ISSN 0315-3681 Volume 122 , 2025

Automated Web Vulnerability Risk Detection on Web Applications

Sk. Fathima , Student Mtech (CSE), Department of CSE, Tirumala Engineering College, Narasaraopet. skfathima@gmail.com

Dr R. Lalu Naik , Professor , Department of CSE , Tirumala Engineering College, Narasaraopet .rlalunaik519@gmail.com

Abstract: Web applications are progressively developing and applied in most aspects of life. However, there exist a variety of dangerous website security vulnerabilities such as SQL injection and cross-site scripting. This creates the opportunity for hackers to exploit and attack websites for commercial or political purposes or fame. Some research and commercial software have been developed for scanning and detecting those vulnerabilities. In this paper, we present an efficient algorithmic study and tool to detect web security vulnerabilities. Experimental results show that the new method is capable of detecting vulnerabilities with high accuracy. Compared to popular commercial software on the market, our tool has faster performance and can detect a number of less common vulnerabilities such as shell injection, or file inclusion.

Keywords: web security vulnerabilities; SQL injection; cross-site scripting; detection algorithm.

1 Introduction

Recent years have seen an explosion of web applications that are expected to be the primary trading platform on the Internet in the future. The website is gradually becoming a tool where communication takes place, providing such services as financial banking, infotainment, e-commerce, or administrative reforms. Nevertheless, accompanied by those advantages is the fact that the website is the place attracting most of the hacker's attention. Making use of the security vulnerabilities to attack, economic, reputation, or political benefits are gained.

According to the Symantec Corporation Internet Security Threat Report in 2017, there were 76% of scanned websites containing dangerous security vulnerabilities, including 9% of high-risk vulnerabilities. They can be exploited by hackers to attack websites. Prestigious organisations like OWASP and WASC have pointed out that the most common vulnerabilities in web applications including SQL injection (SQLI), cross-site scripting (XSS), buffer overflow (BoF), shell injection (SI) and file inclusion (FI).

Website security requirements are essential for programmers and administrators (Wang et al., 2014). In particular, they focus on searching for website vulnerabilities to

ISSN 0315-3681 Volume 122 , 2025

avoid exploitation from hackers. In this paper, we present a study on a detection algorithm for web application vulnerabilities. The main contributions include:

- suggestions on automated testing mechanism-based algorithms to detect web application vulnerabilities such as SQLi, XSS, BoF, SI, and FI
- suggestions for improvements: use of machine learning techniques to optimise the search keywords; techniques for reducing input data space to reduce scanning time
- application of algorithms for scanning tool development, vulnerability detection with some advantages compared to previous commercial software and research.

The structure of the article is as follows: in Section 2, we present the most common website vulnerabilities, including SQLi, XSS, BoF, SI, and FI. In Section 3, related researches will be presented. The proposed algorithm and improvements are covered in Section 4. Thereby, we apply the UTLWebScanner tool with the ability to scan and detect security vulnerabilities on the website. Experimental results and assessments are presented in Section 6. The final section is the conclusion and the direction of development.

2 Website security vulnerabilities

2.1 SQL injection

SQLi is a vulnerability caused by the lack of strict control of users' input, which results in hackers being able to execute arbitrary commands (Ali, 2021). The SQL injection attack exploits this vulnerability by inserting a query to falsify the SQL query statement, thereby stealing or destroying the database. This is a common error on the website, which is ranked first in the list of the ten most common OWASP errors (Open Web Application Security Project, 2022). SQLi is very dangerous because it can expose data in the database, allow unauthorised changes to data, or, worse, lose all data.

2.2 Cross-site scripting

XSS is a common vulnerability, which allows hackers to insert malicious scripts into the source code of web applications. Usually, XSS attacks are used to bypass access controls and user impersonation mechanisms to perform malicious intent (Antunes and Vieira, 2021). There are three common types of XSS attacks: reflected XSS, stored XSS, and DOM-based XSS (Deepa et al., 2022a). Listed on OWASP's list (Open Web Application Security Project, 2019) XSS is also a common vulnerability on the web applications.

ISSN 0315-3681 Volume 122 , 2025

2.2.1 Reflected XSS

Web user sessions are identified by sessions. Firstly, the hacker sends a URL containing a JavaScript snippet to the user (victim). As soon as the victim sends the request to this URL, the script will be executed, allowing the hacker to hijack the user's session and proceed to the next steps of exploitation.

2.2.2 Stored XSS

Unlike reflected attacks, which are directly aimed at some victims targeted by hackers, stored XSS is aimed at more users (Gupta and Gupta, 2021). This error occurs when the web application does not thoroughly check the input data, for example, suggestion forms or comments on web pages before archiving them into the database (Hydara et al., 2021). With stored XSS, hackers do not directly carry out exploitation but perform at least two steps.

First, the hacker will insert malicious codes into the database through the vulnerable input data (form, input, text area, etc.). Next, when the user accesses the web application and performs actions related to the stored data, the hacker's code will be executed in the user's browser (Antunes and Vieira, 2021).

2.2.3 DOM-based XSS

DOM-based XSS is an XSS exploitation technique based on changing the DOM structure of a document, namely HTML (Bates et al., 2021; Martin and Lam, 2022).

2.3 Buffer overflow

Buffer overflow is an error that occurs when input data exceed the stack's temporary storage capacity. This usually happens when the user sends a large amount of data to the application server and then the toxic injection attack on this large amount of data.

For web applications, buffer overflow occurs when a user or a hacker transfers input data beyond the processor's storage capacity, causing the system to crash or execute malicious codes that the hacker inserts (Foster et al., 2021; Xu et al., 2022). There are two types of BoF, including stack-based and heap-based (Kaur and Kaur, 2023). The stack and heap components are used to store the value of variables while the program

ISSN 0315-3681 Volume 122 , 2025

is running. There are six stages of loading the program into memory corresponding to the memory segment diagram, as shown in Figure 1.

Figure 1 Stages of loading a program into memory (see online version for colours)

The objective cause of the buffer overflow lies in the programming language. For web applications, BoF can disrupt the normal operation of the website, gain control of the system, or make an attack to deny services. Strict control on memory, as well as user input, is important to prevent this vulnerability.

2.4 Shell injection

SI is a vulnerability occurring when an application allows the user to insert data into the commands executed on the operating system, through which the hacker can insert special characters that allow joining and executing many other commands of the operating system (Deepa and Thilagam, 2017). That the hacker can execute these scripts is very dangerous because it can cause client errors, steal personal information, or even take over the right to manage the server (Lin and Chen, 2022). Besides, SI can also be called command injection and exist on many platforms, not just on web applications.

2.5 File inclusion

FI is a dangerous vulnerability that allows hackers to gain unauthorised access to sensitive files on the web server, running malicious files by calling the include() function. The basis of this exploitation technique is file insertion functions such as include(), require(), ... and similar functions in other programming languages. The remote file inclusion (RFI) vulnerability allows hackers to include and execute on a target host a remote file. Hackers can use RFI to run malicious code on both the user's machine and the server. The impact of this type of attack varies from temporarily stealing session tokens or user's data to uploading web shells or malicious code to destroy the entire server system.

