the least encountered CRLF and memory overflow vulnerabilities from the analysis. This was because obtaining rational association rules for these vulnerabilities was not possible. The support threshold value for the process in question was 0.02.

Another reason for the low number of rules obtained in the first association analysis is that the confidence threshold value is determined to be 80%, as stated earlier. A confidence threshold set at a high level makes it difficult to obtain the association rule for vulnerabilities, such as Blind SQLi, encountered in many samples in the data set. Many web developers know and have blocking instincts against error-based SQL injection. However, it would not be correct to mention the same situation for the time-based blind SQL injection. Thus, the number of time-based blind SQL injections detected on the tested pages exceeded the number of error-based SQL injections. When the association analysis between two vulnerabilities was performed, the probability of observing the error-based SQL injection on the pages where the time-based blind SQL injection was observed was found to be 33%. Even if this rate is initially considered low, it would be wrong to say that there is little association between these vulnerabilities, which are tested with the same logic and of which only payloads used for the test differ. Therefore, the support threshold value was determined to be 0.02, and the confidence threshold value was 0.3 in the association analysis. The rule table of the association analysis conducted is presented in Figure 10.

After determining the rules following the association analysis conducted according to the data set, the mentioned rules were integrated into the Sansar. Table 8 shows the number of vulnerabilities detected in each page type in the data set used in page classification and association analysis. The algorithm will work as follows in the process of using association analysis after page classification. Firstly, the page

type of the URL address tested will be determined. Then, the most common five vulnerabilities in the specified page type will be detected from the data set. After this process, the first vulnerability test class will be called. Suppose the tested vulnerability is detected on the page this time. In that case, it will be referred to the rule table obtained from the association analysis, and vulnerabilities most likely to be observed on the page where the vulnerability is detected will be tested. For each vulnerability detected on the page, the association rule will be examined, and the associated vulnerabilities will be tested. Suppose no vulnerability to test in the association rule is left. In that case, the system will go back and continue the process with the vulnerability observed on the page most commonly but not tested. If no vulnerability to be tested is left in the list of most common vulnerabilities and the association rules, the scanning will end. Before reporting the obtained data, if there is any class which has not been scanned or about which an association rule has not been found while scanning, they will be asked whether these classes should be scanned before reporting. Thanks to this algorithm established, the vulnerability testing process will be based on certain logic instead of using all test classes with the brute force method, as standard web application vulnerability scanners do. In this way, vulnerabilities associated with each other will be scanned one after another. Another advantage of this system is that the test processes produce more successful results by the successive scanning of associated vulnerabilities, which may trigger each other. Another advantage of the developed algorithm is that fewer test classes will be used to test a page. In this way, the test classes of vulnerabilities, which are likely to degrade the web page and database structure and of which the possibility of being observed on the page is low, will not be run.

To better understand the method used, the vulnerability test steps built on a sample URL address are numbered below.

- (1) <http://tezttest.com/addinventory.php> URL address is taken from the URL list sent from the

