ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Website: www.ijircce.com

Vol. 5, Issue 5, May 2017

Counter Measure for Web Application Vulnerabilities using Augmented Algorithm

Neha, Sachin Shrivastava

M.Tech Student, Department of Computer Science Engineering, Satya College of Engineering and Technology, Palwal, Haryana, India

Assistant Professor, Department of Computer Science Engineering, Satya College of Engineering and Technology, Palwal, Haryana, India

ABSTRACT: Web applications have gained popularity over the years and have become an integral part of our daily lives interaction. We use these applications on a regular basis to interact with our friends and family, purchase items online and access bank accounts among others. However, these applications are not 100% secure, they are subject to a wide array of vulnerabilities such as such as SQL injection, Cross site tracing , cross site reference forgery and server side injections among others. To discover these weaknesses, web application scanners are used to report vulnerabilities found. The main objective of this study is to perform a comparative study of open source vulnerability testing tools, study their algorithm for these tools and propose an improved augmented algorithm. A simulation to test and validate the augmented algorithm was also developed. This research focuses on six of the open source web scanning tools which, were tested against four web based applications with known vulnerabilities to compare the tools capabilities and features. In addition, the algorithm of these tools was scrutinized with an aim of producing a augmented algorithm that will be more accurate in detecting web vulnerabilities**.**

KEYWORDS: Cross-Site Request Forgery, Cross Site Scripting, Cross-site tracing, Zed attack proxy, Structured query language injection, Server side injection.

I. INTRODUCTION

The number and importance of web applications have increased rapidly over the years (Jovanovic, Kruegel and Kirda, 2014). many organizations have embraced these technologies to explore new business opportunities and some companies have been forced to adopt the electronic commerce by their customers or competitors.

Web applications have gained popularity and have become part of our daily lives interaction. At the same time, new web application vulnerabilities emerge every now and then and endanger the use of the web-based applications. Therefore, manual code inspection or security audits must be done by highly trained experts who are labour-intensive, expensive, and prone to errors. (Kals et al, 2014).

Today employees are constantly responding to requests from both inside and outside the organization’s corporate network. While this has enormous benefits, it also present a challenge since it provides a weak access point that can be exploited by hackers to gain unauthorized access company information.

While the internet infrastructure is developed by very experienced experts with security flaws and solutions at the back of their mind, some of the web applications are developed by novice programmers who have little or no knowledge of about web application security. For this reason, they produce vulnerable web application that can be hacked exposing the organization’s confidential information.

Many organizations using web-based applications, experience one or more forms of security breaches. For instance, hackers may gain access to company data, unauthorized programs steal customer's login credentials and send them to cyber criminals, viruses may also be used to execute illegal transactions as well as other fraudulent activities. Hackers are also known to deface company's website and deny users access to services. Whereas some companies shy away from publicizing such information to avoid negative reputation, the news find their way to the public domain in one way or another. There is a need to identify the security lapse in various organizations and come up with ways of minimizing cybercrime.

With advancement in web technologies and shift from traditional desktop application to web-based solutions, the popularity of web-based applications has grown tremendously. Today, the web-based applications are used in security-critical environments, such as medical, financial and military systems (Stuttard & Pinto, 2011). Although the internet infrastructure is developed by experienced programmers with security concerns in their mind, some of the web applications are engineered by less experienced consulting programmers with little or no knowledge about security. This exposes the web application to various vulnerabilities and provides avenues for cyber criminals to gain unauthorized access to confidential information.

In one of the recent studies by the Ponemon Institute, they found out that that 45% of breaches exceed $500,000 in losses. In the largest of incidents, many Fortune-listed companies have given shareholder guidance that the losses would vary from a few dollars to millions of dollars. For this reason, it is prudent to do something in a proactive manner to avert or reduce harm before a cyber-attack.

Past studies have concentrated in benchmarking open source web vulnerability scanners to find out their capabilities and limitations. There is need to analyse different algorithms, identify their strength and weaknesses, with an aim of coming up with a Augmented algorithm that is superior, performs faster and can work on more inputs and in a complex situation. This scheme seeks to benchmarks different web vulnerability scanners, identify those flaws and suggest an improvising using augmented algorithm that can be adopted while developing such web applications.

Classification of Web Scanning Tools : Web scanning tools can be classified into three categories, namely white-box, black-box and grey-box.