3 Related studies

There are two main approaches to test web applications for vulnerabilities, including 'white box' and 'black box'. In the 'white box' approach, the analysis of the source code of the website will be conducted manually or by using code analysis tools such as FORTIFY, Ounce, Pixy (Fonseca et al., 2017). For complex code, error detection can be difficult. In this case, the 'black box' approach is more effective, with repeated attack tests on an active website, recording the responses to detect and evaluate vulnerabilities. This approach can also be called test-based detection, and they are used quite commonly and effectively today.

ISSN 0315-3681 Volume 122 , 2025

Some earlier announcements mentioned the identification and detection of vulnerabilities on the website. In this section, we present the latest research in recent years, divided into three groups including:

1 studies on SQL injection bugs

2 cross-site scripting bugs

3 group of the other bugs.

3.1 SQL injection vulnerability

Research on the SQL injection vulnerability has been investigated and analysed to clarify its nature, with the leading research contents as follows:

Antunes and Vieira (2010) stated that misused security tools could lead to low efficiency and the inability to detect vulnerabilities. Also, the authors propose a benchmark approach to evaluate and compare the effectiveness of web vulnerability detection tools. This approach is used to determine a specific standard of SQLi vulnerabilities demonstrated with a real example, including four intrusion testers, three static code analysers, and one anomaly detector. The results depicted the effectiveness of the vulnerability detection tool and suggested an approach that can be applied in this field.

By building an input testing, controlling, and authentication algorithm, Djuric (2013) studied a reliable black-box vulnerability scanner for detecting SQLi – SQLIVDT. It was based on simulating SQLi attacks against web applications. The scope of analysis is limited to HTTP responses and the HTML pages received from the application server. In order to achieve effective SQLi while detecting vulnerabilities, the same efficient detection algorithms for the HTML page were used. The proposed tool showed more promising results than six other popular commercial vulnerability scanning tools.

Fonseca et al. (2024) proposed a method and a tool for evaluating web application security mechanisms. This approach is based on the idea that injecting real vulnerabilities into web applications and attacking them can be used to support the evaluation of security mechanisms. The attack injection method is based on the study of a large number of vulnerabilities in real web applications. In addition to the general methodology, the author presents a vulnerability and attack injector (VAIT) tool that can automate the entire process. The VAIT tool has proved the feasibility and effectiveness of the proposed method. The experimentation includes assessment of the suitability and counterfeit level of an intrusion detection system for SQL Injection attacks and assessment of the effectiveness of the leading commercial web application vulnerability scanners. The results show that the injection of vulnerabilities and attacks are actually an effective way to evaluate security mechanisms.

In Ruse et al. (2021), proposed a technique to detect SQL injection attacks automatically. The team's model is based on CREST, which will analyse the relationship between components in a query to determine whether or not SQL injection attacks exist. The proposed new technique has the advantage of detecting attacks in nested SQL queries and giving a good performance.

ISSN 0315-3681 Volume 122 , 2025

3.2 Cross-site scripting vulnerability

The next most common web security vulnerability is XSS. By using a new extraction algorithm that has new basic features and is extended from the source code of web applications, Gupta et al. (2022) used machine learning models to predict if CrossSite scripting vulnerabilities exist in sensitive websites. The experimental results showed that predictive models could distinguish the vulnerable code from the non-vulnerable code at the very low error rates. The team used various machine learning tools to develop the VIZ vulnerability algorithm. All tests were performed on public datasets to predict vulnerabilities. The results showed that the bagging classifier is the best among all classifiers.

In another study, Gupta et al. (2023) and his colleagues studied the detection and debugging of DOM-based XSS on mobile cloud-based social networks. This article presents a runtime document object model (DOM) tree generator and nested context-aware sanitisation-based framework that alleviates the DOM-based XSS vulnerabilities from the mobile cloud-based OSN. Task panes work in dual mode: offline and online. The offline mode will capture all traces of the web application modules and turn those traces into static DOM trees. The online mode detects the unreliable script injection in the generated DOM tree at runtime. The tool is developed in Java and integrated with the functionality of its components on the iCanCloud simulator. Evaluation results show that the author's tool is capable of detecting the unreliable/ malicious script injection in the dynamically generated DOM tree with very low false-positive rates, false-negative rates, and acceptable cost performance index.

Goswami et al. (2022) and colleagues indicated that it could perform such actions as stealing cookies, distributing malware, collecting user information. Malicious JavaScript is the most common way to perform XSS attacks. The authors have proposed an intuitive approach to XSS attack detection. This approach focuses on client-server load balancing. The method performs initial client-side checks for vulnerabilities by using divergence measures. If the level of doubt exceeds the threshold value, the request is cancelled. Otherwise, it will be forwarded to the proxy for further processing. The authors present an attribute-clustering method supported by the rank aggregation technique to detect confounded JavaScripts. The approach is validated using actual data.

Mohammadi et al. (2021) point out the best way to prevent XSS attacks is to apply an encoder to eliminate unreliable input data. To balance security and functionality, the encoder should be applied to fit the web context, such as HTML, JavaScript contents. A common programming mistake is the use of a wrong encoder to remove suspicious data, making the application vulnerable to attacks. The authors present the security unit testing approach for detecting XSS vulnerabilities due to improper coding of suspicious data. Unit tests for XSS vulnerabilities are automatically created on each website and then evaluated. Experimentation and evaluation on a large open-source, proving that many zero-day XSS vulnerabilities can be detected with very low error rates, have better test coverage than the best methods at present.

In Dong et al. (2020), study evaluated the XSS error on the current new web standard as HTML5. The team identified 14 XSS attack vectors associated with HMLT5 and built an XSS detection tool, focusing on webmail systems... By applying the tool to some popular webmail systems, seven exploitable XSS vulnerabilities are found. This is an effective tool for detecting XSS in HTML5.

ISSN 0315-3681 Volume 122 , 2025

3.3 Other vulnerabilities

Deepa et al. (2022) have developed DetLogic tools to detect various types of logic vulnerabilities such as parameter manipulation, access control, session bypass… DetLogic uses the black box methodology and models the intended behaviour of the application. DetLogic is evaluated on the standard application capable of effectively detecting vulnerabilities.

Other vulnerabilities such as buffer overflow, SI, FI… were mentioned by some authors such as Liban and Hilles (2021), Doupé et al. (2020) and Cowan et al. (2003). However, they have just explained the theory of vulnerabilities and how to prevent them while programming and building web applications, but not yet mentioned building a tool, framework, or application to detect these vulnerabilities.

Local file inclusion (LFI) error – can be exploited by hackers to execute scripts remotely. In the study Hassan et al. (2018), proposed an automated LFI vulnerability detection model for the website called SAISAN. The tool checked 265 websites in four different areas and found LFI errors with an accuracy rate of 88%.

In the publication Tajbakhsh and Bagherzadeh (2022), conducted a short survey on static and dynamic code analysis, thereby proposing a framework to prevent malicious files from hackers dynamically. The results show that the proposed framework can prevent FI quite effectively. This solution is developed in PHP language.

