A Vulnerability Detection Model For Web Applications

Bodunde Akinyemi
Department of Computer Science and Engineering
Obafemi Awolowo University
Ile-Ife, Nigeria
bakinyemi@oauife.edu.ng

***Abstract*— Web application security is critical with today's rapidly growing dependency on web applications. Because of the increase in security flaws in web applications, there is an urgent need to develop robust and efficient web application scanners. Most existing web application scanners only focus on SQL injection and cross-site scripting and underestimate vulnerabilities like HTTP header injection and web server domain file disclosure, thus leading to high false-positive or false-negative results. This study formulated a vulnerability detection model by employing a tainted mode and a penetration testing model. The tainted mode model was used to map an HTTP request made to a web application in its initial state to the resulting HTTP response and the state of the web application after the request was made. The penetration testing model was used to fuzz a working instance of a web application and also to check for security vulnerabilities that contained signatures of HTTP header injection, file disclosure, or directory listing. The implementation was done using the Java programming language to design a multithreaded web crawler and a test module for fuzz processes. The proposed model was evaluated using a sample web application's entry point (a link or url) as a test bed. Its performance was assessed using accuracy, precision, and recall as metrics. The result showed that the proposed model yields fewer false-positive results for the selected vulnerabilities. Network administrators can use the proposed model to support quality assurance testing of a web application prior to deployment.**

Keywords—Security, Vulnerability, HTTP header injection, Web applications, file disclosure

I. Introduction

As more and more vital data and information are stored on the web due to the increase in the usage of web resources across various disciplines, the number of transactions on the web and the need for its usage have increased seamlessly. The evolution of web technologies and the substantial innovation in the field has led to both a huge rise in productivity as well as security vulnerabilities [1]. The emergence of new technological possibilities opens up a number of application products and much more effective and suitable approaches to performing tasks, but it also gives rise to the potential for technology exploitation. The web applications that are currently employed by various users have the potential to either help them operate smoothly and productively or result in countless hours of irritation and lost productivity. Some web applications contain sensitive information, such as financial transaction information. As web applications have grown, so have their vulnerabilities [2]. These vulnerabilities are flaws that an attacker can leverage to exploit the system. Day in and day out, new security vulnerabilities are found in widely known applications. In recent times, web applications have become the primary targets of attacks[3]. It is expedient that proper security web applications be put in place.

Web applications, the processes that run on them, and the information disseminated and stored electronically all have long-standing security issues[4]. Web application security is one of the most important security measures to implement now that web application reliance has shifted from personal to business and organizational use. Most of the security flaws and vulnerabilities in web applications presently are due to oversight on the part of the developers during the development process or poor development turnaround. The use of coding guidelines, security reviews of the code, penetration tests, code vulnerability analyzers, etc. are all necessary for preventing vulnerabilities. By strictly enforcing secure development principles, web application security can be improved [5]. This can also be achieved in other ways because of the array of development languages and libraries that are considerably more advanced, offering routines that are already well-tested and can be used in applications. Security testing is, therefore, a significant part of testing web applications.

The convenience of use, tremendous quality of automation, and autonomy provided by the technologies of the web application have all contributed to the growth of web application scanners. In many cases, web application scanners are positioned as "point-and-click" pentesting instruments [3]. Web application scanners crawl through the pages of a web application to check for vulnerabilities through the use of attack simulation. The tools also seek out software coding flaws like invalid input strings and buffer overflows, along with vulnerabilities specific to web applications.

Several web application scanners have been implemented in the past, most of which focus on SQL injection and cross-site scripting (XSS). XSS typically attacks the web browser on the client side, whereas SQL injection, a similar web vulnerability, is directly implicated on the server side [2]. These available security scanners overlook vulnerabilities like HTTP header injection, usually known as Carriage Return and Line Feed (CRLF) injection, and web server domain file disclosure and directory listing. Other scanners produce false-positive or false-negative results when scanned against these vulnerabilities.

This paper attempts to provide a tool for exposing poor development and an automated and dynamic security analysis tool that produces fewer false-positive or false-negative results on HTTP header vulnerabilities and file disclosure or directory listing vulnerabilities.

II. RELATED WORKS

There are several approaches to scanning a web application for vulnerabilities; most of these approaches incorporate a fault injection technique that uses real-life vulnerability test cases to inject the Data Entry Point (DEP) of a web application. According to the Open Web Application Security Project (OWASP), manual code review is the most efficient way to discover web application security vulnerabilities. This technique is time-consuming and requires the expertise of a professional. It is also known to be prone to overlooking errors due to the exhaustiveness of the technique. Although identifying and resolving vulnerabilities in web applications is the most important means of web application security, there are other approaches as well, such as secure development, the use of intrusion detection and/or prevention systems, and the use of web application firewalls [6].