White-box : White box web scanner checks the source code of the any web application to detect vulnerabilities. Through the analysis of the code, the white box tool can be able to identify vulnerabilities found in the application. The main advantage of using white box tools is the fact that they are able to identify more weaknesses, however, they are known to report vulnerabilities that do not exist. This is simply because the analysis of the code may overestimate the program paths that the program can execute. (Mirjalili, Nowroozi, & Alidoosti 2014)

Limitations of white-box tools :The main drawback of using the white box tools is that they are application specific, if a white box tool is meant for PHP, it will not work with other applications.

Black-box : Black box web scanners do not check the source code; instead they interact with the application just like a user using a web browser. They are comprised of three components

1. Crawling component which is responsible for browsing from one page to the other on the web application and parsing the provided links crawling to all the pages in an application.
2. Fuzzing is a component mutates or generates inputs either structurally or randomly and inserts it into the web application to discover vulnerabilities. The quality of any fuzzing component is determined by the number of inputs that are used to find vulnerabilities.
3. Analyser- checks the results of the attacker and determines which ones were successful or not. (Park , 2015)

Grey-box : Grey box is a Augmented approach that combines both the black box techniques and white box techniques. The main objective is to generate all the vulnerabilities that can be detected by the white box method and test them using the black box approach. If the test is successful, then, it will be reported by the tool. Grey-box takes

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Website: www.ijircce.com

Vol. 5, Issue 5, May 2017

advantage of black box approach; however, it also inherits the weaknesses of black box tools. For this reason, these tools are not popular.

Types of Web Application Vulnerabilities: As described by Stuttard and Pinto (2011), the number and the importance of Web applications have increased rapidly over the last few years. These vulnerabilities include - Cross-Site Scripting (XSS), Overflow Buffer Overflow, Server side injection (SSI), Command Injection, HTTP Response Splitting, Remote file inclusion, Local file inclusion, X-Path injection, Cross-site tracing and Cross-Site Request Forgery among others.

The number and impact of security weaknesses in such applications has grown as well. As illustrated on the figure below.

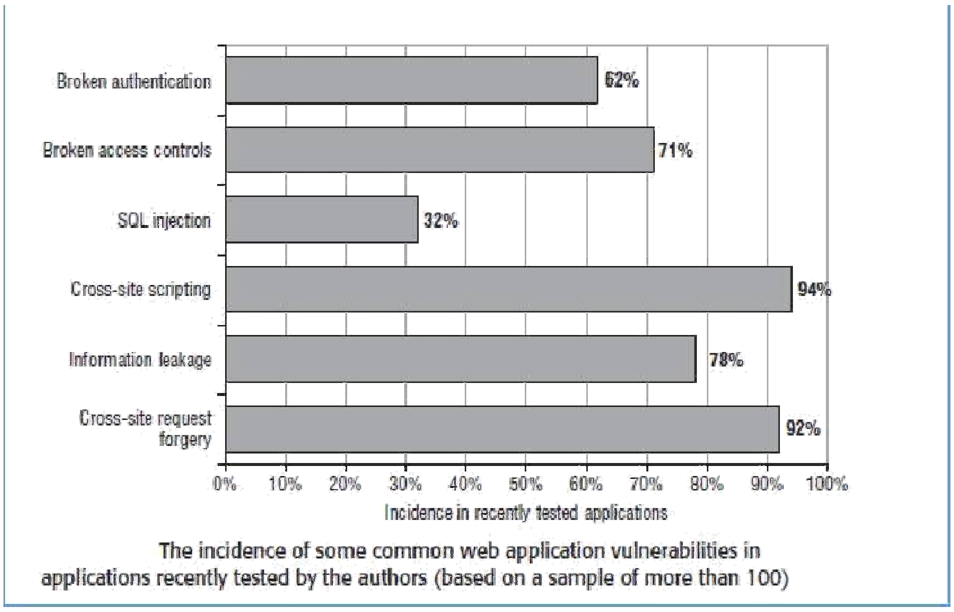

Figure 1: Common web vulnerabilities Source: The web applications hacker's handbook 2nd Edition. Stuttard and Pinto (2011)