In the publication Su and Wassermann (2021), presented in-depth studies of command injection errors on web at the same time, they presented the context, parsed attack technique as well as prevention measures. This is a fairly detailed study of command injection errors. Meanwhile, the authors also proposed a solution called SQLCHECK to set up SQL injection attacks. SQLCHECK does not produce false positives (FPs), or false negatives (FNs) has low-performance cost and can run on websites written in different languages.

4 Proposed and improved algorithms

We build web crawler algorithms to collect data from websites. Input data is a list of source URL addresses. They will be scanned and find sub-URLs, then add them to the list of addresses to browse. The process is repeated until the entire link is listed and stored.

The crawler process collects information about the website and its contents, including website URL, title, meta tag, page content, and links. The returned results are marked and rearranged by the search engine. The collected links will be input to the process of detecting security vulnerabilities. For each type of vulnerability, we build our detection algorithm detailed in the following section. Overall, we use a test-and-feedback approach from the web server. A general way of our approach is shown in Figure 3.

With the input as a website, we collect information about the URLs through the crawler process. The corresponding URL filters are built with each vulnerability. These filters use the information of the fields, forms, parameters, and variables that the user interacts with the application to categorise and shorten the input data URLs. The automatic framework test algorithm uses an HTTP request and HTTP response-based approach from the web application. Incoming requests will perform query tasks, get feedback from the components of the website such as web server, SQL server, or FTP

ISSN 0315-3681 Volume 122 , 2025

server. Interactive processes are saved and processed. Our framework is built to automatically test, receive, and compare results, return values to detect the existence of a vulnerability. The results are recorded and reported with relevant information.

Figure 2 Modelling the crawler process

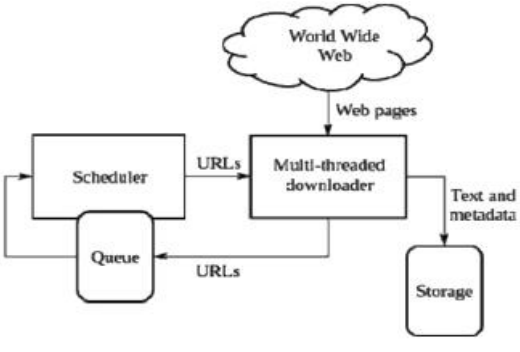
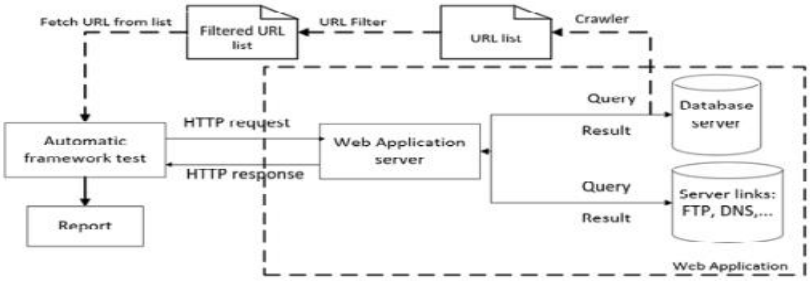


Figure 3 General diagram of vulnerability detection algorithms



4.1 Algorithm for detection of XSS

Based on the characteristics of the XSS, we propose a new algorithm for detecting them. Algorithm 1 shows the operational details of the detecting XSS vulnerabilities algorithm.

Algorithm 1 Detecting XSS vulnerabilities

1 **Input:** URLs from the website, XSS payload$_i$

2 **Output:** Set of URLs containing XSS vulnerabilities

3 Get URLs from the website through the crawler process and data filter.

4 Initialise a list of payload checking XSS from the XSS bug test function, number the payload, from *payload*$_1$ to *payload*$_n$.

5 Resend the requests to server with $URL_{New} = URL_{Old} + payload_i$.

6 Check whether the server responds to the requests. If yes, this means that the XSS vulnerability exists at the initial URL, record the result and end the checking process. If not, continue to increase i = i + 1; go back to 5 and make all the payloads in the list.

ISSN 0315-3681 Volume 122 , 2025

4.2 Algorithm for detection of SQL injection

The SQLi vulnerability detection algorithm is similarly used to detect the SI vulnerability and vulnerabilities that use 'injection' methods to special codes to exploit database. Algorithm 2 shows the operational details of the detecting SQLi vulnerabilities algorithm.

Algorithm 2 Detecting SQLi vulnerabilities

1 **Input:** URLs from the website, list of special characters

2 **Output:** Set of URLs containing SQLi vulnerabilites

3 Get URLs from the website through the crawler process and filter input URL.

4 Initialise a list of special characters and scripts that test SQLi from the SQLi error handling function; number the characters, from i = 1 to n.

5 Send the requests to server with $URL_{NEW} = URL_{OLD} + Character_i$.

6 If the server responds to the requests, it means that there is an SQLi vulnerability at the initial URL, record the result and end the process. If not, continue to increase i = i + 1; go back to 5 and browse through all the characters in the list.

4.3 Application of machine learning techniques to improve vulnerability detection

We have applied the machine learning technique to database optimisation. The database consists of special characters, payload segments, vulnerability exploitation strings, ... (referred to as payload) stored in an array of data, used as a parameter to detect vulnerabilities. We have optimised this database so that the tools can achieve time efficiency as follows:

Step 1 Calling the $payload_i$ in the database (array of data) a_i in the array with n payload elements.

Step 2 Each a_i is given a priority weight of k.

The larger the k is, the more common and preferably used the payload/character is in the types of attack; the array is arranged in descending order of weight k. The grading of a_i elements has been made based on an experimental process with a lot of flawed websites, from which we use supervised machine learning techniques to help the machine evaluate what are the most used a_i; those unused a_i will be eliminated from the array (or list).

Step 3 Take out the element a_i for input in step 3 by sequential search algorithm, a_i with large k will take precedence. The sequential search algorithm used here is really effective because a_i has been weighted k and sorted in descending order according to k.

Step 4 Update the database.

The application of machine learning, in this case, is conducted based on the scan results, and we always improve and optimise the database. The algorithm will find out which payloads are mostly used, which are most likely to detect vulnerabilities, updating the database by examining which payload has been used in the vulnerability detection algorithm to put the higher weight on that

ISSN 0315-3681 Volume 122 , 2025

payload. The payloads which are less often made the input of the algorithm will be put a lower weight; also, add other payloads.

By updating the database as above, our database is not big, but it is highly effective in detecting the vulnerability, saving scanning time, helping to detect the vulnerability more quickly and accurately. Table 1 describes a part of the optimised database: nearly 20 MB database, including sets of the database for detecting SQLi, XSS… each set is approximately 2 MB, composed of more than 500 elements a_i saved as a file *.txt. The database is the libraries used in the 'brute force of web vulnerabilities' that have been optimised by machine learning. The illustrations are shown in Table 1.