Several web application scanners have been implemented in the past, taking into account different categories of vulnerabilities. One of the most prevalent and significant vulnerabilities in web environments is SQL injection, which has drawn a lot of interest from researchers and software developers. SQL injection vulnerabilities are crucial for web services as they are closely related to the way the service's code is constructed. Various studies have proposed a number of SQL injection vulnerability detection mechanisms [1], [4], and [7–16]. Aside from web-based systems, a SQL injection attack detection model has been developed for other applications, such as cloud applications, particularly SaaS applications, because they are vulnerable to most common web attacks [17], network flow data [18], and mobile applications.

Cross-site scripting is also another category of web vulnerability that attackers have explored. It is a kind of scripting code that is injected into the flexibly resulting pages of trusted websites in order to transmit sensitive information to an unknown source (i.e., the attacker's server) without being noticed by a relatively similar policy or cookie safeguard mechanism, allowing attackers to access private information. Much effort has been put in place to curb this menace [19–25]. Also, some researchers focused on varieties of vulnerabilities, for example, cross-site scripting and SQL injection vulnerabilities [26] and cross-channel scripting and code injection attacks on cloud-based applications [27]. Also, a web application injection tool for attacks and vulnerabilities based on the controlled injection of real vulnerabilities and a subsequent attack on the system using those weaknesses [28], and so on.

The major drawback of most existing security scanners is their inability to cover the ranges of several other vulnerabilities, produce fewer false positives, and provide a user interface with which a common man with little or no knowledge of security scanning can easily access these tools.

In this study, an attempt is made to develop a security scanner focusing on two types of web application vulnerabilities that result from poor turn-around during the development of a particular web application. These vulnerabilities are: HTTP header injection, such as the carriage return line feeder vulnerability (i.e. CRLF injection), and Web application domain server file disclosure or directory listing. The existing security scanners provide a manual interface for accessing these vulnerabilities; therefore, anyone using these tools needs expertise in web application security assessment. The proposed security scanner would be an automated tool for accessing these vulnerabilities, which have been overlooked by most of the earlier developed security scanning tools.

III. METHODOLOGY

The study's primary objective is to develop a vulnerability detection system that covers HTTP header injection and web application domain server file disclosure or directory listing. The conceptual model is depicted in Fig. 1.

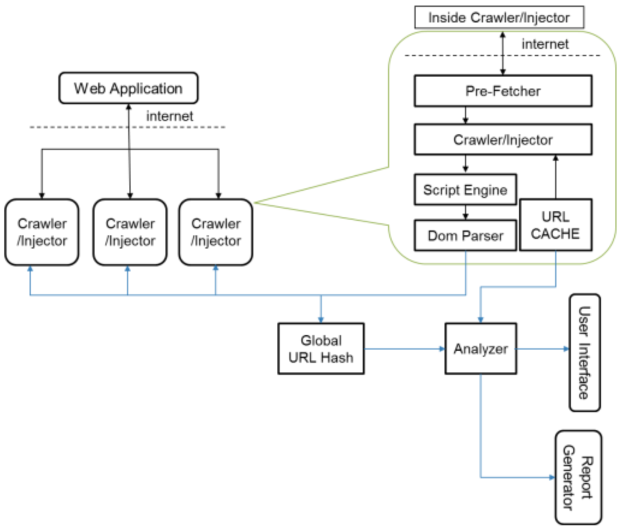


Fig. 1: A Conceptual model of the proposed system

The proposed web application scanner consists of three major modules: a web crawler module, an attacker or injection module, and an analysis module. The HTML document parsing was used to extract the URL from the HTML document file, and the injection module utilised the socket connection to issue a raw HTTP request. The three key modules are described as follows:

A. Web Crawler Module

The web crawler segment is populated with a collection of URLs. It then receives the related pages, explores the links, and redirects to discover all the web application's accessible pages. Additionally, the crawler recognises all of the web application's entry points, such as GET request parameters, input fields of the HTML form, and controls for file uploads.

B. Attack or Injection Module

The attacker module examines the input points and URLs that the crawler has detected. The attacker component then constructs values for each input that are likely to trigger a vulnerability, and for each type of vulnerability, it runs the web application vulnerability scanner. The attacker segment would attempt to exploit the HTTP header injection vulnerability by manipulating the web application into adding extra HTTP headers to legitimate HTTP responses or injecting a CRLF sequence into the response to add or modify existing data, including headers and even the entire response body.

Additionally, the attacker may force information disclosure from the web server domain by having to interact with the website in unusual or malicious ways and then studying the website's responses to look for interesting behaviour, such as directly specifying the names of database tables or columns in error messages, exposing hidden directories' names, structures, and contents via directory listing, and providing temporary backup copies of source code files for access.

C. Analysis Module

The analysis component analyses the pages the web application returns in response to attacks launched by the attacker module in order to find potential problems and notify the other modules. The analysis module determines the presence of an HTTP header injection and a web application domain server file disclosure or directory listing vulnerability.

IV. PROPOSED WEB APPLICATION VULNERABILITY DETECTION MODEL

A vulnerability detection model was formulated as a hybrid model integrating the following models:

- A tainted mode model that maps an HTTP request issued to a web application in its initial state, the resulting HTTP response, and the state of the web application after the request was made; and
- A penetration testing model that fuzzes a working instance of a web application to check for security vulnerabilities that contain signatures of HTTP header injection and file disclosure or directory listing.