Cross-Site Scripting (XSS) :One of the main goals of XSS attacks is to steal the credentials (using a cookie) of an authenticated user using malicious JavaScript code. Due to the sandbox model, JavaScript has access only to cookies that belong to the website from which the malicious JavaScript originated. All XSS attacks circumvent the sandbox model by injecting malicious JavaScript into the output of susceptible applications that have access to the wanted cookies. The sandbox model involves the isolation of computing environment used by software testers or developers to test new programming code. Kalman (2014) describes XSS as sanitization failure, whereby the attacker provides a web application JavaScript tags on input. When the link is returned to the user, un-sanitized, the user internet browser will execute it. This can be as simple as persuading a user to click on a link. Once the link has been clicked, the script will execute and perform undesirable actions
Prevention : XSS attacks can be prevented by designing web applications in a way that they don't return HTML tags to the client. Or by using regular expressions to strip away HTML tags.

SQL Injection: Dougherty, (2012) argues that SQL injection vulnerabilities are caused by software applications that accept data from an un-trusted source such as the internet. Howard, LeBlanc and Viega (2010) argues that an invalidated user input is used to construct an SQL algorithm which is then executed by the web server, for instance, the query used by a user's login. Data from un-rusted sources can lead to failure of validation and sanitation and sub which can be used to dynamically construct an SQL query to the database backing that application. However, argue that in the "SELECT *FROM users WHERE username='entered username' AND password='entered password'." If an attacker

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Website: www.ijircce.com

Vol. 5, Issue 5, May 2017

enters the string x' OR '1'='1 in both the username and the password. The query converts to "SELECT * FROM users WHERE username='x' OR '1'='1' AND password='x' OR '1'='1' " and because '1' is always equal to '1', this query is true for all records in the database. Khoury (2011)

X-Path Injection: As noted by Van der Loo & Poll (2011) X-Path injection is pretty much similar to SQL injection. The main difference between these two vulnerabilities is that SQL injection takes place in SQL database, whereas X-Path injection occurs in an XML, since X-Path is query language for XML data. Just like SQL injection the attack is based on sending malformed information to the web applications. This way the attacker can discover how the XML data is structured or access data that he has no authority to access.

Cross-Site Tracing: Cross-site tracing often abbreviated as XST as an attack that abuses the HTTP TRACE function. This function can be used to test web applications as the web server replies the same data that is sent to it via the TRACE command. An attacker can trick the web application in sending its normal headers via the TRACE command. This allows the attacker to be able to read information in the header such as a cookie. (Shelly, 2010)

There are three types of XSS namely;

1. Stored XSS – whereby the attackers code is stored on the web server
2. Reflected XSS **-** whereby the attackers code is added to a link in the web application
3. DOM based XSS- whereby the attacker's code is not injected in the web application but instead uses existing java scripts on the web page to write scripts.

Cross-Site Request Forgery: Cross-Site Request Forgery, is an attack where a malicious script tricks a user's browser into loading a request that performs an action on a web application that user is currently authenticated to. For example an attacker might post the following HTML on a website or send it in an HTML email <img src="http://www.bank.com/transfer_money?amount=10000&target_account=12345">. If the user is authenticated at his bank website (at http://www.bank.com) when this link is loaded it would transfer 10000 from the user's account to bank account number 12345. (Howard, LeBlanc and Viega, 2010) CSRF is the issuance of requests by 3rd party websites to a target site, say your bank using your internet browser and cookies while your session is still active. For instance, if you are logged on your bank's websites on one of the tabs, another tab in the browser can misuse the credentials on behalf of the attacker and do something the attacker instructs it to do. (Kalman, 2014)

Local File Inclusion: It is also known as path traversal or directory traversal. In this types of vulnerability, a file on the same server as the one where the web application is running is included on the page. For example a web application with the URL http://www.example.com/index.php?file=some_file.txt by manipulating the file parameter the attacker might be able to load a file that he should not be able to see.

As noted by Nagpal, Chauhan & Singh, (2015) local file inclusion vulnerability occurs when a web page is not properly sanitized and allows directory traversal characters such as dot or dash to be injected. This kind of attack can lead to:

1. Code execution on a web server
2. DOS denial of service attacks
3. Disclosure of confidential information
4. Code execution on the client side.

Remote File Inclusion: Van der Loo, (2011) argues that remote file inclusion is the same as local file inclusion, except for that the file that is included is a file from a different server than the one the web application runs on. An example of this vulnerability is the same as for local file inclusion. However, instead of changing the file name parameter to a local file, the attacker enters a path to a remote file. Testing for this vulnerability is also similar to local file inclusion. However, instead of a path to a local file a path to a remote file is used.

