An Integrated Approach for Detecting Security Vulnerabilities in Web Applications: A Theoretical Perspective

Richard Amankwah
Presbyterian College of Education
P. O Box 27 Akropong-Akuapem
Ghana

Patrick Kwaku Kudjo
Sch. of Comp. Sci.
Datalink Institute
P.O. Box CO2481

Beatrice Korkor Agyemang
Presbyterian College of Education
P. O Box 27
Akropong-Akuapem
Ghana

Kofi Mensah
Presbyterian College of Education
P. O Box 27 Akropong-Akuapem
Ghana

Bright Brew
Presbyterian College of Education
P. O Box 27 Akropong-Akuapem
Ghana

Samuel Yeboah Antwi
Presbyterian College of Education
P. O Box 27 Akropong-Akuapem
Ghana

ABSTRACT
Software security vulnerability is a flaw in a software product that could compromise the integrity, availability, or confidentiality of a software system. The growth and development of software have brought about a corresponding increase in vulnerabilities, which has necessitated the need to develop software security assurance tool that can detect and prevent these vulnerabilities. Previous studies have suggested both commercial and open source tools such as Ashcan, Web Inspect, Web King, Skipfish, and OWASP ZAP just to mention but a few to help mitigate against this security gaps. However, each of this approach has its merits and demerits in detecting vulnerabilities. As a result, this paper seeks to develop a more proactive approach which is a merger or integration of the strength of existing techniques into one system: An integrated web vulnerability detector scanner: which is a software assurance tool for detecting vulnerabilities in web application. The analysis involves presenting a general overview of web application, web application scanners and web application vulnerabilities. Lastly, we present the theoretical framework for detecting web application vulnerabilities based on the proposed model. The preliminary findings show that the concept is feasible within the domain of vulnerability detection

General Terms
Software Engineering, Information Security

Keywords
Software Vulnerability, Static Analysis, Web Application

1. INTRODUCTION
The growth in technology has influence the use of web application by individuals and organizations in the field of Education, commercial, political, social etc. Most of the aforementioned institutions and organization uses web application such as blogs, social network, web mail, bank etc. which have sensitive information stored in a centralized database. The inevitable use of web application in our daily life have also attracted the attention of hackers and intruders whose aim is to target the weaknesses in these databases and exploit it maliciously making the functioning of most web application inefficient and ineffective. The cause of a number of vulnerabilities exploited by these unscrupulous people stems from design flaws or an implementation bugs [1] [2]. There exist a number of vulnerabilities including command injection, buffer overflow, data manipulation, path manipulation, authentication, session hijacking, cookie misinterpretation, and others [3]. Recently, several empirical studies have proposed varied tools and technique to aid the detection of the aforementioned vulnerabilities. The most widely applied technique or tool is commonly referred to as web vulnerability scanners. Web vulnerability scanners are tools that allow developers and security experts to test applications against security breaches. Additionally, they provide an automatic way to detect vulnerabilities to avoid the manual repetitive and tedious task of inspecting several hundred or even thousands of tests (i.e. source code). Akinetic Web Vulnerability Scanner [4], IBM Rational Ashcan [5], and HP Web Inspect [6] are some of the most widely used commercial web scanners. Aside the commercial web scanners, there are some publicly available web scanners which include Found stone Snigger and fuzzier.

In addition to the aforementioned tools, there are other techniques that have been applied in literature to detect web application vulnerabilities. For example Jovanovic et al. [7] used white box testing by analyzing the source code before it is deployed on a sever. Furthermore, black box testing [8] can also be applied in many ways to detect bugs in web applications. Despite the significant development and growth in the aforementioned techniques, their detection capability is debatable. This is partly due to the fact that, it application requires basic skills and technical know-how. [9]. Hence, we proposed an integrated web vulnerability detector scanner to merge the strengths of the various techniques into one platform for efficient and effective detection of web vulnerabilities. The contributions of this paper are the following:

1. To present an integrated web vulnerability detection scanner
2. To present a general overview of web application architecture.