The modelling process adopted in the development of the system is described as follows:

A. Tainted mode model

The taint mode model is employed for code security to make a web application more selective about the data it receives from external sources like users, file systems, web environments, local data, other programmes, and various system calls. The tainted mode model is expressed as follows:

A web application is described as the mapping from a request and its current state to the response, the Data Dependency Graph, and the new state as shown in Equations (1) and (2):

$$W: Req \times State \rightarrow DDG \times Response \times State \quad (1)$$

$$DDG = (V, E) \quad (2)$$

Where:

Req is the HTTP request submitted to the web application.

State is the left part refers to the web application state, which is made up of the elements of the web application context (i.e., database, file system, LDAP, etc.)

Response represents the HTTP response that the web application provided.

DDG is the Data Dependency Graph (sometimes referred to as the Program Dependency Graph), which depicts the web application's execution and data flow path.

To more precisely quantify the degree of security vulnerability caused by poor (or absent) input validation, the following assumptions were established:

- All client-provided data obtained through HTTP requests is considered incredible (or tainted).
- All locally generated data for a web application is regarded as credible (or untainted), and internal attacks do not exist. The only kind of communication, however, is HTTP requests and responses.
- Any flawed data can be rendered credible using special sanitization.

B. Penetration testing approach

The methodology used for penetration testing is based on simulated attacks on web applications. The steps involved in this simulation are as follows:

- Identification of all web pages as being part of the web application. Attacks are launched against recognised application Data Entry Points (DEPs) during this phase. This task can be accomplished manually by recording it with a proxy, automatically using web crawlers, or semi-automatically.
- Extraction of DEPs from web pages. The outcome is a collection of DEPs that can be evaluated.
- Attack simulation is commonly referred to as "fuzzing." Every DEP parameter is utilised in an HTTP request to a web application that has been fuzzed with malicious patterns.
- Scanning each HTTP response that is obtained for evidence of vulnerabilities

Black box testing is used to implement penetration testing. Using the best injection pattern on a web application, fault injection techniques are used to find vulnerabilities.

The sequence and class diagram of the proposed model are depicted in Figs. 2 and 3. The activities are described as follows:

The analyst (web app tester) enters the web host URL of the web application into a web browser already configured to listen on an intercepting proxy of the proposed system. The proxy intercepts the web request and forwards it to the web spider, which crawls the URL to retrieve corresponding URLs to all web applications' pages and subpages. The fetched page's URLs are saved in a queue, then forwarded to the vulnerability scanner/injection module for vulnerability testing. In the vulnerability scanner module, the URLs are fuzzed with test cases of the project vulnerabilities; the fuzzed URLs are then used to make HTTP requests from the web application. To detect the presence of a vulnerability, the HTTP response corresponding to the HTTP request is scanned for expected and unexpected signatures. The result of the scan is returned to the web browser that made the original request for ease of use and proper presentation.