Index	antecedents	consequents	ntecedent suppor	onsequent suppor	support	confidence
15	frozenset({'SSSI'})	frozenset({'COMMAND EXEC'})	0.0421196	0.456522	0.0326087	0.774194
21	frozenset({'SSSI'})	frozenset({'DTRAV'})	0.0421196	0.241848	0.03125	0.741935
29	frozenset({'SSSI'})	frozenset({'LFI'})	0.0421196	0.307065	0.03125	0.741935
31	frozenset({'SSSI'})	frozenset({'PHPI'})	0.0421196	0.383152	0.0326087	0.774194
36	frozenset({'SSSI'})	frozenset({'SSTI'})	0.0421196	0.0597826	0.0217391	0.516129
8	frozenset({'SSTI'})	frozenset({'BLIND SQLI'})	0.0597826	0.557065	0.0285326	0.477273
16	frozenset({'SSTI'})	frozenset({'COMMAND EXEC'})	0.0597826	0.456522	0.048913	0.818182
22	frozenset({'SSTI'})	frozenset({'DTRAV'})	0.0597826	0.241848	0.0380435	0.636364
30	frozenset({'SSTI'})	frozenset({'LFI'})	0.0597826	0.307065	0.0434783	0.727273
32	frozenset({'SSTI'})	frozenset({'PHPI'})	0.0597826	0.383152	0.0394022	0.659091
35	frozenset({'SSTI'})	frozenset({'SSSI'})	0.0597826	0.0421196	0.0217391	0.363636
3	frozenset({'HTMLI'})	frozenset({'BLIND SQLI'})	0.120924	0.557065	0.0747283	0.617978
24	frozenset({'HTMLI'})	frozenset({'SQLI'})	0.120924	0.226902	0.0720109	0.595506
25	frozenset({'HTMLI'})	frozenset({'XSS'})	0.120924	0.180707	0.111413	0.921348
9	frozenset({'XSS'})	frozenset({'BLIND SQLI'})	0.180707	0.557065	0.116848	0.646617
26	frozenset({'XSS'})	frozenset({'HTMLI'})	0.180707	0.120924	0.111413	0.616541
34	frozenset({'XSS'})	frozenset({'SQLI'})	0.180707	0.226902	0.13587	0.75188
7	frozenset({'SQLI'})	frozenset({'BLIND SQLI'})	0.226902	0.557065	0.186141	0.820359
23	frozenset({'SQLI'})	frozenset({'HTMLI'})	0.226902	0.120924	0.0720109	0.317365
33	frozenset({'SQLI'})	frozenset({'XSS'})	0.226902	0.180707	0.13587	0.598802
2	frozenset({'DTRAV'})	frozenset({'BLIND SQLI'})	0.241848	0.557065	0.108696	0.449438
10	frozenset({'DTRAV'})	frozenset({'COMMAND EXEC'})	0.241848	0.456522	0.131793	0.544944
17	frozenset({'DTRAV'})	frozenset({'LFI'})	0.241848	0.307065	0.163043	0.674157
19	frozenset({'DTRAV'})	frozenset({'PHPI'})	0.241848	0.383152	0.141304	0.58427
4	frozenset({'LFI'})	frozenset({'BLIND SQLI'})	0.307065	0.557065	0.127717	0.415929
12	frozenset({'LFI'})	frozenset({'COMMAND EXEC'})	0.307065	0.456522	0.233696	0.761062
18	frozenset({'LFI'})	frozenset({'DTRAV'})	0.307065	0.241848	0.163043	0.530973
27	frozenset({'LFI'})	frozenset({'PHPI'})	0.307065	0.383152	0.210598	0.685841
5	frozenset({'PHPI'})	frozenset({'BLIND SQLI'})	0.383152	0.557065	0.127717	0.333333
14	frozenset({'PHPI'})	frozenset({'COMMAND EXEC'})	0.383152	0.456522	0.309783	0.808511
20	frozenset({'PHPI'})	frozenset({'DTRAV'})	0.383152	0.241848	0.141304	0.368794
28	frozenset({'PHPI'})	frozenset({'LFI'})	0.383152	0.307065	0.210598	0.549645
1	frozenset({'COMMAND EXEC'})	frozenset({'BLIND SQLI'})	0.456522	0.557065	0.168478	0.369048
11	frozenset({'COMMAND EXEC'})	frozenset({'LFI'})	0.456522	0.307065	0.233696	0.511905
13	frozenset({'COMMAND EXEC'})	frozenset({'PHPI'})	0.456522	0.383152	0.309783	0.678571
0	frozenset({'BLIND SQLI'})	frozenset({'COMMAND EXEC'})	0.557065	0.456522	0.168478	0.302439
6	frozenset({'BLIND SQLI'})	frozenset({'SQLI'})	0.557065	0.226902	0.186141	0.334146

Figure 10. The association rules obtained with the support = 0.02 and confidence = 0.3 values

Table 9. The number of vulnerabilities by the page type included in the data set

	Submit Page	Register Page	Login Page	List Page	Info Page	Search Page	Contact Us Page	Forgot Password Page
Blind SQLi	723	271	366	28	50	41	60	26
Command Exec.	573	50	31	40	90	18	38	12
DTRAV	532	55	37	24	8	18	387	32
PHPi	468	40	34	31	68	16	34	13
SSTi	123	34	6	49	1	3	0	0
LFi	457	53	36	32	16	14	23	11
SQLi	227	134	93	2	5	51	24	2
HTMLi	131	83	6	1	2	35	4	0
Open Redirect	49	120	127	0	56	4	3	1
SSI	80	37	4	14	9	5	3	0
XSS	49	88	66	3	2	103	8	1
Buffer Overflow	0	0	4	0	9	1	0	0
CRFL	0	0	2	0	1	0	0	0
XXE	0	0	0	2	0	0	0	0

- Crawler module and inserted into the classifier.
- (2) The “Submit Page” value returns from the classifier.
 - (3) The most common vulnerabilities in the Submit Page class are determined according to the data set.
 - (4) The testing process starts with blind SQLi, the first of the most common vulnerabilities in the Submit Page type (1. Blind SQLi, 2. Command Execution, 3. Directory Traversal, 4. Code Injection).
 - (5) The blind SQLi vulnerability is detected on the page. According to the Apriori Algorithm, scanning continues with SQLi, the vulnerability with the highest possibility of being observed with the blind SQLi and then with command execution.
 - (6) Both of the tested vulnerabilities cannot be detected. Since there is no other vulnerability related to the blind SQLi, according to the rule table, the scanning process continues with the next element among the vulnerabilities observed most commonly on the Submit page.
 - (7) The command execution vulnerability, the second element in the list of most observed ones on the Submit page, was previously tested due to its association with blind SQLi. Therefore, the testing process will continue with the directory traversal test, the third most common vulnerability on the Submit page.
 - (8) The testing process will continue until no test classes are left to run in the ordered list and the association rule lists.
 - (9) The tests will end When no class is run for

testing in the ordered list and the association rule list is left. However, as the last step before reporting, users are asked if they want to run the remaining test classes in the scanning process. However, they are the test classes about which an association rule has not been found and, therefore, have not been run.