3. To provide justification for the weaknesses in web application scanners in detecting stored vulnerabilities

The remaining sections of the paper are structured as follows: Section 2 presents a review of related works. Section 3 presents background of the study. Section 4 discusses web scanner and its approaches. Section 5 present details of web application vulnerabilities with specific reference to OWASP most critical vulnerability list. Section 6. Present the theoretical framework for the study Section 7 conclude the study and provides future research directions.

2. RELATED WORK

It is undoubtedly true that the availability of web application in our daily routine activities anywhere and anytime makes it vulnerable. As a result of these and many factors, hackers take advantage of the situation and put in place all sort of dubious plans to have access to sensitive information in order to comprise software systems. We briefly discuss some of the approaches presented in previous studies that aim at resolving the menace.

2.1 Code analysis approach

This approach combines programing code examination and analyses. It also combines two techniques static analysis and runtime monitoring. With this approach the technique surveys the web application and make a guess about what kind of queries application could generate. After studying this they generate some patterns of the legal queries using program analysis. During dynamic analysis, queries which application generates dynamically by using user input, checked using runtime monitoring and observe whether they are according to the statically built patterns or not. If the query model generated during static analysis is more accurate the approach is more successful. The demerit of this technique is that Certain types of changes done in a source code of application could make this step less precise and result in both false positives and false negatives [10]

2.2 Clustering approach

This approach depends on the clustering of response pages generated by the server. It starts with searching all injection points then injects specially crafted request at each point and observes the response pages. It detects the vulnerability by providing the vulnerable input to all injection points. It has the disadvantage of been good for SQL Injection and also generates false positive and cannot find all injection areas [11]

2.3 Proxy-based approach

This approach act between the client and the server, serving as a proxy, one of such is Nixes which detect the cross-site scripting attacks over a client side. It detects the attack manually or sometimes uses automatically generated rules for reduction. We know working of firewall which works at application-level to block and detect malware. Function of Nixes is same as firewall. User can control every connection coming or leaving the local machine. The decision is up to the user whether accept the connection or blocked because firewall prompts the user about mismatched connection [12].this has the disadvantage of springing up false positive due to protection of the unassuming link with no examination of the bugs

2.4 Browser define approach

This approach is deployed to enable the browser to decipher between authentic script and non-authentic scripts. This is done in two-fold where detection of scripts done by the browser is more accurate so that browser can be used to filter the scripts and second is that the programmer of the web application knows scripts that should be executed for proper application functioning so the website can specify the authentic scripts and filter the non-authentic scripts. In this the website inserts a security policy in its pages that fully describes or specifies allowed scripts to run and browser execute these policies i.e. security policies specifies type of a data that server sends to BEEP browsers [13] the demerit of this approach is that some as vector can go round these security policies

2.5 Template matching approach

Template is pre –define route, hence this is where the exact format in which the html tags are executed and define distinctively. The template system allows the code only in define format to be executed and filter all the malicious code injected from the third party website [14]. If we deploy a strong template system on the client machine, we will able to avoid these as attacks. If the developer defines the template wrongly the original content of the web page will be filtered out and that is the disadvantage to this approach

2.6 Path expression approach

One of the main roles of scanners is to capture malicious input injected by user, the role of the Path expression is to intercept XQuery, after parsing identify the user input and separates it from XQuery, which is stored in the XML file after it has been generated. Finally that file would be validated through a schema [15] but it is not fully automated as schema is generated manually and generates large no of false positives.

2.7 Static and dynamic analysis approach

This approach is almost similar to the code analysis approach for SQL injection attack which is based on combination of static and dynamic analysis approach. By examining location of Path statement and version of its contents they identify query. During a training phase valid XPath statements are analyzed and build a pattern of valid queries. At runtime this mechanism verifies all application generated queries with the initially build query pattern. Detect the vulnerability if mismatch is found [16] the disadvantage to this approach is that when the when the application is altered, the new source code structure invalidates existing query identifiers.