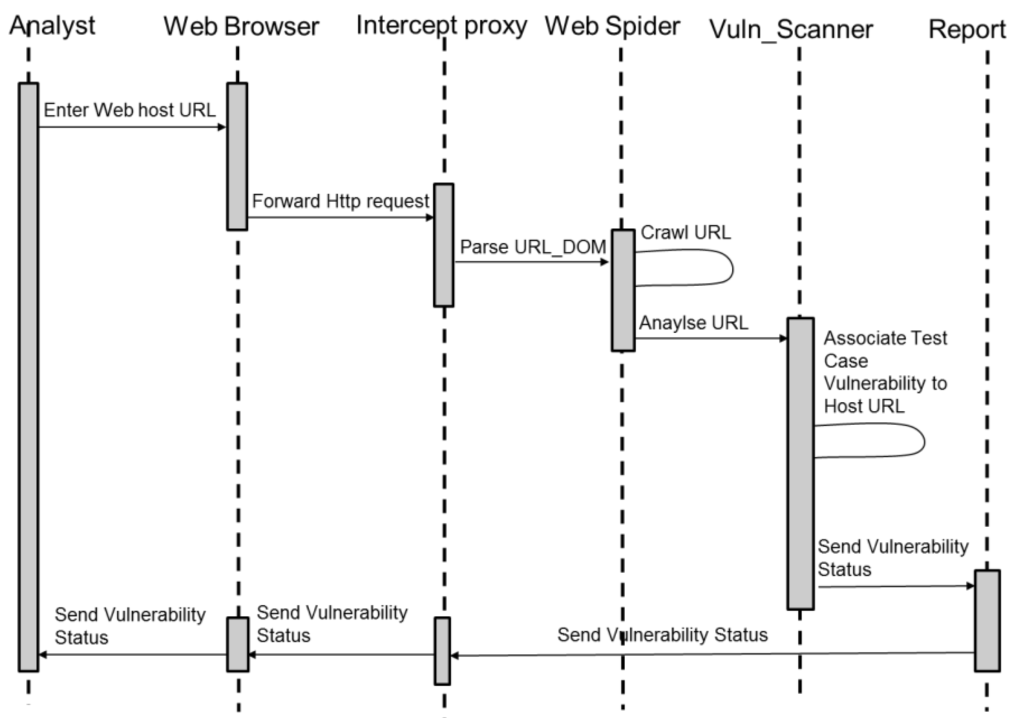

Analyst
Web Browser
Intercept proxy
Web Spider
Vuln_Scanner
Report
Enter Web host URL
Forward Http request
Parse URL_DOM
Crawl URL
Anaylse URL
Associate Test Case Vulnerability to Host URL
Send Vulnerability Status
Send Vulnerability Status
Send Vulnerability Status
Send Vulnerability Status

Fig. 2: Event Diagram of the proposed model

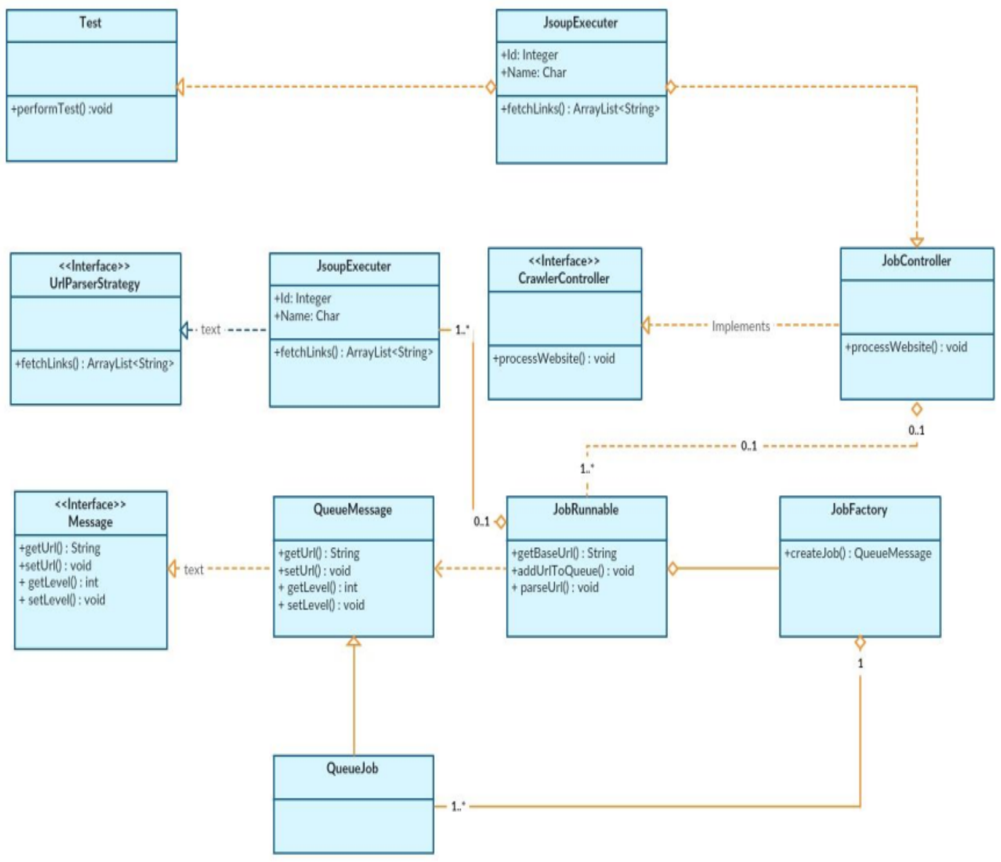


Fig. 3: Class Diagram of the proposed model

V. SYSTEM IMPLEMENTATION

The proposed web application vulnerability evaluation system was implemented using the Java Programming Language. The proposed model implemented three different modules: the web crawler module, the test/injector module, and the interface of the designed model.

The web crawler module was implemented using a multi-threaded process for speedy crawling using the Java *ThreadExecutorService* class with a maximum number of eight threads to handle the crawling and effective computer memory management as memory usage increases during crawling. The web crawler is also integrated with an HTML document parser for effective parsing of HTML tags and prefetching the links embedded in the HTML document, which are further added to a queue from which the crawling scheduler picks the next URL to crawl. The Java *Jsoup* API was employed for parsing the HTML document as needed.