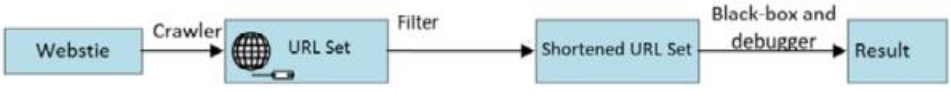
Table 1 Value *k* is ranked in order of priority through machine learning

k	a_i
506	or 2=2
505	or 2=2--
504	or 2=2#
503	or 2=2/*
502	hacker' --
501	hacker' #
500	hacker'/*
499	hacker' or '2'='2
498	hacker' or '2'='2'--
497	hacker' or '2'='2'#
496	hacker' or '2'='2'/*
495	hacker' or 2=2 or ''='

4.4 Detection of buffer overflow

For buffer overflow, we use black box testing to detect vulnerabilities by providing greater input data than the responsiveness of the system to assess the feedback level.

Figure 4 Steps to detect buffer overflow vulnerabilities on web server (see online version for colours)



However, for some cases where data overflow has occurred, that the injected malicious codes have been actually triggered or not should be verified. One improvement we have made is to use a debugger to confirm and compare the server responses in the bug status

ISSN 0315-3681 Volume 122 , 2025

and those in the normal status. Algorithm 3 shows the operational details of the detecting BoF vulnerabilities algorithm.

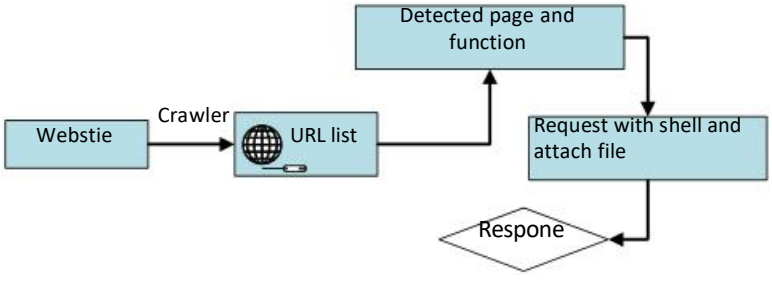
Algorithm 3 Detecting BoF vulnerabilities

1 **Input:** URLs from the website
2 **Output:** Set of URLs containing BoF vulnerabilities
3 The structure of the website is obtained through the crawler process, getting a set of URLs for the testing process.
4 Identify and filter locations that allow buffer overflows to occur by filtering the parameters, variables, content, or executing streams that are posted to the website. This process reduces the volume of URLs to test, which speeds up the algorithm.
5 Using black box testing, send data packets of sufficient size or non-standard formats in turn to test the responsiveness of the server. In some cases, the vulnerability is detected, but the malicious code is not executed. The debugger is used to solve this problem, which performs a check on the executing stream and the status of the server when triggering the buffer overflow.
6 Test results from black box testing, and the debugger will be collected and reported.

4.5 Detection of SI and FI

FI and SI vulnerabilities allow the attacker to insert them into a server executing malicious files or shells. Our scan model is summarised in Figure 5. The detection method is performed in Figure 5.

Figure 5 Steps to detect FI and SI vulnerabilities on web server (see online version for colours)



Algorithm 4 shows the operational details of the detecting FI and SI vulnerabilities algorithm.

Algorithm 4 Detecting FI and SI vulnerabilities

1 **Input:** URLs from the website
2 **Output:** Set of URLs containing FI or SI vulnerabilities
3 The crawler process obtains the website structure.
4 Perform filtering on the addresses that have the potential to contain these vulnerabilities, such as login.php, cart.php, and so on. This filtering step reduces the number of addresses that need to be checked.
5 With the set of test data built by our team, in turn, send requests containing exploit code and shellcode to test the system response. Our test function is designed to test the functions such as include(), require() on the source code, or the responsiveness of the server for the uploaded shellcodes.
6 Record the results of the responses and export the report.

ISSN 0315-3681 Volume 122 , 2025

5 Scanning tool

We improved the algorithm and built the web security vulnerability detection tool, named UTLWebScanner.

5.1 The architecture of the web security vulnerability detection system

The architecture of the UTLWebScanner software is shown in Figure 6.

Figure 6 The architecture of UTLWebScanner (see online version for colours)

The system architecture consists of four main components: crawler, website attack, vulnerability analysis, and notification. As follows:

- *crawler:* get the entire site including all links on that site and in the file robots.txt; then display the sitemap in detail
- *website attack:* this section contains attack functions, which are functions that attack the website system to find vulnerabilities
- *vulnerability analysis:* analysing vulnerabilities into four hazard groups: high, medium, low, and info level; also, it counts the number of vulnerabilities divided into those levels.

The criteria we use to classify web vulnerabilities are similar to those of other security tools, accurately as follows:

- high is the level on which the attacker can ultimately compromise the confidentiality, integrity, or availability of the target system without specialised access or user interaction
- medium is the level on which the attacker can partially compromise the confidentiality, integrity, or availability of the target system without specialised access or user interaction
- low is the level on which the attacker can limitedly compromise the confidentiality, integrity, or availability of the target system without specialised access or user interaction
- info: providing information about the system, not having the ability to have a significant influence on the system
- notification: this section contains *.htm files containing information about the vulnerability, the location of the vulnerability, and gives suggestions on how to patch the vulnerability.

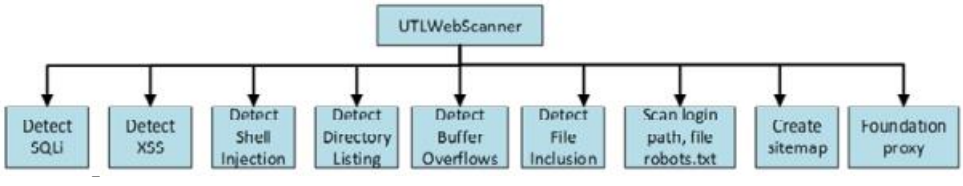
ISSN 0315-3681 Volume 122 , 2025

5.2 The UTLWebScanner tool

We use the Java language to create the UTLWebScanner vulnerability scan tool, which enables them to run multi-platform on windows, Linux, or Mac OS X operating systems. The tool consists of nine modules, each of which has its function, as shown in Figure 7.

- *SQLi, XSS, SI, directory listing, BoF, FI detector module:* detect and identify vulnerabilities on the website; classify danger levels and make suggestions for remedies.
- *Scanning module for the login path and the robots.txt file:* search for the login path of the website and detect the robots.txt file of the website.
- *Create sitemap:* search for all directories and files of the website to make a sitemap. The sitemap helps the tool identify the website structure for scanning. Besides, the information in the robots.txt file is also searched to assist in this process.
- *Foundation proxy:* install proxy server or change IP address, intercept the package. This feature is used in case the system needs to use a proxy.

Figure 7 Function decomposition of the scanning tool (see online version for colours)



6.1 The environment and the tool

We conducted experiments of the tools on the system with CPU Intel Core i5-5200U, 12 GB RAM, Windows 10 operating system. The tool is tested and compared with the popular software in the world in the detection of web security vulnerabilities, namely Acunetix, Nessus, and Arachni. It is denoted as Table 2.