HTTP Response Splitting: As noted by Kalman (2014). HTTP response splitting is an attack whereby the attacker can control data that is used in a HTTP response header and appends a new line in this data. If the web application uses a redirect using GET parameter. The attacker can insert a new line to the value of GET parameter and add customized headers. This type of attack is experienced when, data is provided to a web application via untrusted sources such as HTTP or when the data is included in a HTTP response header without proper checking malicious characters. For this

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Website: www.ijircce.com

Vol. 5, Issue 5, May 2017

attack to be successful, the application must allow carriage return or line feed in the header. The underlying platform must also be prone to injection of such characters (%Od or %Oa)

Server Side Injection (SSI): SSI attack involves, the attacker enters SSI directives on the web server. These commands are executed directly on the web server and making undesirable changes to the web application. SSI attack allows web applications by injecting scripts in HTML web pages or executing arbitrary codes remotely. An attack will be successful if the web application allows the execution of SSI code execution without proper validation. For instance, one of the known vulnerabilities in SSI exist in IIS version 4 and 5, which allows cyber criminals to obtain system privileges via buffer overflow failure in a dll file (ssinc.dll). By creating a malicious webpage, the criminals perform undesirable actions or perform fraud. (Mirjalili, Nowroozi, & Alidoosti 2014)

II. RELATED WORK

The use of computers, tablets and smart phones has greatly increased over the past few years, as noted by Stuttard & Pinto (2011) web applications have been developed to perform practically every useful function you could implement online. These include-Online Shopping, Social Networking, Gambling & Online casino, Banking, Web search, Auctions, Webmail, and Interactive web pages among others. In a report published by Whitehat "86% of all websites tested by Whitehat Sentinel had at least one serious vulnerability, and most of the time, far more than one – 56% to be precise. (Whitehat, 2015)

According to Shema (2011), many organizations rely upon customized web applications to implement business processes. These may include full-blown applications, or consist of modules such as online, login pages shopping carts, and other kinds of dynamic content. Some of these software applications in your network could be developed in-house. In addition, some may be legacy websites with no designated ownership or support. Manually analyzing all of these for loopholes and prioritizing their importance for remediation can be a daunting task without organizing efforts and using automated tools to improve accuracy and efficiency.

Employees are continuously responding to requests from both inside and outside the organization's corporate network using gadgets such as tablets, smartphones or laptops. While this has enormous benefits, the negative drawback is the fact that hackers may take advantage of connectivity to gain unauthorized access to vital company information. For this reason, it is imperative for any company to ensure that they protect their web applications and reduce the possibility of a security breach to their electronic system. Testing the weakness of web applications with automated penetration testing tools produces relatively quick results. Currently, there are many such tools, both commercial and open source.

Yu et al (2011) noted that web application security vulnerabilities are on the increase. These vulnerabilities allow attackers to perform undesirable actions that range from gaining unauthorized account access, to obtaining confidential data such as credit card numbers and in some extreme cases, they threaten to reveal the identities of intelligence personnel.

In a one of the recent article published in the "The New York Times", Kevin Granville (2015) reported that, In November 2014, a huge attack that wiped clean several internal data centres. It out rightly led to the cancellation of the theatrical release of "The Interview," a film about the fictional assassination of Kim Jong-un, the North Korean leader. Contracts, film budgets, salary lists, entire films and Social Security numbers were stolen, including - to the dismay of top executives leaked emails that included criticisms of one of the top celebrities Angelina Jolie and undesirable remarks about the Former USA President. Former President Obama administration said that, North Korea was behind the attack.

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Website: www.ijircce.com

Vol. 5, Issue 5, May 2017

III. PROPOSED SYSTEM

The algorithm was be tested by translating it in to a simulation developed using Microsoft .NET and Python development platform. The simulation will be run against the few web applications and the results collected about detection accuracy, the time taken to scan a given application as well as the reliability and consistency. After the testing process, the results of the simulation were compared with the other opens source web scanners also. The Augmented algorithm was designed with an aim of improving weaknesses that were found with existing algorithms. A black box approach will be adopted with an aim of improving application scanners. The tool used to test and validate the proposed Augmented algorithm will demonstrate the improved capability of the tool. During the development of this algorithm, divide and conquer approach was adopted. This means that the code is engineered to crawl all the Webpages in a web application and scan for the various vulnerabilities independently.