3. BACKGROUND OF THE STUDY

3.1 Overview

This section of the paper is dedicated to a brief background of web application to help us comprehend well its attack and security. The Web Application Security Consortium (WASC) [17] defines a web application as “a software application, executed by a web server, which responds to dynamic web page requests over HTTP.” Web application as we see today has really undergone a lot of transformation. It’s started with use of HTML which was used to transform information into visual images. But one disadvantage of this method was its user unfriendliness. As a result, the Common Gateway Interface (CGI) was introduce to improve upon the lapses of HTML. It became the first standard environment which generates dynamic web pages. It must be noted that the use of CGI for website processing is called Web Application [1].after CGI, there are a lot of web application development tools such as PHP, Active Server Pages (ASP), Perl, Java Server Pages (JSP), JavaScript, VBScript, etc. Some of the broad categories of web application technologies are communication protocols, formats, server-side and client-side scripting languages, browser plug-ins, and web server

API and Others framework which are flexible and powerful solution for transforming and managing data within web application as shown in the figure 1

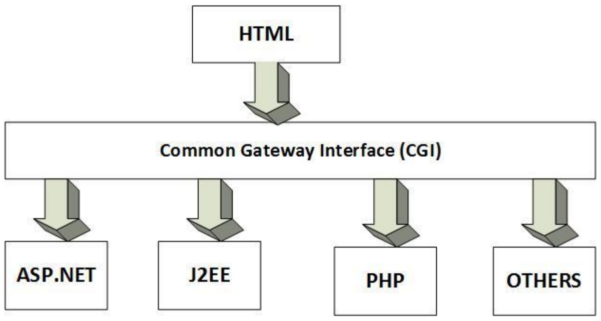

Fig: 1 Flow of development for web application

3.2 Architecture of web application

Web application has a distributed n-tiered architecture. Typically, there is a client (web browser), a web server, an application server (or several application servers), and a persistence (database) server.

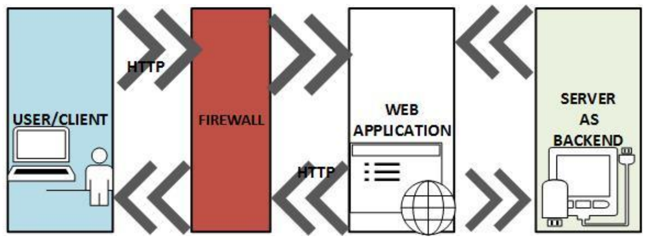

Figure: 2 Simplified view of a web application

4. WEB APPLICATION SCANNERS

A web application scanner is an automated program that examines web applications for security vulnerabilities [18]. In addition to searching for specific vulnerabilities in web applications it performs other functions such as looking for errors in codes of software, illegal input strings and buffer overflow. Web application scanner examines an application by going through its web pages and performs penetration testing - an examination of a web application by simulating attacks on it. This involves coming out with malicious inputs and further evaluation of application's response. Web application scanner performs different types of attack. Web vulnerability scanners consist of three main components: (1) a crawling component (crawling function), (2) an attacker component (fuzzing function), (3) and an analysis component (scraping function) [8]. Basically there are two main approaches [19] to test web application for available vulnerabilities: White box testing: This involves the process of analysis the source code of the web application either manually or using a code analysis tools. The major drawback of this approach is that due to the complexities of most of the codes it may be very hard and difficult to find all bugs in the application. Black box testing: this technique also involves the process of execution the application to look for vulnerabilities. This techniques is normally referred to as penetration testing, the scanner does not know the internals of the web application and it uses fuzzing techniques over the web HTTP requests [20]. Examples of commercial web application scanners: Ashcan [21] , Web King [22], Web Inspect [23] Topsider [24] Others can also be obtain from this references [25] [26] [27]

5. WEB-APPLICATION VULNERABILITIES

In this section, we briefly discuss the Open Web Application Security Project (OWASP) web application vulnerabilities [28].

5.1 Cross-site scripting (XSS) vulnerabilities

This type occurs when an attacker submits malicious data to a web application. Examples of such data are client-side scripts and hyperlinks to an attacker's site. After receiving the data without proper validation within, its generated web pages, it will display the malicious data in a legitimate user's browser. This will render the attacker access to manipulate or steal the credentials of the legitimate user, impersonate the user, or execute malicious scripts on the user's machine. In [29] XSS has been classified again into three main domains which are reflected, stored or DOM-based. This grouping depends on the feedback generated by the server, whether it's as a result of the scam script or save on the sever.