Table 2 Name of the tested scanning tools and their descriptions

Abbr	*Scanning tool*	*Description*
AWS	Acunetix web vulnerability scanner	An effective vulnerability scanning software program, which can check for all vulnerabilities on the web, including SQL injection, XSS, and other vulnerabilities.
Ness	Nessus	A proprietary security vulnerability scanner developed by tenable network security.
Ara	Arachni	A full-featured, high-performance framework for compromising and evaluating the security of web applications.
UTL	UTLWebScanner	The tool that our team developed with proposed algorithms and improvements.

ISSN 0315-3681 Volume 122 , 2025

The datasets we use include websites and public datasets that are built to test vulnerability scanning tools described in detail in Table 3.

Table 3 Name of the tested datasets and their descriptions

Abbr	*Address*	*Description*
TESTP	http://testphp.vulnweb.com	The website that was built by Acunetix to test vulnerability scanning tools (Murzaeva and Akleylek, 2021).
WEBSC	https://www.webscantest.com	The website established to test the automated web application scanning tools like AppSpider (Shah and Mehtre, 2023).
ASPNE	http://aspnet.testsparker.com	The webpage is written in the asp.net language containing vulnerabilities to test (Baykara, 2021).
192.16	http://192.168.189.147	The web server we built to test UTLWebScanner.
DEMOT	http://demo.testfire.net	Written by Watchfire to demonstrate the effectiveness of Watchfire in detecting web application vulnerabilities (Divya et al., 2022).
ZEROW	http://zero.webappsecurity.com	The free online banking site, written by Micro Focus Fortify, aiming at demonstrating the functionality and effectiveness of Micro Focus's WebInspect products in detecting and reporting web application vulnerabilities (Singh and Singh, 2018).
TESTA	http://testaspnet.vulnweb.com	The website was built by Acunetix to test web vulnerabilities (Qasaimeh et al., 2018).

The website vulnerabilities that our algorithm discovered are described in Table 4.

Table 4 Symbol of web vulnerabilities

Acronyms	*The name of the vulnerability*
SQLi	SQL injection
XSS	Cross-site scripting
BoF	Buffer overflow
RFI	Remote file inclusion
LFI	Local file inclusion
OSCI	OS command injection
PHPCI	PHP command injection

We consider command injection errors under two variants, OS command injection and PHP command injection.

6.2 Results

First, we tested the tools with the above groups of website datasets. The results of the number of vulnerabilities detected by the UTLWebScanner tool are specifically shown in Table 5.

ISSN 0315-3681 Volume 122 , 2025

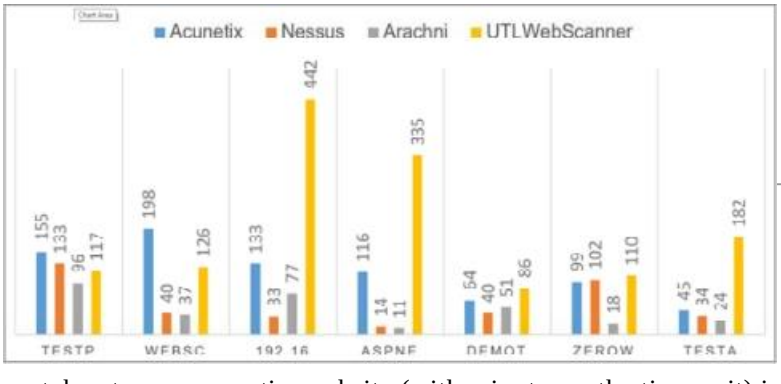
Table 5 Statistics on the number of vulnerabilities detected by UTLWebScanner on the test website datasets

Website / *Vuln*	*SQL*	*XSS*	*BoF*	*RFI*	*OSCI, PHPCI*
TESTP	10	6	0	5	14
WEBSC	16	4	7	3	1
192.16	6	8	2	44	4
ASPNE	2	2	125	1	0
DEMOT	14	10	3	6	3
ZEROW	6	11	7	0	2
TESTA	3	9	3	6	6
OWASP	107	55	22	102	9

The comparisons of scan parameters, such as the number of detected vulnerabilities (DV) and scan time between UTLWebScanner and other commercial software, are listed in Table 5.

The total number of vulnerabilities detected by the software. The experimental results show that the new suggested tool can detect more vulnerabilities in half of the scanned sites, giving better results than the Nessus and Arachni software do. The results are shown in Figure 8.

Figure 8 Comparison of the number of vulnerabilities that can be scanned among the tools (see online version for colours)



The time taken to scan an entire website (with minutes as the time unit) is shown in Figure 9. Experimental results show that the new tool has a much higher scanning speed of the entire site than the commercial version software.

Statistics by the danger level of vulnerabilities scanned by some software programs. Specific results are given in Figure 10, Figure 11, Figure 12, and Figure 13.

ISSN 0315-3681 Volume 122 , 2025

Figure 9 Comparison of the scanning time of the entire website among the tools (see online version for colours)

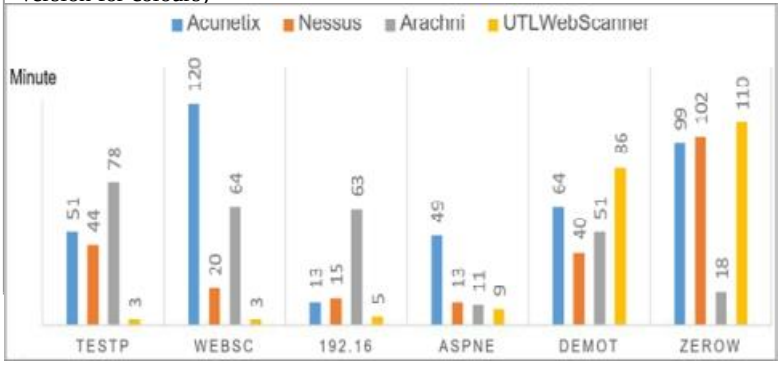


Figure 10 Vulnerability at the high level of danger (see online version for colours)