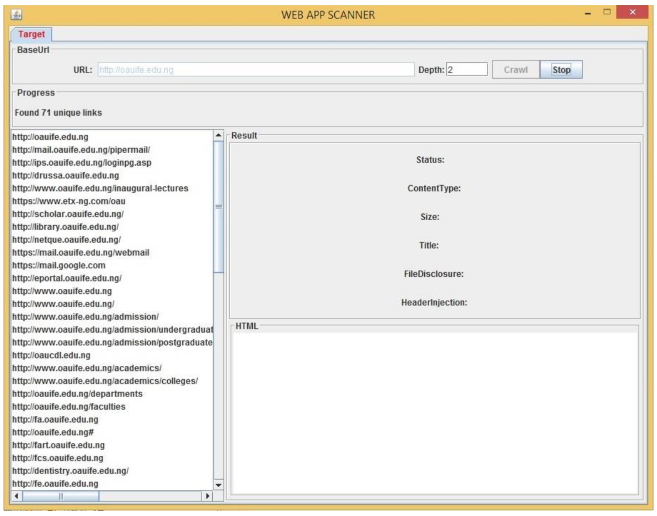
The test/injection module consists of the file disclosure test and the header injection test. The file disclosure test was implemented by parsing an HTML document returned by a URL using the *Jsoup* API to fetch the HTML title tag and check if its contents contain "index of," which is the signature that denotes file disclosure on a web application. If the title tag has "index of," the test is successful. The header injection test was implemented by posting a modified raw HTTP request to the output stream of the socket connection created with the hostname of the web application to be tested. The corresponding HTTP response to the HTTP request made is retrieved by reading input from the input stream of the socket connection created earlier. If the HTTP response code from the read input stream returns "200 ok" or "404 not found," then the test was successful.

The interface for the web application vulnerability evaluation system was implemented using the *Java Swing* API. Among the interfaces created are:

- The Base URL Text Field: This is where the user will input the web application URL to crawl
- The Crawler depth Text Field: This is where the user will input the desired depth of the crawl.
- The Crawl Button: This button initiates the start process for the crawl.
- The Stop Button: This button initiates the termination of an initiated crawling process.
- The Crawl Result View: This is where the crawled URL results are appended.
- The Test/Injection Result View area: This view area consists of components that display the result of the Test/Injection performed on a URL in the Crawl Result View. The Test or Injection action is initiated on clicking a URL in the Crawl Result View List.

VI. RESULTS AND DISCUSSIONS

The implemented system was tested with the URL of Obafemi Awolowo University, Ile-Ife, Nigeria (i.e., http://oauife.edu.ng) to identify the two vulnerabilities under consideration. Fig. 4 shows the snapshot of the web crawler initiation to "http://oauife.edu.ng" with a depth of 2. Fig. 5 shows the initiation of a file disclosure or header injection vulnerability evaluation of "http://ips.oauife.edu.ng/loginpg.asp".

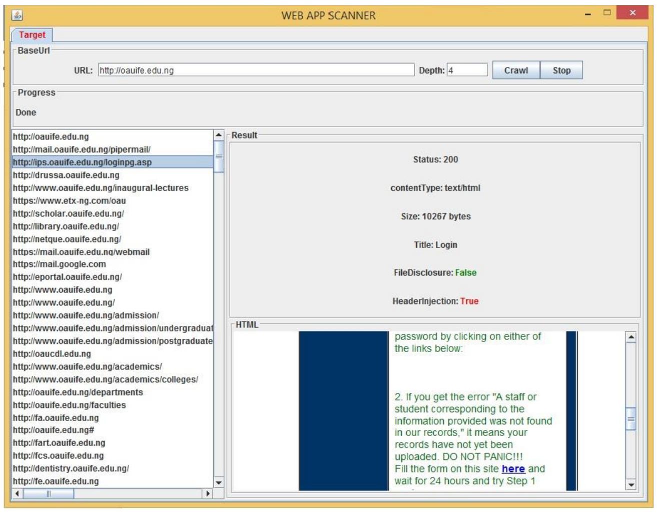


```
URL: http://oauife.edu.ng    Depth: 2
Found 71 unique links

http://oauife.edu.ng
http://mail.oauife.edu.ng/pipermail/
http://ips.oauife.edu.ng/loginpg.asp
http://drussa.oauife.edu.ng
http://www.oauife.edu.ng/inaugural-lectures
https://www.etx-ng.com/oau
http://scholar.oauife.edu.ng/
http://library.oauife.edu.ng/
http://netque.oauife.edu.ng/
https://mail.oauife.edu.ng/webmail
https://mail.google.com
http://eportal.oauife.edu.ng/
http://www.oauife.edu.ng
http://www.oauife.edu.ng/
http://www.oauife.edu.ng/admission/
http://www.oauife.edu.ng/admission/undergraduat
http://www.oauife.edu.ng/admission/postgraduate
http://oaucdl.edu.ng
http://www.oauife.edu.ng/academics/
http://www.oauife.edu.ng/academics/colleges/
http://oauife.edu.ng/departments
http://oauife.edu.ng/faculties
http://fa.oauife.edu.ng
http://oauife.edu.ng#
http://fart.oauife.edu.ng
http://fcs.oauife.edu.ng
http://dentistry.oauife.edu.ng/
http://fe.oauife.edu.ng

Status:
ContentType:
Size:
Title:
FileDisclosure:
HeaderInjection:
```