- (10) After the necessary process, if any, is carried out according to the user preference, it is passed to the reporting step, and the scanning process is completed.

After the classification model and association rules are integrated, the class diagram of Sansar’s vulnerability test module is shown in Figure 11. Sansar has the following capabilities after the addition of classification and association rules:

- to perform tests using both GET and POST,
- to perform multithreading with dynamically produced threads according to the number of computer cores,
- to log in by scanning the user login pages automatically by giving only username and password information, and to scan the pages on the user panel,
- to produce more successful and logical test results by running test classes associated with each other one after another instead of running all test classes directly, as in other vulnerability scanners in the literature.

5 Discussion

In this study, an artificial intelligence and dynamic analysis-based web application vulnerability scanner

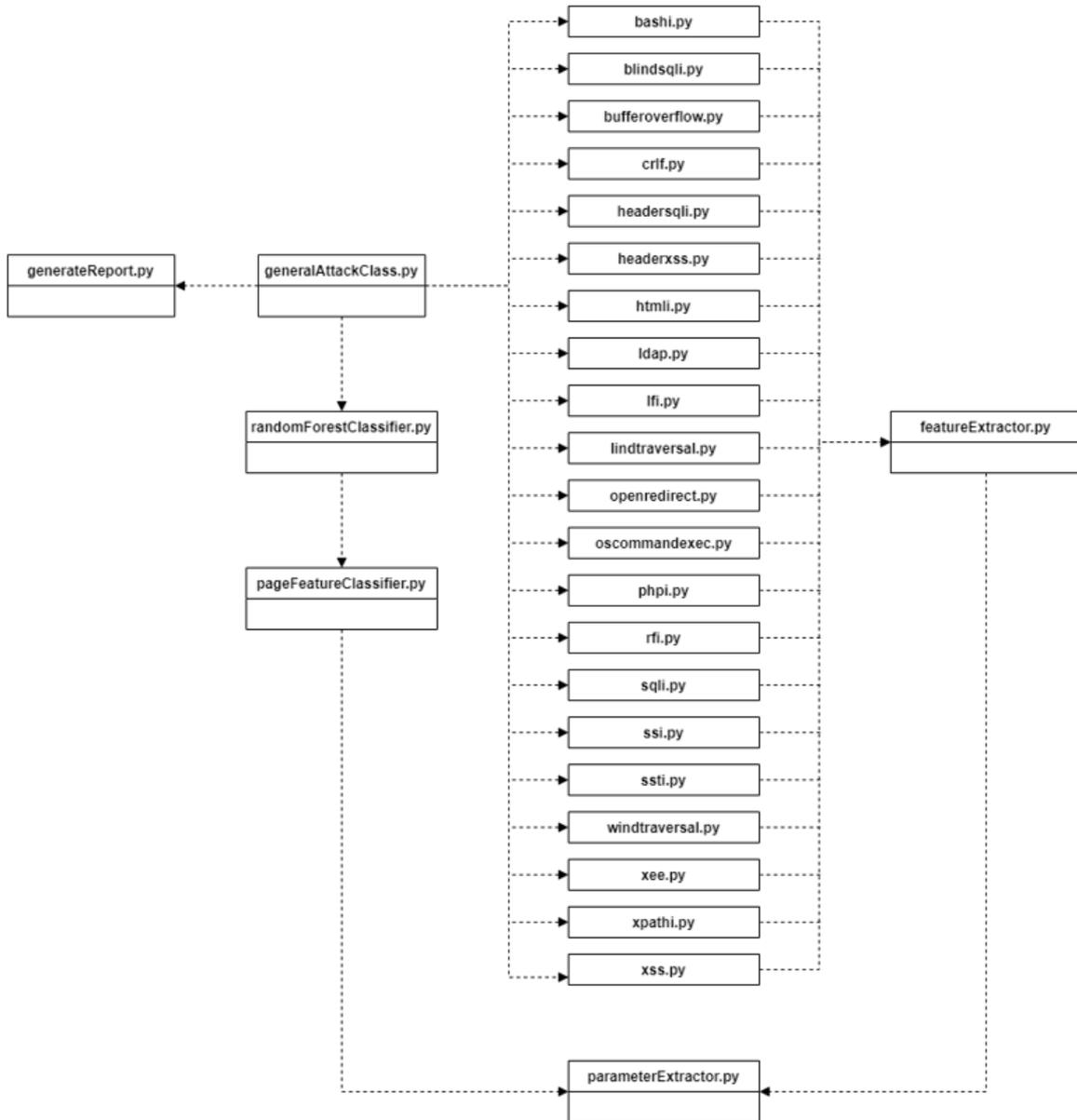


Figure 11. Class diagram showing the association between classes

named Sansar was developed. Sansar gives better results than the previous studies in the literature from various aspects. The comparison of the present study conducted with other studies in the literature will be dealt with separately for each module.