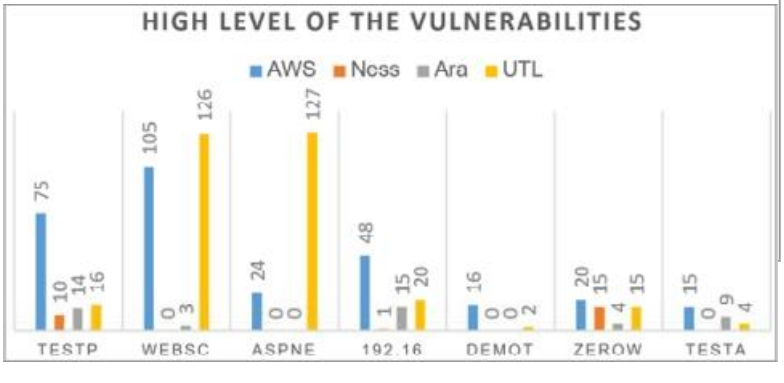
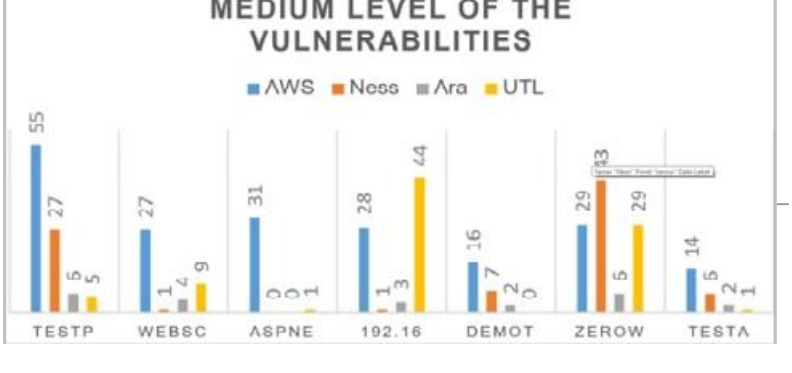


Figure 11 Vulnerability at the medium level of danger (see online version for colours)



ISSN 0315-3681 Volume 122 , 2025

Figure 12 Vulnerability at the low level of danger (see online version for colours)

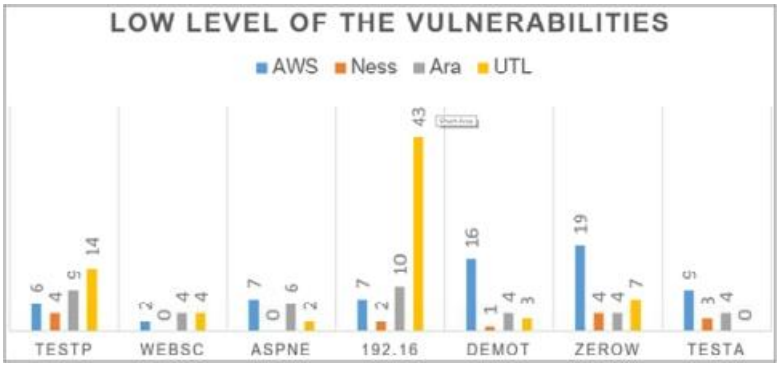
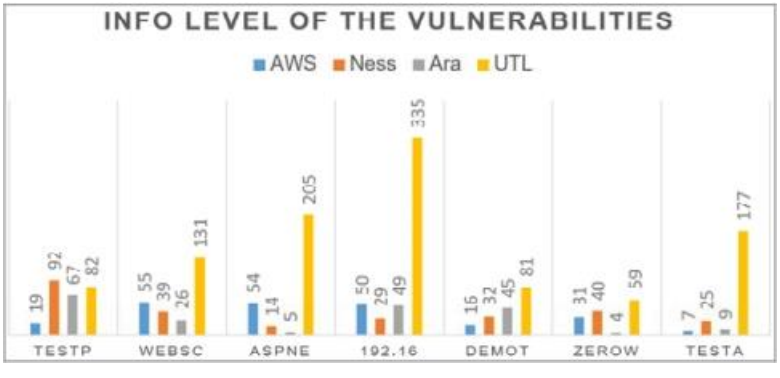


Figure 13 Vulnerability at the info level (see online version for colours)



A detailed report on the vulnerabilities and the number of web application vulnerabilities are described in Table 6. The number of vulnerabilities found in the application and relevant details are reported for each type of vulnerability. Details of the vulnerability include the URL of the web page, the specific form where the vulnerability was found, and the test injection function used for the detection. This information will be useful for web application developers to identify the locations of vulnerabilities and the types of vulnerabilities that need to be addressed. In addition, this is also continuously updating data for the machine learning-based improvement steps that we have covered in Section 4.3 of this article.

6.3 Evaluation and discussion

We evaluated the vulnerability detection results of UTLWebScanner with FP and FN. In Table 7, we calculate the FP and FN parameters, with the vulnerabilities (D) column representing the number of vulnerabilities detected by UTLWebScanner, including detected as the total number of DV, true vulnerabilities detected (TVD) is the number of vulnerabilities detected correctly. Vulnerabilities (E) is the number of vulnerabilities that exist on the test website.

Table 6 A detailed description of the vulnerabilities of the UTLWebScanner tool

ISSN 0315-3681 Volume 122 , 2025

Module	*Vulnerability details*
Name of vulnerability	SQL injection
Classification	Input validation error
Source	http://testphp.vulnweb.com/listproducts.php
Level of danger	High
Test function	http://testphp.vulnweb.com/listproducts.php?cat=1%27
Remedies	Programmers should review the request and response to the code to verify the existence of a vulnerability manually.
	'Minor bug' input can prevent these vulnerabilities. String variables must be filtered for special characters, and numeric types should be checked to make sure they are valid.
	Object-relational mapping eliminates the need for SQL.
Name of vulnerability	Cross-site scripting
Classification	Input validation error
Source	http://testphp.vulnweb.com/search.php
Level of danger	High
Test function	<script>alert(1)</script>
Remedies	Programmers must identify that unreliable data had appropriate filtering. General rules on XSS prevention can be found in the OWASP XSS prevention cheat sheet.
	POST/search.php [searchFor=1.htaccess.aspx-->">'>'" goButton=go]
Name of vulnerability	SQL injection
Classification	Input validation error
Source	http://testphp.vulnweb.com/artists.php
Level of danger	High
Test function	http://testphp.vulnweb.com/artists.php?artist=1%20or%201%27
Remedies	Programmers should review the request and response to the code to verify the existence of a vulnerability manually.
	'Minor bug' input can prevent these vulnerabilities. String variables must be filtered for escape characters, and numeric types should be checked to make sure they are valid.
	Object-relational mapping eliminates the need for SQL.

The result shows that the UTLWebScanner tool can perform effective scanning, especially with fast scan speed. For tested datasets, the tool has an average false alarm rate of 11.87%, the leave-out error rate of 3.23%. It can be seen that the UTLWebScanner tool has a low FN, while the FP percentage is perfectly acceptable because false alarms are still more positive than the omission of errors.

In another experiment, we performed on larger datasets. The results shown in Table 8 and Table 9 show that, with low analysis-scanning time and low FP rate, the UTLWebScanner tool has the ability to detect vulnerabilities with a high accuracy rate in terms of the total number of vulnerabilities and accuracy. The average time to scan seven websites is 4.29 s. Security vulnerabilities mainly focus on SQLi vulnerabilities with an average rate of 17.44%, XSS is 52.32%; the remaining 30.24% is RFI, LFI, and OSCI vulnerabilities.