Fig. 4: The Web Crawler initiation to "http://oauife.edu.ng" with depth of 2



```
URL: http://oauife.edu.ng    Depth: 4
Done

http://oauife.edu.ng
http://mail.oauife.edu.ng/pipermail/
http://ips.oauife.edu.ng/loginpg.asp
http://drussa.oauife.edu.ng
http://www.oauife.edu.ng/inaugural-lectures
https://www.etx-ng.com/oau
http://scholar.oauife.edu.ng/
http://library.oauife.edu.ng/
http://netque.oauife.edu.ng/
https://mail.oauife.edu.ng/webmail
https://mail.google.com
http://eportal.oauife.edu.ng/
http://www.oauife.edu.ng
http://www.oauife.edu.ng/
http://www.oauife.edu.ng/admission/
http://www.oauife.edu.ng/admission/undergraduat
http://www.oauife.edu.ng/admission/postgraduate
http://oaucdl.edu.ng
http://www.oauife.edu.ng/academics/
http://www.oauife.edu.ng/academics/colleges/
http://oauife.edu.ng/departments
http://oauife.edu.ng/faculties
http://fa.oauife.edu.ng
http://oauife.edu.ng#
http://fart.oauife.edu.ng
http://fcs.oauife.edu.ng
http://dentistry.oauife.edu.ng/
http://fe.oauife.edu.ng

Status: 200
contentType: text/html
Size: 10267 bytes
Title: Login
FileDisclosure: False
HeaderInjection: True

password by clicking on either of
the links below:

2. If you get the error "A staff or
student corresponding to the
information provided was not found
in our records," it means your
records have not yet been
uploaded. DO NOT PANIC!!!
Fill the form on this site here and
wait for 24 hours and try Step 1
```

Fig. 5: The Initiation of file disclosure/header injection check on http://ips.oauife.edu.ng/loginpg.asp

The network traffic training dataset for this study was gathered using Wireshark application at the cybersecurity laboratory of the African Center of Excellence, Obafemi Awolowo University, Ile-Ife. The setup is a controlled room where inbound and outbound traffic are safe from intrusions. The setting also enables the execution of real life attack scripts.. The statistics of the dataset used for the simulation of the proposed model are given in Table 1.

Table 1: Dataset Statistics

Dataset	Normal Instances	Attack Instances	Sub-total
Training dataset	23,375	22,204	45,579
Testing Dataset	4,906	5,094	10,000
Total	28,281	27,298	55,579

The values for True Positive (TP), False Positive (FP), True Negative (TN), and False Negative (FN) across five (5) simulation iterations are displayed in Table 2, where;

- TP indicates that relevant vulnerabilities have been detected.
- FN indicates that irrelevant vulnerabilities are detected
- FP indicates relevant vulnerabilities that are not detected; and
- TN indicates irrelevant vulnerabilities that are not detected.

Table 2: Detection results of the Proposed model

Iterations	TP	FP	TN	FN
1	76	4	212	13
2	86	11	202	23
3	77	2	191	5
4	78	3	181	9
5	90	6	211	12

The performances of the proposed vulnerability detection model were evaluated using precision, recall, and accuracy described as follows:

- Precision reveals the detection model's ability to detect only the relevant vulnerabilities while attempting to avoid combining them with irrelevant ones. It is evaluated in Equation 3 as follows:

$$Precision\ (P) = TP/(TP+FP) \quad (3)$$

- Recall reveals the detection model's ability to identify relevant vulnerabilities. It is calculated in Equation 4 as follows:

$$Recall\ (R) = TP/(TP+FN) \quad (4)$$

- The detection accuracy reflects the exactness of the detection. It is evaluated using Equation 5.

$$Accuracy = (TP+TN)/(TP+TN+FP+FN) \quad (5)$$

Table 3 and Fig. 6 show the percentage evaluation results of the proposed model. The results show that the proposed model accurately and precisely classifies the selected vulnerabilities. There is a reduction in the false alarm rate. Fig. 7 depicts the Receiver Operating Curve of the model with a 0.88 Area Under the Curve (ROC_AUC). Higher accuracy is shown by the curve being drawn closer to the graph's left boundary.

Table 3: Evaluation results of the proposed model

Iterations	Precision	Recall	Accuracy
1	95.00	85.39	94.43
2	91.49	78.90	90.28
3	97.47	93.90	97.45
4	89.66	89.66	93.50
5	93.75	88.24	94.36