When the Sansar is compared with WASCAN, WAPITI, VEGA, Arachni, Skipfish and NIKTO, which are six open-source scanners commonly used in web penetration testing by security experts, it provides superiority concerning the number of vulnerability types it tests. WASCAN tests 16 different vulnerabilities.

WAPITI, widely used in the black box web penetration tests, can test 12 vulnerabilities. VEGA vul-

nerability scanner also used as a proxy in penetration tests, can test about ten different types of vulnerabilities. Developed with the Ruby language, Arachni can successfully test 14 different types of vulnerabilities. The Skipfish tool, installed by default in Kali Linux and widely used, can test vulnerabilities at 19 different criticality levels. NIKTO, on the other hand, can test 14 different vulnerabilities. Sansar contains 21 different vulnerability test classes. Furthermore, vulnerability test classes were planned and coded so that it was very easy to adapt when a new vulnerability was to be added.

The largest web penetration testing laboratory in the literature was created in this study. Within the laboratory are also types of sites, such as a blog,

news, shopping and banking, and automation applications, such as schools, restaurants, hotels, hostels, libraries, planes, and buses. Within the prepared web penetration testing laboratory were 207 PHP and 55 ASP-based web applications. Thanks to the laboratory-created, the opportunity to test the most probable web applications in real life was obtained. When the studies in the literature are examined, it is seen that most studies use existing vulnerable web applications such as WebGoat and DVWA. In some studies, researchers have developed their vulnerable web applications or set up laboratories consisting of several applications. In their study, Karakaya and Uçar tested the effectiveness of different web application vulnerability scanners on various web applications. This study used three different web applications in testing processes [29]. Yang and others have developed a laboratory for cybersecurity testing. One web application was published over a web server in the developed laboratory [30]. Basin *et al.* discussed the vulnerabilities that may arise in web applications, and they set up a basic laboratory for these vulnerability tests. The developed laboratory has two web applications, namely OWASP WebScarab and OWASP WebGoat [31]. Wenliang Du has developed an open-source lab environment that offers a practical approach to computer security education, allowing students to experience security scenarios similar to real-world situations. This laboratory has five different web applications to test different types of web vulnerabilities [32]. Chen and Tao have developed a web application called SWEET, which contains many different types of vulnerabilities [33].

Using the web application vulnerability scanner developed in this study, each application in the created web application penetration testing laboratory was tested. A data set was obtained using the vulnerability data acquired after the tests. In the literature, the data sets used in the page classification processes using HTML tags contain features formed by segmenting the text data in the mentioned tags. However, in the data set formed in this study, HTML tags were addressed numerically. Within the data set, there is information on how many HTML tags are on the page and the form where the vulnerability is detected. In the related dataset, 70 different features belong to the page, form, and input where the vulnerability is found. To our knowledge, the data set created within the scope of this study is the only data set in the literature considering the features it contains. In this way, the classification and association analysis processes carried out using the mentioned data set are unique.

The first process performed using the data set obtained in this study is the page classification process. The samples in the data set are tagged in eight dif-

ferent types: login page, register page, submit page, search page, info page, list page, forgot password page, and contact us page. The page classification process was carried out after tagging the samples in the data set. Five different classification algorithms were used in the page classification process. The highest accuracy rate among the performed classification processes was obtained with the random forest algorithm, with a success rate of 95.39%. This rate is unique in the literature regarding the originality of the data used and the page type classified.

In page classification studies on HTML tags in the literature, web pages were classified for purposes such as indexing and advertisement under categories such as sports, news, and technology. Furthermore, the content of the determined HTML tags, i.e., text data, was used in these studies. In this study, the only classification process performed using the amount of HTML input fields available on the page was carried out. The success rate also included an original and successful study in the literature.

Another feature of Sansar is using a classifier while recording the URL addresses to be tested for vulnerability by crawling the web application before the test. In this way, the developed tool recognizes the user login pages of the web application and logs in successfully. None of the commercial and open-source web application vulnerability scanners used widely in web tests nowadays have this feature. Open-source WASCAN, WAPITI, VEGA and Skipfish tools need valid cookie data to log in. User login should be made to the web page to be scanned; the cookie data generated after the login should be obtained somehow and given as a parameter to the mentioned tools. This process is challenging to realize for users, time-consuming, and unsustainable. NIKTO web vulnerability scanner, on the other hand, cannot log in during scanning. Arachni can log in to web applications with form-based login (autologin) and Script-based login (login_script) methods. However, this tool also requests the address of the login page from the user before scanning. The two leading web application vulnerability scanners among commercial web application vulnerability scanners in the cybersecurity world are Netsparker and Acunetix. The Netsparker tool requests the login page's URL address, username, and password information from the user to scan the pages behind the verification by logging in to a web application. The Tenable Nessus tool also uses the same method as Netsparker. The Acunetix tool requests the user to log in to the web application to be scanned using a scanner. It opens within itself before scanning. In this way, it records the information about the address the user has entered and inputs in which the user name and password have been written

and then repeats later. The Portswigger Burp Suite tool asks the user to save the login steps, as in the Acunetix tool. Sansar needs only valid username and password information to log in. It classifies the pages it navigates when running one by one and checks whether there is a login page. When the login page is detected in the classification process, the inputs on the page are automatically filled in with the username and password information, and the login process is performed successfully. This feature is a new function brought to the literature in web application vulnerability scanners.