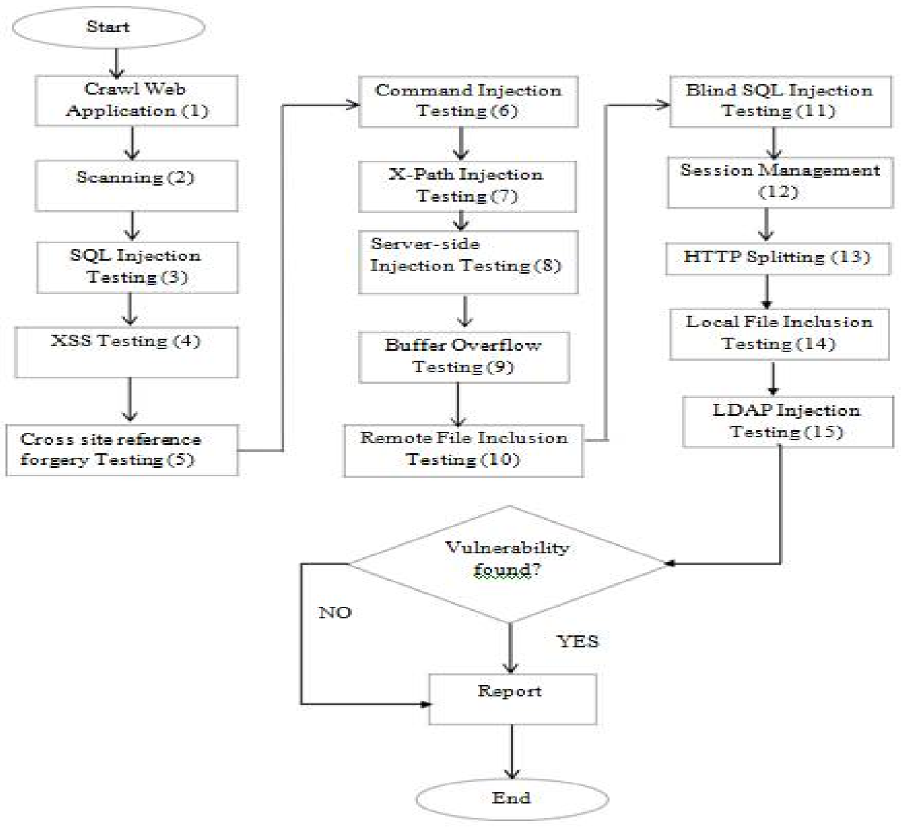
Start
Crawl Web Application (1)
Scanning (2)
SQL Injection Testing (3)
XSS Testing (4)
Cross site reference forgery Testing (5)
Command Injection Testing (6)
X-Path Injection Testing (7)
Server-side Injection Testing (8)
Buffer Overflow Testing (9)
Remote File Inclusion Testing (10)
Blind SQL Injection Testing (11)
Session Management (12)
HTTP Splitting (13)
Local File Inclusion Testing (14)
LDAP Injection Testing (15)
Vulnerability found?
NO
YES
Report
End

Figure 2: Architecture and Flow Diagram for proposed Augmented Algorithm Scheme.

IV. PROPOSED ALGORITHM

Crawling

1) Identify the root of the website (the home page url)
2) Mark the page as visited and push it into a queue
3) Traverse down to identify the immediate sub folders / sub urls
4) Mark the urls as visited and add them to the queue
5) For each url in the url queue
    a. Traverse down to identify sub urls
    b. Mark them as visited and push them in to queue

c. Repeat step 5 until a dead end is reached
d. Once dead end is reached remove the url in the immediate top level from the queue

6) Urls in the visited urls array/list it the complete set of urls for the web application

Ask user to specify the starting URL on web and file type that web App should crawl. Add the URL to the visited list of URLs and the url queue to search.

```
While not empty ( the list of URLs in url queue search )
   {
        Take the first URL in from the list of URLs
        Mark this URL as already searched URL.
        If the URL protocol is not HTTP then
              break;
              go back to while
        If robots.txt file exist on site then
              If file includes .Disallow. statement then break;
                    go back to while
        Open the URL
        If the opened URL is not HTML file then
        Break;
          Go back to while Iterate the HTML file
        While the html text contains another link {
          If robots.txt file exist on URL/site then
                If file includes .Disallow. statement then break;
                      go back to while
          If the opened URL is HTML file then
                If the URL isn't marked as searched then
                Mark this URL as already searched URL. Else if type of file is user requested
                Add to list of files found.
        }
   }
```