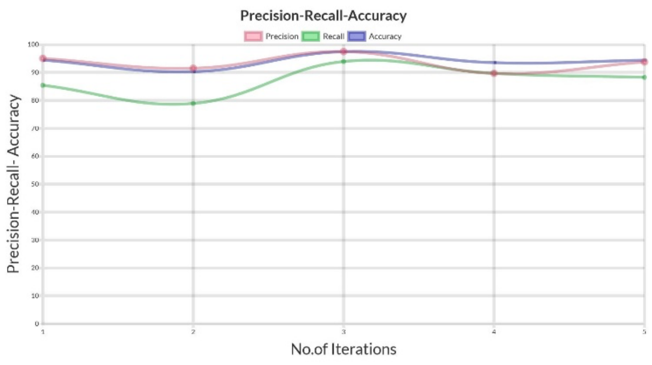


Fig. 6: Evaluation results of the proposed model

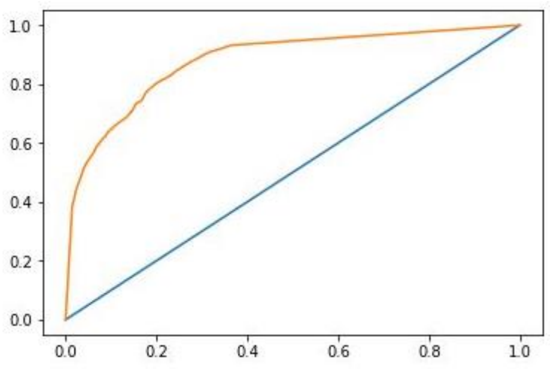


Fig.7. Area under Curve of the proposed vulnerability detection model

VII. CONCLUSIONS

The usefulness of the web application vulnerability evaluation system goes beyond a security test tool, as it serves as an analyst for individuals and product owners with no prior knowledge of web application security test procedures. It is also being used to build a sitemap of the web application, which will be useful for search engine optimization. This study developed a vulnerability detection model that covers HTTP header injection and web application server file disclosure or directory listing. The model performs effectively and efficiently in all measures when used to evaluate a web application through its entry point URL; thus, the system can be leveraged to support quality assurance testing of a web application before deployment. Nonetheless, it is recommended that developers use test-driven development during the development phase of a web application to scrutinize or reduce the possibility of flaws or vulnerabilities during product deployment.

REFERENCES

[1] M.S. Aliero, I. Ghani, and K.N Qureshi, "An algorithm for detecting SQL injection vulnerability using black-box testing, "J Ambient Intell Human Comput, vol. 11, pp. 249–266, 2020.

[2] J. Majumder and G. Saha, "Analysis of SQL Injection Attack," International Journal of Computer Science and Informatics: Vol. 2, Iss. 4, 2013, DOI: 10.47893/IJCSI.2013.1102

[3] E. Fong and V. Okun, "Web Application Scanners: Definitions and Functions," 2007 40th Annual Hawaii International Conference on System Sciences (HICSS'07), Waikoloa, HI, USA, 2007, pp. 280b-280b, doi: 10.1109/HICSS.2007.611.

[4] N. Antunes and M. Vieira, "Detecting SQL Injection Vulnerabilities in Web Services," 2009 Fourth Latin-American Symposium on Dependable Computing, João Pessoa, Brazil, 2009, pp. 17-24, doi: 10.1109/LADC.2009.21

[5] M. Vieira, N. Antunes and H. Madeira, "Using web security scanners to detect vulnerabilities in web services," 2009 IEEE/IFIP International Conference on Dependable Systems & Networks, Lisbon, Portugal, 2009, pp. 566-571, doi: 10.1109/DSN.2009.5270294.

[6] A. Petukhov and D. Kozlov, "Detecting Security Vulnerabilities in Web Applications Using Dynamic Analysis with Penetration Testing, " In Proceedings of the OWASP Application Security Conference, May 19-22, 2008.

[7] N. Antunes, N. Laranjeiro, M. Vieira and H. Madeira, "Effective Detection of SQL/XPath Injection Vulnerabilities in Web Services," 2009 IEEE International Conference on Services Computing, Bangalore, India, 2009, pp. 260-267, doi: 10.1109/SCC.2009.23.

[8] S. Madan and S. Madan, "Shielding against SQL Injection Attacks Using ADMIRE Model," in Computational Intelligence, Communication Systems and Networks, International Conference on, Indore, India, 2009 pp. 314-320. doi: 10.1109/CICSYN.2009.58

[9] A. Ciampa, C. A. Visaggio, and M. D. Penta, " A heuristic-based approach for detecting SQL-injection vulnerabilities in web applications, " SESS '10: Proceedings of the 2010 ICSE Workshop on Software Engineering for Secure Systems, May 2010 Pages 43–49https://doi.org/10.1145/1809100.1809107

[10] A. B. M. Ali, A. I. Shakhatreh, M. S. Abdullah, and J. Alostad, SQL-injection vulnerability scanning tool for automatic creation of SQL-injection attacks, Procedia Computer Science, Vol. 3, 2011, pp. 453-458,

[11] D. Appelt, C.D. Nguyen, L.C. Briand, and N.Alshahwan, "Automated testing for SQL injection vulnerabilities: An input mutation approach," In Proceedings of the 2014 International Symposium on Software Testing and Analysis, San Jose, CA, USA, 21–25 July 2014; pp. 259–269

[12] İ. kara and M. Aydos, "Detection and Analysis of Attacks Against Web Services by the SQL Injection Method," 2019 3rd International Symposium on Multidisciplinary Studies and Innovative Technologies (ISMSIT), Ankara, Turkey, 2019, pp. 1-4, doi: 10.1109/ISMSIT.2019.8932755.