5.1.1 *Reflected or Non-Persistent XSS*: Reflected XSS mostly found in search fields of a web page where the input is get reflected in the output page. When server receives malicious scripts, it does not store in a database, instead it is used to form response pages without any validation.

5.1.2 *Stored or Persistent XSS*: Stored or Persistent XSS occurs when vulnerability lies in server side which allows malicious scripts injected by the attacker store in a database permanently and then references it in a webpage. Blogs, message forums and social networking sites are example where Persistent XSS cause harm to user's browser. Whenever victim visit that site this malicious code is executed in his browser every time. So, it is more dangerous.

5.1.3 *DOM Based XSS*: Document Object Model is nothing but the convention for representing and working with an object in an HTML document. Inappropriate handling of an object with associated DOM makes it vulnerable. Here client-side code itself is vulnerable; vulnerability not lies in the server-side code. Therefore, if we modify the DOM environment, the malicious code is executed in the victim's browser. In a DOM based XSS server does not include the malicious in http response but the client-side code runs itself in an unexpected way due to malicious content. The page remains same but appearance get change. Environment. Both reflected and stored XSS attacks are due to the vulnerability lies in server-side scripts so it handles user input improperly.

5.2 Injection vulnerabilities

The various type of injection that may occur includes data injection, command injection, resource injection, and SQL injection. SQL Injection occurs when a web application does not properly examine user input and places it directly into a SQL statement. This can allow disclosure or modification of data in the database [30]. In terms of attack performance, the OWASP classified SQL injection into the following categories:

5.2.1 Tautology: This attack injects malicious SQL tokens inside where clause and causes conditional query statements always evaluates to true. The main purpose is that to bypass the authentication and access data through vulnerable input field.

5.2.2 Illegal/Logically Incorrect Queries: The main idea is that sending incorrect SQL query purposefully and observe the descriptive error message coming from database and take the advantage of it. This error may contain some useful debugging information which can be used to form further attack.

5.2.3 Union Queries: The Union keyword in SQL can be used to gather information from more than one tables in the database. Injected queries are combined with normal query using union operator. And if used properly database takes the result of the both queries union together and sends to the user.

5.2.4 Piggy-backed Queries: This is the kind of attack where an attacker tries to appends another query to the original legal query by using;(query delimiter). Database treat it as two queries and execute both of them.

5.2.5 Stored Procedure: Stored procedures provide extra layer of protection. Stored Procedures is a group of SQL statements that form a logical unit stored in the database. It provides benefits like encapsulation and strong validation. Even though vulnerability may appear in stored procedures. The vulnerability here is same as in web applications.

5.2.6 Blind Injection: In Blind Injection attacker can send a number of Boolean type queries to gain data.

5.2.7 Timing Attacks: This attack act as a preliminary step be- fore actually performing attack. Initially attacker fire malicious queries and observe the responses. We can use WAITFOR keyword to execute the queries at different times. By observing timing delays between responses attacker can guess sensitive information. This helps them to form a next more dangerous attack.

5.2.8 Alternate Encodings: The main purpose of this attack is escape detection approach. In this technique attacker uses alternate encoding techniques, like ASCII to hide the identity of actually malicious parameters. Cookie poisoning**:** This technique is mainly for achieving impersonation and breach of privacy through manipulation of session cookies, which maintain the identity of the client. By forging these cookies, an attacker can impersonate a valid client, and thus gain information and perform actions on behalf of the victim.

5.3 Invalidated input:

Bugs such as XSS, SQL Injection, and cookie poisoning vulnerabilities are some of the specific instances of this problem. In addition, it includes tainted data and forms, improper use of hidden fields, use of invalidated data in array index, in function call, in a format string, in loop condition, in memory allocation and array allocation.

5.4 Authentication

Authorization and access control vulnerabilities could allow malicious user to gain control of the application or backend servers. This includes weak password management, use of poor encryption methods, use of privilege elevation, and use of insecure macro for dangerous functions, use of unintended copy, authentication errors, and cryptographic errors.