In this study, a new test model, produced by combining page classification and association analysis processes, was developed to be used in web application security tests. The answer to the question, “What type of vulnerability is more likely to appear on a page where vulnerability is detected?” was searched in the association analysis process performed in the developed model. After the association analysis was conducted, the web application vulnerability scanner was updated to run the test class of the vulnerability associated with that vulnerability when it detected vulnerability. Experienced web application penetration testing specialists verified the results obtained at the end of the association analysis. In the opinions received, it was stated that the obtained associations between vulnerabilities fitted the structures of the vulnerabilities. According to the developed model, firstly, the web page to be tested is classified according to the page type detected after classification, based on the data set, the test class of the most commonly observed vulnerability is run. During the testing processes, if a vulnerability is detected, according to the association rules obtained with the association analysis, other vulnerability test classes that are most likely to be observed on pages where that vulnerability is observed are run. Thus, the tests continue in a way that they are based on the experience and logic of a true penetration testing expert. Thanks to the developed model, test classes of vulnerabilities not associated with the detected vulnerabilities will not be run. Thus, the degradation of the page structure or database tables due to unnecessary tests will be prevented. The findings obtained in this study suggest that the developed model provides faster results than the classical model, in which all test classes are run in sequence without considering an association.

In Figure 12, the testing process was carried out using two versions of the developed vulnerability scanner: in one of them, standard scanning was performed, and in the other, scanning was performed using the model of “page classification-association analysis.” The testing process was carried out on the <http://testphp.vulnweb.com> address. The reason for

choosing this address for testing is that this website is a laboratory developed by Acunetix, which has vulnerabilities and is open to everyone. These test processes, performed on the same target, using the same test classes, and with the same internet connection, allowed comparing the developed page classification-association analysis-based model with the classical scanning method. As is observed in Figure 12 (a), scanning using the classical method was completed in 2428 seconds; in other words, in approximately 40 minutes. As Figure 12 (b) shows, the scanning process using the developed model was completed in 1912 seconds; in other words, in approximately 31 minutes. After both scans were performed, 29 vulnerabilities, including seven blind SQL injections, two error-based SQL injections, four command executions, four HTML injections, four local file injections (LFI), four PHP code injections, and four cross-site scriptings (XSS), were determined. When the results were examined, it was detected that the developed model had the same effect as the standard scanning model but ran faster. According to the findings obtained, it was observed that the developed new model provides a speed gain of 21% compared to the standard scanning model.

To our knowledge, the page classification-association analysis-based test model created in this study is the only study aiming to run test classes within the framework of certain rules in dynamic analysis-based web application vulnerability tests. Vulnerability scanners used nowadays run test classes after they detect the input fields to be tested. A few vulnerability scanners (e.g., WAPITI) stipulate running another class before running some test classes (e.g., the SQLi test before the BlindSQLi test). However, to our knowledge, no study aims to model all testing processes.

The developed tool has been compared with the WAPITI, WASCAN, VEGA, Arachni, Skipfish, NIKTO, Netsparker, and Acunetix tools. This is because these tools are the most commonly used tools in web application tests nowadays and continue to be developed and updated due to their widespread use. The tool developed in this study is superior in various aspects compared to other studies in the field of web application vulnerability scanners in the literature. For example, in SQLi tests, the Secubat tool [4] creates a confidence value by examining the post-test return page response and using various response patterns and keywords to check whether the test is successful. The SQL injection test class developed within the scope of this study includes an extensive error checklist defined for 14 different types of database servers, such as Oracle, MySQL, and MS SQL. Sania is another web application vul-

```

C:\Users\Admin\Desktop\sansarvulnerabilityscanner-master>python sansar.py -u http://testphp.vulnweb.com/login.php -a -c --username test --password test -m post

SANISAR
-----
SAISAR - Web Application Vulnerability Scanner
Mehmet Ali YALCINKAYA

[*]Excel File for Dataset is Created.
[*]The attributes that are collected for the dataset are currently being written to the excel file.
Time:2428.0118176

C:\Users\Admin\PycharmProjects\sansarvulnerabilityscanner>python sansar.py -u http://testphp.vulnweb.com -c -a --username test --password test -m post

SANISAR
-----
SAISAR - Web Application Vulnerability Scanner
Mehmet Ali YALCINKAYA

[!]There are 7 unused test classes: ['bufferoverflow.py', 'crlf.py', 'ldap.py', 'ssi.py', 'ssti.py', 'xpathi.py', 'xee.py']
[!]Do you want to continue scanning over POST with unused test classes? (Y/N)
[*]Excel File for Dataset is Created.
[*]The attributes that are collected for the dataset are currently being written to the excel file.
Time:1912.8944968