[13] H. Gu, J. Zhang, T. Liu, M. Hu, J. Zhou, T.Wei, and M. Chen,"DIAVA: A Traffic-Based Framework for Detection of SQL Injection Attacks and Vulnerability Analysis of Leaked Data," in IEEE Transactions on Reliability, vol. 69, no. 1, pp. 188-202, March 2020, doi: 10.1109/TR.2019.2925415.

[14] J. Harefa, G. Prajena, A. A. Muhamad, E.V. S. Dewa, S. Yuliandry, "SEA WAF: The Prevention of SQL Injection Attacks on Web Applications," Advances in Science, Technology and Engineering Systems Journal, Vol. 6, No. 2, pp. 405-411, 2021.

[15] Z. Marashdeh, K. Suwais and M. Alia, "A Survey on SQL Injection Attack: Detection and Challenges," 2021 International Conference on Information Technology (ICIT), Amman, Jordan, 2021, pp. 957-962, doi: 10.1109/ICIT52682.2021.9491117.

[16] M. Alghawazi, D. Alghazzawi, and S. Alarifi, "Detection of SQL Injection Attack Using Machine Learning Techniques: A Systematic Literature Review," J. Cybersecurity. Priv., 2022, vol. 2, pp. 764–777, doi: 10.3390/jcp2040039

[17] D. Tripathy, R. Gohil, and T. Halabi, "Detecting SQL Injection Attacks in Cloud SaaS using Machine Learning. In Proceedings of the 2020 IEEE 6th Intl Conference on Big Data Security on Cloud (BigDataSecurity), IEEE Intl Conference on High Performance and Smart Computing (HPSC) and IEEE Intl Conference on Intelligent Data and Security (IDS), Baltimore, MD, USA, 25–27 May 2020; pp. 145–150

[18] I. S. Crespo-Martínez, A. Campazas-Vega, Á. M. Guerrero-Higueras, V. Riego-DelCastillo, C. Álvarez-Aparicio, and C. Fernández-Llamas, "SQL injection attack detection in network flow data," Computers & Security, Vol. 127, 2023, 103093,

[19] G. Wassermann, and Z. Su, "Static detection of cross-site scripting vulnerabilities," ICSE '08: Proceedings of the 30th international conference on Software engineering, May 2008, pp. 171–180, doi: 10.1145/1368088.1368112

[20] T. S. Barhoom and S. N. Kohail, A New Server-Side Solution for Detecting Cross Site Scripting Attack, International Journal of Computer Information Systems, Vol. 3, No. 2, pp. 19-23, 2011.

[21] M.I.P. Salas and E. Martins, "Security Testing Methodology for Vulnerabilities Detection of XSS in Web Services and WS-Security," Electronic Notes in Theoretical Computer Science, Vol. 302, pp. 133-154, 2014.

[22] E. Uma and A. Kannan, "Improved cross site scripting filter for input validation against attacks in web services," Kuwait Journal of Science (KJS), Vol. 41, No. 2, 2014.

[23] G. E. Rodríguez, J. G. Torres, P. Flores, and D. E. Benavides, "Cross-site scripting (XSS) attacks and mitigation: A survey, Computer Networks," Vol. 166, 106960, 2020.

[24] F. M. M. Mokbal, D. Wang, and X. Wang, "Detect Cross-Site Scripting Attacks Using Average Word Embedding and Support Vector Machine," International Journal of Network Security, Vol.24, No.1, pp.20-28, 2022, doi: 10.6633/IJNS.202201 24(1).03.

[25] S. Lee, S. Wi, and S. Son, "Link: Black-Box Detection of Cross-Site Scripting Vulnerabilities Using Reinforcement Learning," WWW '22: Proceedings of the ACM Web Conference, 2022, pp. 743–754, doi.: 10.1145/3485447.3512234

[26] M. K. Gupta, M. C. Govil and G. Singh, "Static analysis approaches to detect SQL injection and cross site scripting vulnerabilities in web applications: A survey," International Conference on Recent Advances and Innovations in Engineering (ICRAIE-2014), Jaipur, India, 2014, pp. 1-5, doi: 10.1109/ICRAIE.2014.6909173.

[27] M. Indushree, M. Kaur, M. Raj, R. Shashidhara and H. Lee, "Cross Channel Scripting and Code Injection Attacks on Web and Cloud-Based Applications: A Comprehensive Review, " Sensors 22, no. 5:1959, 2022, doi :10.3390/s22051959.

[28] J. Fonseca, M. Vieira and H. Madeira, "Vulnerability & attack injection for web applications," 2009 IEEE/IFIP International Conference on Dependable Systems & Networks, Lisbon, Portugal, 2009, pp. 93-102, doi: 10.1109/DSN.2009.5270349.

[29] B.O. Akinyemi, J. B. Adekunle, T. A. Aladesanmi, G A. Aderounmu and B.H. Kamagaté, "An Improved Anomalous Intrusion Detection Model, " FUOYE Journal of Engineering and Technology, Vol. 4, No. 2, pp.81-88,.2019,