ISSN 0315-3681 Volume 122 , 2025

Table 7 Calculated results of FP and FN of UTLWebScanner

Datasets	*Vulnerabilities (D)*		*Vulnerabilities (E)*	*False positive (FP)*	*False negative (FN)*
	DV	*TVD*			
TESTP	117	116	120	1	4
WEBSC	126	126	160	0	34
192.16	442	442	442	0	0
ASPNE	335	220	220	115	0
DEMOT	86	89	100	3	4
ZEROW	110	89	98	12	0
TESTA	182	160	160	22	0

Table 8 Statistics of vulnerabilities on some other web applications

Web application	*Detected vulnerability*						
	SQLi	*XSS*	*RFI, LFI*	*OSCI*	*Total*	*FP*	*Analysis time (s)*
DVWA 1.0.7	4	4	3	6	17	8	15
ZiPEC 0.32	3	4	0	0	7	1	2
Mfm 0.13	0	8	0	0	8	3	6
Measureit 1.14	1	11	0	0	12	7	2
SAMATE	3	11	6	0	20	1	1
Peruggia	4	7	6	5	22	0	2
Total	*15*	*45*	*15*	*11*	*86*	*20*	*28*

Table 9 shows the comparison between WAP tool of Medeiros et al. (2016) with our UTLWebScanner tool. Testing on seven websites shows that the UTLWebScanner tool is better as it detects more holes than WAP with a lower FP rate. Vuln Found's real Vul rate is quite high, with 96.4%, while WAP's exact detection rate is 93.41%. The total number of FPs during the use of the UTLWebScanner tool is 6, while the WAP tool is 11.

In summary, the UTLWebScanner tool can scan and detect vulnerabilities on the web much more effectively than commercial software with the same functionality as Acunetix, Nessus, or Arachni under the circumstance in question. This is most clearly shown in the higher number of vulnerabilities found (with the same criteria for the classification of vulnerabilities) and shorter scan time in most test cases. In particular, with the machine learning technology we have applied to optimise the database, UTLWebScanner has achieved amasing vulnerability scanning speed, absolutely better than commercial software.

The average scanning speed of UTLWebScanner is approximately 1/10 compared to the average speed of Acunetix, Nessus, or Arachni. The UTLWebScanner tool is also

capable of detecting the more serious but less common vulnerabilities, such as FI, SI, or buffer overflow, which gain insufficient attention of many programmers or system testers. However, the tool also has limitations: the number of DV is lower than that of Acunetix, Nessus; the incapability of scanning scans many websites at the same time.

Table 9 Comparison table between WAP and UTLWebScanner

Web application	*WAP*			*UTLWebScanner*		
	Founded Vuln	*Real Vuln*	*FP*	*Founded Vuln*	*Real Vuln*	*FP*
Adminer-1.11.0	3	3	0	3	2	1
GTD-PHP	111	111	0	111	111	0
emoncms	15	12	3	15	14	1
PHPLib 7.4	14	14	0	14	13	1
Hotelmis 0.7	7	2	5	7	5	2
Currentcost	4	2	2	4	4	0
Wordpress 2.0	13	12	1	13	12	1
Total	*167*	*156*	*11*	*167*	*161*	*6*

7 Conclusions and development direction

In this article, we have covered some common security vulnerabilities on the web, such as SQL injection, XSS, BoF, FI, SI. Thereby, we propose an algorithm and improvements to enhance the efficiency of the detection of web application vulnerabilities. The algorithms used to build the scanning tool are UTLWebScanner, performing the test, and make comparisons with some commercial software programs with similar functionality as Acunetix, Nessus, and Arachni on standard datasets. The results show that our tool has a high rate of error detection on web applications, and has the advantage of faster scanning time compared to the tools we use.

For further development, we will develop the database of the software, apply machine learning techniques to increase scanning performance, and make predictions about the location of the vulnerability to focus the search on.

.

ISSN 0315-3681 Volume 122 , 2025

References

[1] Ali, N.S. (2018) 'Investigation framework of web applications vulnerabilities, attacks and protection techniques in structured query language injection attacks', *International Journal of Wireless and Mobile Computing*, Vol. 14, No. 2, pp.103–122, DOI: 10.1504/IJWMC.2018. 091137.

[2] Antunes, N. and Vieira, M. (2010) 'Benchmarking vulnerability detection tools for web services', *ICWS 2010 – 2010 IEEE 8th International Conference on Web Services*, pp.203–210, DOI: 10.1109/ICWS.2010.76.

[3] Antunes, N. and Vieira, M. (2011) 'Defending against web application vulnerabilities', *Computer*, Vol. 45, No. 2, pp.66–72, DOI: 10.1109/mc.2011.259.

[4]Antunes, N. and Vieira, M. (2015) 'Assessing and comparing vulnerability detection tools for web services: benchmarking approach and examples', *IEEE Transactions on Services Computing*, Vol. 8, No. 2, pp.269–283, DOI: 10.1109/TSC.2014.2310221.

[5] Bates, D., Barth, A. and Jackson, C. (2010) 'Regular expressions considered harmful in client-side XSS filters', *Proceedings of the 19th International Conference on World Wide Web, WWW'10*, pp.91–99, DOI: 10.1145/1772690.1772701.

[6] Baykara, M. (2018) 'Investigation and comparison of web application vulnerabilities test tools', *Journal of Computer Science and Information Technology*, Vol. 7, No. 12, pp.197–212.

[7] Cowan, C. et al. (2003) 'Buffer overflows: attacks and defenses for the vulnerability of the decade', *Foundations of Intrusion Tolerant Systems, OASIS 2003*, pp.227–237, DOI: 10.1109/FITS. 2003.1264935.

[8] Deepa, G. and Thilagam, P.S. (2016) 'Securing web applications from injection and logic vulnerabilities: approaches and challenges', *Information and Software Technology*, Vol. 74, pp.160–180, DOI: 10.1016/j.infsof.2016.02.005.

[9] Deepa, G., Thilagam, P.S., Khan, F.A. et al. (2018a) 'Black-box detection of XQuery injection and parameter tampering vulnerabilities in web applications', *International Journal of Information Security*, Vol. 17, No. 1, pp.105–120, DOI: 10.1007/s10207-016-0359-4.

[10] Deepa, G., Thilagam, P.S., Praseed, A. et al. (2018b) 'DetLogic: a black-box approach for detecting logic vulnerabilities in web applications', *Journal of Network and Computer Applications*, Vol. 109, pp.89–109, DOI: 10.1016/j.jnca.2018.01.008.

[11] Divya, K.V. et al. (2019) 'Progress in advanced computing and intelligent engineering', *Progress in Advanced Computing and Intelligent Engineering*, Springer, Singapore, DOI: 10.1007/978- 981-13-0224-4.

[12] Djuric, Z. (2013) 'A black-box testing tool for detecting SQL injection vulnerabilities', *2013 2nd International Conference on Informatics and Applications, ICIA 2013*, pp.216–221, DOI: 10.1109/ICoIA.2013.6650259.

[13] Dong, G. et al. (2014) 'Detecting cross site scripting vulnerabilities introduced by HTML5', *2014 11th Int. Joint Conf. on Computer Science and Software Engineering: 'Human Factors in Computer Science and Software Engineering' – e-Science and High Performance Computing: eHPC, JCSSE*, pp.319–323, DOI: 10.1109/JCSSE.2014.6841888.