```

(a)

(b)

Figure 12. Comparison of the developed model and the classical method

nerability scanner developed in the literature [5]. This tool works like a Proxy server between the web application and the database server and captures SQL queries sent from the application to the server. The mentioned tool performs vulnerability testing by adding SQL injection test payloads to the queries it captures. The vulnerability scanner developed, on the other hand, dynamically leads to harmful queries containing test payloads by examining the form fields on the page from HTML source codes instead of catching the queries. Another web application vulnerability scanner in the literature is SDAPT [6]. This vulnerability scanner uses the static analysis method to detect potential SQL injection attack points on the web page. The attack points obtained with the static analysis method are then tested with the dynamic analysis method. The tool in question lags behind the tool developed in this study since it performs a single vulnerability test and needs the application's source codes to be tested. As reported above, the studies reviewed under the web application vulnerability scanner heading in the literature mostly aimed to test a few specific vulnerabilities. In the study carried out in 2009 [7], a tool that performs SQL injection tests on web services was developed. In the study conducted in 2010 [8], a web application vulnerability scanner was developed to detect SQL injection and XSS vulnerabilities. Galan *et al.* [9] developed a tool to detect XSS vulnerabilities in web applications. Ali *et al.* [10] developed a tool

called MySQLInjector that performed SQL injection vulnerability testing on web applications developed with the PHP language. In their study, Singh and Roy [11] developed a tool called NVS to detect SQL injection vulnerabilities on web applications. In the study conducted by Djuric in 2013 [12], a web application vulnerability scanner named SQLIVDT was developed to detect SQL injection vulnerabilities. In their study, Qianqian and Xiangjun [34] added an interface to an open-source web application vulnerability scanner running from the command screen and published it as a service. Basak *et al.* [35] present a review of web application vulnerability scanners that detect web application vulnerabilities. In addition, the effectiveness of several vulnerability scanners for many different types of web application vulnerabilities has been examined. Jain *et al.* [36] examined three open-source web application scanners, namely Arachni, Nikto, and Wapiti and their methods for detecting and reporting different vulnerabilities. Hu *et al.* [37] present a method for automatic vulnerability scanning web applications using machine learning techniques. Shahid *et al.* [38] conducted a comparative study of web application vulnerability scanners and evaluated various features and techniques. The study in question found that automatic scanners outperform manual scanners. It was also noted that the scanners gave different results in terms of accuracy, sensitivity and specificity. Joshi *et al.* [39] evaluated the effectiveness of web application vulnerability

scanners. Their study tested it in various applications using 14 different browsers. The results showed that the scanners differed in accuracy, sensitivity and specificity. It was also stated that all the tested scanners could detect different vulnerabilities. Huang *et al.* [40] proposed a new web application vulnerability scanning method. The method in question can detect vulnerabilities in applications by combining deep learning and data mining techniques. The new method was tested on various applications, and it was stated that successful results were obtained. In their study, Rabheru *et al.* [41] presented an approach for detecting security vulnerabilities in web applications using machine learning techniques. In the study, as mentioned earlier, an automated tool has been developed for the scanning process. This tool uses a pre-trained classifier to detect vulnerabilities commonly found in web applications.

When the reviewed studies and the developed tool are compared, it has been observed that the Sansar vulnerability scanner can test more vulnerabilities compared to the studies in question. In the literature, there are studies in which artificial intelligence models are used for vulnerability detection. The developed Sansar application, unlike these studies, uses the classification process only to determine the types of web pages. It is used to determine the type of a web page, recognize login pages, and login automatically. Sansar was the first software in the literature to use association rules to detect vulnerabilities in vulnerability tests.

6 Conclusion

With web applications, many jobs such as product buying and selling, banking transactions and smart home control can be performed today [42]. Data stored and processed in web applications has become a critical resource for individuals and organizations [43]. Therefore, it is important to protect the data in question and to perform vulnerability tests of web applications at certain intervals. In this study, a web application vulnerability scanner named SANSAR was developed. When all the comparisons made have been summarized, it has been observed that Sansar has more test classes than open-source vulnerability scanners that are widely used in penetration tests. It is the only tool that uses the classification model in indexing web applications. This way, Sansar can automatically detect login pages and log in without needing external input or intervention. In addition, a model based on page classification-association analysis has been created to be used in web application tests, and it is observed that the created model performs faster and more reliable tests than the classical scanning method. We hope that integrating artificial

intelligence into web application vulnerability scanners will be quite inspiring for future work.