5.5 Incorrect error handling and reporting:

Incorrect error handling and reporting may reveal information thus opening doors for malicious users to guess sensitive information. This includes catch NullPointerException, empty catch block, overly-broad catch block and overly-broad “throws” declaration.

There are other vulnerabilities that happen but may not fall under the above categories such as Denial of service (DoS), Path manipulation, broken session management, Synchronization timing problems

6. THEORETICAL FRAMEWORK

This part of the paper details the proposed approach that integrated the existing web application scanner into a unified model to detect Cross Site Scripting, and XPath Injection attack. This approach is based on techniques from black-box testing, static analysis and dynamic analysis. The merit of this model is to complement the strength and weakness of the scanners. Again, the model provides a reliable module that allows users to easily generate reports all of all the integrated scanners as an output, as compared to the existing scanners where each scanner individually generate reports Fig. 3 depicts the proposed framework.

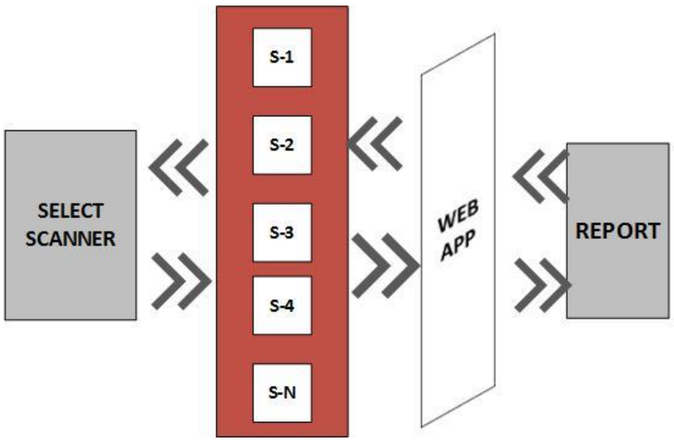

Fig: 3 Proposed Flow Chart

7. CONCLUSION

This study presented a theoretical framework that seeks to unify web application scanners using related concepts from black-box testing, static analysis and dynamic analysis. Firstly, an overview of existing approaches that have been developed by researchers to detect security vulnerabilities in web application was discussed followed by a brief background of web application and its architecture to help us comprehend well its attack and security. The findings from the study suggest that the proposed model is feasible for bug detection in web applications.

8. REFERENCES

[1] S. Patil, N. Marathe, and P. Padiya, "Design of efficient web vulnerability scanner," in *Inventive Computation Technologies (ICICT), International Conference on*, 2016, pp. 1-6.

[2] O. Alhazmi, Y. Malaiya, and I. Ray, "Security vulnerabilities in software systems: A quantitative perspective," in *IFIP Annual Conference on Data and Applications Security and Privacy*, 2005, pp. 281-294.

[3] P. Baral, "Web application scanners: a review of related articles [Essay]," *IEEE Potentials*, vol. 30, pp. 10-14, 2011.

[4] M. Vieira, N. Antunes, and H. Madeira, "Using web security scanners to detect vulnerabilities in web services," in *Dependable Systems & Networks, 2009. DSN'09. IEEE/IFIP International Conference on*, 2009, pp. 566-571.

[5] N. Antunes and M. Vieira, "Detecting SQL injection vulnerabilities in web services," in *Dependable Computing, 2009. LADC'09. Fourth Latin-American Symposium on*, 2009, pp. 17-24.

[6] D. D. Neal and S. S. Rahman, "Securing Systems after Deployment," in *Advances in Computer Science, Engineering & Applications*, ed: Springer, 2012, pp. 685-693.

[7] N. Jovanovic, C. Kruegel, and E. Kirda, "Static analysis for detecting taint-style vulnerabilities in web applications," *Journal of Computer Security,* vol. 18, pp. 861-907, 2010.

[8] S. Kals, E. Kirda, C. Kruegel, and N. Jovanovic, "Secubat: a web vulnerability scanner," in *Proceedings of the 15th international conference on World Wide Web*, 2006, pp. 247-256.

[9] Y. Makino and V. Klyuev, "Evaluation of web vulnerability scanners," in *Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS), 2015 IEEE 8th International Conference on*, 2015, pp. 399-402.