[14] Doupé, A. et al. (2012) 'Enemy of the state: a state-aware black-box web vulnerability scanner', *USENIX Security Symposium*, pp.523–538 [online] https://www.usenix.org/conference/ usenixsecurity12/technical-sessions/presentation/doupe (accessed 18 July 2019).

[15] Fonseca, J. et al. (2014) 'Analysis of field data on web security vulnerabilities', *IEEE Transactions on Dependable and Secure Computing*, Vol. 11, No. 2, pp.89–100, DOI: 10.1109/TDSC. 2013.37.

[16] Fonseca, J., Vieira, M. and Madeira, H. (2007) 'Testing and comparing web vulnerability scanning tools for SQL injection and XSS attacks', *Proceedings – 13th Pacific Rim International Symposium on Dependable Computing, PRDC*, pp.365–372, DOI: 10.1109/PRDC.2007.63.

[17] Foster, J.C. et al. (2005) 'Buffer overflow attacks', DOI: 10.1016/B978-1-932266-67-2.X5031-2.

[18] Goswami, S. et al. (2017) 'An unsupervised method for detection of XSS attack', *International Journal of Network Security*, Vol. 19, No. 5, pp.761–775, DOI: 10.6633/IJNS.201709. 19(5).14.

[19] Gupta, M.K., Govil, M.C. and Singh, G. (2015) 'Predicting cross-site scripting (XSS) security vulnerabilities in web applications', *Proceedings of the 2015 12th International Joint Conference on Computer Science and Software Engineering, JCSSE*, pp.162–167, DOI: 10.1109/JCSSE.2015.7219789.

[20] Gupta, S. and Gupta, B.B. (2016) 'Enhanced XSS defensive framework for web applications deployed in the virtual machines of cloud computing environment', *Procedia Technology*, Vol. 24, pp.1595–1602, DOI: 10.1016/j.protcy.2016.05.152.

[21] Gupta, S., Gupta, B.B. and Chaudhary, P. (2018) 'Hunting for DOM-based XSS vulnerabilities in mobile cloud-based online social network', *Future Generation Computer Systems*, Vol. 79, pp.319–336, DOI: 10.1016/j.future.2017.05.038.

[22] Hassan, M.M. et al. (2018) 'SAISAN: an automated local file inclusion vulnerability detection model', *International Journal of Engineering and Technology (UAE)*, Vol. 7, No. 2, pp.4–8, DOI: 10.14419/ijet.v7i2.3.9956.

[23] Hydara, I. et al. (2015) 'Current state of research on cross-site scripting (XSS) – a systematic literature review', *Information and Software Technology*, Vol. 58, pp.170–186, DOI: 10.1016/j.infsof.2014.07.010.

[24] Kaur, D. and Kaur, P. (2016) 'Empirical analysis of web attacks', *Physics Procedia*, Vol. 78, pp.298–306, DOI: 10.1016/j.procs.2016.02.057.

[25] Liban, A. and Hilles, S.M.S. (2014) 'Enhancing Mysql injector vulnerability checker tool (Mysql Injector) using inference binary search algorithm for blind timing-based attack', *Proceedings – 2014 5th IEEE Control and System Graduate Research Colloquium, ICSGRC*, pp.47–52, DOI: 10.1109/ICSGRC.2014.6908694.

[26] Lin, J.C. and Chen, J.M. (2007) 'The automatic defense mechanism for malicious injection attack', *CIT 2007: 7th IEEE International Conference on Computer and Information Technology*, pp.709–714, DOI: 10.1109/CIT.2007.4385168.

[27] Martin, M. and Lam, M.S. (2008) 'Automatic generation of XSS and SQL injection attacks with goal-directed model checking', *USENIX Security Symposium*, pp.31–43 [online] http://portal. acm.org/citation.cfm?id=1496714 (accessed 6 July 2019).

[28] Medeiros, I., Neves, N. and Correia, M. (2016) 'Detecting and removing web application vulnerabilities with static analysis and data mining', *IEEE Transactions on Reliability*, Vol. 65, No. 1, pp.54–69, DOI: 10.1109/TR.2015.2457411.

[29] Mohammadi, M., Chu, B. and Lipford, H.R. (2017) 'Detecting cross-site scripting vulnerabilities through automated unit testing', *Proceedings – 2017 IEEE International Conference on Software Quality, Reliability and Security, QRS*, pp.364–373, DOI: 10.1109/QRS.2017.46.

[30] Murzaeva, A. and Akleylek, S. (2018) 'An automated vulnerable website penetration', *International Conference on Advanced Technologies, Computer Engineering and Science (ICATCES'18)*, Safranbolu, Turkey, 11–13 May, pp.297–301.

[31] Open Web Application Security Project (2016) *OWASP Broken Web Applications Project* [online] https://www.owasp.org/index.php/OWASP_Broken_Web_Applications_Project (accessed 28 June 2019).

[32] Qasaimeh, M., Shamlawi, A. and Khairallah, T. (2018) 'Black box evaluation of web application scanners: standards mapping approach', *Journal of Theoretical and Applied Information Technology*, Vol. 96, No. 14, pp.4584–4596.

[33] Ruse, M., Sarkar, T. and Basu, S. (2010) 'Analysis & detection of SQL injection vulnerabilities via automatic test case generation of programs', *Proceedings – 2010 10th Annual International Symposium on Applications and the Internet, SAINT*, pp.31–37, DOI: 10.1109/SAINT. 2010.60.

.

[34] Shah, S. and Mehtre, B.M. (2015) 'An automated approach to vulnerability assessment and penetration testing using net-nirikshak 1.0', *Proceedings of 2014 IEEE International Conference on Advanced Communication, Control and Computing Technologies, ICACCCT*, No. 978, pp.707–712, DOI: 10.1109/ICACCCT.2014.7019182.

[35] Singh, S. and Singh, K. (2018) 'Performance analysis of vulnerability detection scanners for web systems', *Advances in Intelligent Systems and Computing*, Springer, Singapore, DOI: 10.1007/ 978-981-10-8536-9_37.

[36] Su, Z. and Wassermann, G. (2006) 'The essence of command injection attacks in web applications', *ACM SIGPLAN Notices*, Vol. 41, No. 1, pp.372–382, DOI: 10.1145/1111320. 1111070.

[37] Tajbakhsh, M.S. and Bagherzadeh, J. (2015) 'A sound framework for dynamic prevention of local file inclusion', *2015 7th Conference on Information and Knowledge Technology, IKT*, DOI: 10.1109/IKT.2015.7288798.

[38] Wang, S., Zhu, X. and Yang, F. (2014) 'Efficient QoS management for QoS-aware web service composition', *International Journal of Web and Grid Services*, Vol. 10, No. 1, pp.1–23, DOI: 10.1504/IJWGS.2014.058763.

[39] Xu, J. et al. (2016) 'Run-time resolution of service property conflicts in web service composition', *International Journal of Web and Grid Services*, Vol. 12, No. 2, pp.142–161, DOI: 10.1504/IJWGS.2016.076617.