References

- [1] Verizon Enterprise data breach investigations report 2022. <https://www.verizon.com/business/resources/reports/dbir/>. Accessed: 2023-04-15.
- [2] ITRC identity theft resource center breach report hits record high in 2021. <https://www.idtheftcenter.org/publication/2022-data-breach-report/>. Accessed: 2023-04-16.
- [3] G Deepa and P Santhi Thilagam. Securing web applications from injection and logic vulnerabilities: Approaches and challenges. *Information and Software Technology*, 74:160–180, 2016.
- [4] Stefan Kals, Engin Kirda, Christopher Kruegel, and Nenad Jovanovic. Secubat: a web vulnerability scanner. In *Proceedings of the 15th international conference on World Wide Web*, pages 247–256, 2006.
- [5] Yuji Kosuga, Kenji Kono, Miyuki Hanaoka, Miho Hishiyama, and Yu Takahama. Sania: Syntactic and semantic analysis for automated testing against sql injection. In *Twenty-Third Annual Computer Security Applications Conference (ACSAC 2007)*, pages 107–117. IEEE, 2007.
- [6] William GJ Halfond, Shauvik Roy Choudhary, and Alessandro Orso. Penetration testing with improved input vector identification. In *2009 International Conference on Software Testing Verification and Validation*, pages 346–355. IEEE, 2009.
- [7] Marco Vieira, Nuno Antunes, and Henrique Madeira. Using web security scanners to detect vulnerabilities in web services. In *2009 IEEE/IFIP International Conference on Dependable Systems & Networks*, pages 566–571. IEEE, 2009.
- [8] Jan-Min Chen and Chia-Lun Wu. An automated vulnerability scanner for injection attack based on injection point. In *2010 International Computer Symposium (ICS2010)*, pages 113–118. IEEE, 2010.
- [9] Eduardo Galán, Almudena Alcaide, Agustín Orfila, and Jorge Blasco. A multi-agent scanner to detect stored-xss vulnerabilities. In *2010 International Conference for Internet Technology and Secured Transactions*, pages 1–6. IEEE, 2010.
- [10] Abdul Bashah Mat Ali, Mohd Syazwan Abdullah, Jasem Alostad, et al. Sql-injection vulnerability scanning tool for automatic creation of sql-injection attacks. *Procedia Computer Science*, 3:453–458, 2011.
- [11] Avinash Kumar Singh and Sangita Roy. A network based vulnerability scanner for detecting

- sql attacks in web applications. In *2012 1st international conference on recent advances in information technology (RAIT)*, pages 585–590. IEEE, 2012.
- [12] Zoran Djuric. A black-box testing tool for detecting sql injection vulnerabilities. In *2013 Second international conference on informatics & applications (ICIA)*, pages 216–221. IEEE, 2013.
- [13] Soyoung Lee, Seongil Wi, and Soel Son. Link: Black-box detection of cross-site scripting vulnerabilities using reinforcement learning. In *Proceedings of the ACM Web Conference 2022*, pages 743–754, 2022.
- [14] TIAN Xiaopeng and TANG Di. A distributed vulnerability scanning on machine learning. In *2019 6th International Conference on Information Science and Control Engineering (ICISCE)*, pages 32–35. IEEE, 2019.
- [15] Patrick Dave P Woogue, Gabriel Andrew A Pineda, and Christian V Maderazo. Automatic web page categorization using machine learning and educational-based corpus. *Int. J. Comput. Theory Eng*, 9(6):427–432, 2017.
- [16] Luca Deri, Maurizio Martinelli, Daniele Sartiano, and Loredana Sideri. Large scale web-content classification. In *2015 7th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management (IC3K)*, volume 1, pages 545–554. IEEE, 2015.
- [17] WA Awad. Machine learning algorithms in web page classification. *International Journal of Computer Science & Information Technology (IJCSIT)*, 4(5):93–101, 2012.
- [18] Myungsook Klassen. A frame work for search forms classification. In *2012 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pages 1029–1034. IEEE, 2012.
- [19] Yanbo Ru and Ellis Horowitz. Automated classification of html forms on e-commerce web sites. *Online Information Review*, 31(4):451–466, 2007.
- [20] Yılmaz Vural. Enterprise information security and penetration testing, June 2007. Available at https://tez.yok.gov.tr/UlusalTezMerkezi/tezDetay.jsp?id=8MxC_qYwjgBUX1uTtg0jhg&no=ULsA81dv3hIIQ38dGrZdeA.
- [21] Khairul Anwar Sedek, Norlis Osman, Mohd Nizam Osman, and Jusoff Hj Kamaruzaman. Developing a secure web application using owasp guidelines. *Comput. Inf. Sci.*, 2(4):137–143, 2009.
- [22] Özlem Akar and Oguz Güngör. Classification of multispectral images using random forest algorithm. *Journal of Geodesy and Geoinformation*, 1(2):105–112, 2012.
- [23] Ebru Korkem. Random forest and naïve bayes approach in microarray gene expressions data sets, June 2013.
- [24] Hülya Yılmaz. Studying the missing data problem in random forestsmethod and an application in health field, June 2014.
- [25] Chih-Hsuan Wang and Su-Hau Nien. Combining multiple correspondence analysis with association rule mining to conduct user-driven product design of wearable devices. *Computer Standards & Interfaces*, 45:37–44, 2016.
- [26] Yen-Liang Chen, Jen-Ming Chen, and Ching-Wen Tung. A data mining approach for retail knowledge discovery with consideration of the effect of shelf-space adjacency on sales. *Decision support systems*, 42(3):1503–1520, 2006.
- [27] D Ay and İ Çil. The use of association rules in store layout planning at migros türk a. ş. *Endüstri Mühendisliği Dergisi*, 21(2):14–29, 2008.
- [28] Emre Güngör, Nesibe Yalçın, and Nilüfer Yurtay. Apriori algoritması ile teknik seçmeli ders seçim analizi. In *Akademik Bilişim*, 2013.
- [29] Jason Bau, Elie Bursztein, Divij Gupta, and John Mitchell. State of the art: Automated black-box web application vulnerability testing. In *2010 IEEE symposium on security and privacy*, pages 332–345. IEEE, 2010.
- [30] T Andrew Yang, Kwok-Bun Yue, Morris Liaw, George Collins, Jayaraman T Venkatraman, Swati Achar, Karthik Sadasivam, and Ping Chen. Design of a distributed computer security lab. *Journal of Computing Sciences in Colleges*, 20(1):332–346, 2004.
- [31] David Basin, Patrick Schaller, and Michael Schläpfer. *Applied information security: a hands-on approach*. Springer, 2011.
- [32] Wenliang Du. Seed: hands-on lab exercises for computer security education. *IEEE Security & Privacy*, 9(5):70–73, 2011.
- [33] Li-Chiou Chen and Lixin Tao. Teaching web security using portable virtual labs. In *2011 IEEE 11th International Conference on Advanced Learning Technologies*, pages 491–495. IEEE, 2011.
- [34] Wu Qianqian and Liu Xiangjun. Research and design on web application vulnerability scanning service. In *2014 IEEE 5th International conference on software engineering and service science*, pages 671–674. IEEE, 2014.
- [35] Debadri Basak, Dhruv Ramani, and Ghanshyam Singh. Enhancement of unsupervised object detection using supervised method. In *2020 11th International Conference on Computing, Communication and Networking Technologies (ICCNT)*, pages 1–9. IEEE, 2020.
- [36] Ravi Kant Jain, Bikash Gupta, Mustaq Ansari, and Partha Pratim Ray. Iot enabled smart drip