[10] W. G. Halfond and A. Orso, "Preventing SQL injection attacks using AMNESIA," in *Proceedings of the 28th international conference on Software engineering*, 2006, pp. 795-798.

[11] A. Dessiatnikoff, R. Akrout, E. Alata, M. Kaâniche, and V. Nicomette, "A clustering approach for web vulnerabilities detection," in *Dependable Computing (PRDC), 2011 IEEE 17th Pacific Rim International Symposium on*, 2011, pp. 194-203.

[12] E. Kirda, C. Kruegel, G. Vigna, and N. Jovanovic, "Noxes: a client-side solution for mitigating cross-site scripting attacks," in *Proceedings of the 2006 ACM symposium on Applied computing*, 2006, pp. 330-337.

[13] T. Jim, N. Swamy, and M. Hicks, "Defeating script injection attacks with browser-enforced embedded policies," in *Proceedings of the 16th international conference on World Wide Web*, 2007, pp. 601-610.

[14] C. Rajesh, K. Srikanth, I. Sarwani, and G. S. Rao, "A Brief Study on Defining Templates to Avoid XSS Vulnerabilities Using Auto Escape Templates for Web Applications," *IJCSIT) International Journal of Computer Science and Information Technologies,* vol. 6, 2015.

[15] V. Shanmughaneethi, R. Ravichandran, and S. Swamynathan, "PXpathV: Preventing XPath Injection Vulnerabilities in Web Applications," *International Journal on Web Service Computing,* vol. 2, p. 57, 2011.

[16] D. Mitropoulos, V. Karakoidas, and D. Spinellis, "Fortifying Applications Against Xpath Injection Attacks," *MCIS,* vol. 2009, p. 4th, 2009.

[17] E. Fong and V. Okun, "Web application scanners: definitions and functions," in *System Sciences, 2007. HICSS 2007. 40th Annual Hawaii International Conference on*, 2007, pp. 280b-280b.

[18] F. Elizabeth and O. Vadim, "Web application scanners: Definitions and functions," *HICSS 2007,* pp. 280b-280b, 2007.

[19] C. Ghezzi, M. Jazayeri, and D. Mandrioli, *Fundamentals of software engineering*: Prentice Hall PTR, 2002.

[20] M. Sutton, A. Greene, and P. Amini, *Fuzzing: brute force vulnerability discovery*: Pearson Education, 2007.

[21] E. F. R. G. V. Okun, P. E. Black, and E. Dalci, "Building a Test Suite for Web Application Scanners."

[22] O. Hamed and N. Kafri, "Performance Prediction of Web Based Application Architectures Case Study: .NET vs. Java EE," *International Journal of Web Applications,* vol. 1, 2009.

[23] J. C. Fonseca, M. Vieira, and H. Madeira, "Correlating security vulnerabilities with software faults," 2007.

[24] H. Le and P. Loh, "Unified approach to vulnerability analysis of web applications," in *AIP Conference Proceedings*, 2008, pp. 155-159.

[25] P. E. Black and E. Fong, "Proceedings of Defining the State of the Art in Software Security Tools Workshop," *NIST Special Publication,* vol. 500, p. 264, 2005.

[26] S. Panguluri, W. Phillips, and P. Ellis, "Cyber security: protecting water and wastewater infrastructure," in *Handbook of water and wastewater systems protection*, ed: Springer, 2011, pp. 285-318.

[27] A. J. Evans, "Software Security Quality: Testing Taxonomy and Testing Tools Classification," *Presentation viewgraph for OWASP APPSec DC,* 2005.

[28] A. Makkar and K. Jain, "

Web application in healthcare: a solution to address the security issues," *International Journal Of Management & Behavioural Sciences (IJMBS)*.

[29] A. Singh and S. Sathappan, "A Survey on XSS web-attack and Defense Mechanisms," *International Journal of Advanced Research in Computer Science and Software Engineering,* vol. 4, pp. 1160-1164, 2014.

[30] A. Tajpour, S. Ibrahim, and M. Sharifi, "Web application security by sql injection detectiontools," *IJCSI International Journal of Computer Science Issues,* vol. 9, pp. 332-339, 2012.