- irrigation system using web/android applications. In *2020 11th international conference on computing, communication and networking technologies (ICCCNT)*, pages 1–6. IEEE, 2020.
- [37] Lilan Hu, Jie Chang, Ze Chen, and Botao Hou. Web application vulnerability detection method based on machine learning. In *Journal of Physics: Conference Series*, volume 1827, page 012061. IOP Publishing, 2021.
- [38] Jahanzeb Shahid, Muhammad Khurram Hameed, Ibrahim Tariq Javed, Kashif Naseer Qureshi, Moazam Ali, and Noel Crespi. A comparative study of web application security parameters: Current trends and future directions. *Applied Sciences*, 12(8):4077, 2022.
- [39] Chanchala Joshi and Umesh Kumar Singh. Performance evaluation of web application security scanners for more effective defense. *International Journal of Scientific and Research Publications (IJSRP)*, 6(6):660–667, 2016.
- [40] Yao-Wen Huang, Shih-Kun Huang, Tsung-Po Lin, and Chung-Hung Tsai. Web application security assessment by fault injection and behavior monitoring. In *Proceedings of the 12th international conference on World Wide Web*, pages 148–159, 2003.
- [41] Rishi Rabheru, Hazim Hanif, and Sergio Maffei. A hybrid graph neural network approach for detecting php vulnerabilities. In *2022 IEEE Conference on Dependable and Secure Computing (DSC)*, pages 1–9. IEEE, 2022.
- [42] Salman Sherin, Muhammad Zohaib Iqbal, Muhammad Uzair Khan, and Atif Aftab Jilani. Comparing coverage criteria for dynamic web application: An empirical evaluation. *Computer Standards & Interfaces*, 73:103467, 2021.
- [43] Kevin W Hamlen and Bhavani Thuraisingham. Data security services, solutions and standards for outsourcing. *Computer Standards & Interfaces*, 35(1):1–5, 2013.



Mehmet Ali Yalçinkaya received her master's and Ph.D. degrees from Süleyman Demirel University, Institute of Science, Department of Computer Engineering. He is currently working as a member of Kırşehir Ahi Evran University, Computer Engineering Department. Among his research fields are Cyber Security, Web Application Security, Computer Networks.



Ecir Uğur Küçükşille received her master's degree from Süleyman Demirel University, Institute of Science, Department of Mechanical Education and his Ph.D. from Süleyman Demirel University, Institute of Social Sciences, Department of Business/Numerical Methods. He is currently working as a member of Süleyman Demirel University Faculty of Engineering, Computer Engineering Department. Among his research fields are; Cyber Security, Data Mining, Natural Language Processing.