V. Pseudo Code

```
1)   Initialize sql characters in an array
2)   Create two maps or lists to store the sql error messages
i.   One for storing specific database error messages like oracle, mysql, microsoft sql error messages etc
ii.  Other for storing generic database error messages
73)  Initialize error values in to the maps/list mentioned above
4)   Initialize the scanner method – the scanner accepts the http message as input from the the crawler - http
message has details on each request or url with the parameter list
5)   For each parameter in the http message
i.   Input sql characters from the sql characters array
ii.  Verify the response to check for any matches on error messages from the two maps or lists
iii. If a match occurs -Flag as sql vulnerability
iv.  Else - Repeat step 5 until the end of parameter list is reached
     End
6.   Cross Site Scripting
7)   For each url in the list of visited urls
     Identify all parameters
     Push parameters in to parameter list
     For each parameter in the parameter queue
```

Supply a script or a XSS test case as input to the parameter and pass the request
Verify the response to identify the supplied script or test case reflected back
Report the vulnerability if the response has a script

VI. SIMULATION RESULTS

A Augmented algorithm was designed based on the logic expressed in the diagram below. The major milestone of this algorithm is to reduce the time taken during the scanning process as well as increase the detection accuracy. The Augmented algorithm is derived from existing algorithms with a goal of increasing the vulnerability scanning accuracy and the time it takes to scan any given web application. Although the accuracy may not be achieved 100% emphasis, an effort has been put to raise it above the existing tools. The results of the tests will be benchmarked with OWASP results which are updated on a regular basis.

The Augmented algorithm is based on the idea of carefully, combining desirable features of various components so that the new algorithm has the ability to discover vulnerabilities that could not be detected. However the combination is not done blindly, it is based on various factors such as optimization and sophistication among others with an aim of increasing efficiency. The Augmented algorithm comprises of five phases as shown in the figure above. Inspection or crawling this phase focuses on looking for information about the web application. The more the details found on this stage, the more successful the entire scanning process will be. Once the first phase is completed, the Scanning process begins, which involves, recognizing the weaknesses that exist in the web application. Once the vulnerabilities are discovered they are analyzed in the next phase and then a report is displayed at the end of the entire process.

Trade-offs (Time vs. Accuracy) if the algorithm is designed to scan all the possible web vulnerabilities, such a program would take a very long time to scan. This would be unrealistic, since users want a program that takes less time to detect any vulnerability. For this reason, accuracy has not been fully optimized. However the fuzzing and crawling components have been engineered to work efficiently and deliver acceptable results. Accuracy was given a lower priority while scanning time was awarded a priority since it would not be worthwhile to create an algorithm that is 100% accurate but takes a long period of time to scan. In a real world scenario, it would not be practical for users to wait for time consuming web scanners. As a matter of fact an application that takes long to scan would be ignored by the users.

Simulation Design A program to test and validate the Augmented algorithm is built, based on the flowchart below. This simulation will be useful in testing and validating the Augmented algorithm.

The user will input the URL (uniform resource locator) of the web application to be scanned and click on the scan button. The scanning process involves crawling and parsing and the discovery of the vulnerabilities, this process is repeated until all the vulnerabilities have been discovered. Once this process is completed, the analysis is done and finally a report is displayed showing the discovered vulnerabilities discovered and their location.

The scanning process includes, crawling and fuzzing. After the scanning process is completed, the results are submitted for analysis and a report is displayed.

Input: The URL of the web application to be tested. This is provided by the user who initiates the web scanning process.

Processing: This involves crawling all the web pages, fuzzing, and identification of any weakness and firing inputs to check for any vulnerability.

Output: The results of processing are analysed and presented in a report format.

Contents of the Scanning Report

Although the report displayed depends on the tool used some of the common features include

- Number of vulnerabilities discovered
- Name or type of the vulnerabilities detected
- Quantity or number of vulnerabilities discovered

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Website: www.ijircce.com

Vol. 5, Issue 5, May 2017

- Location or the webpage where the vulnerability has been detected

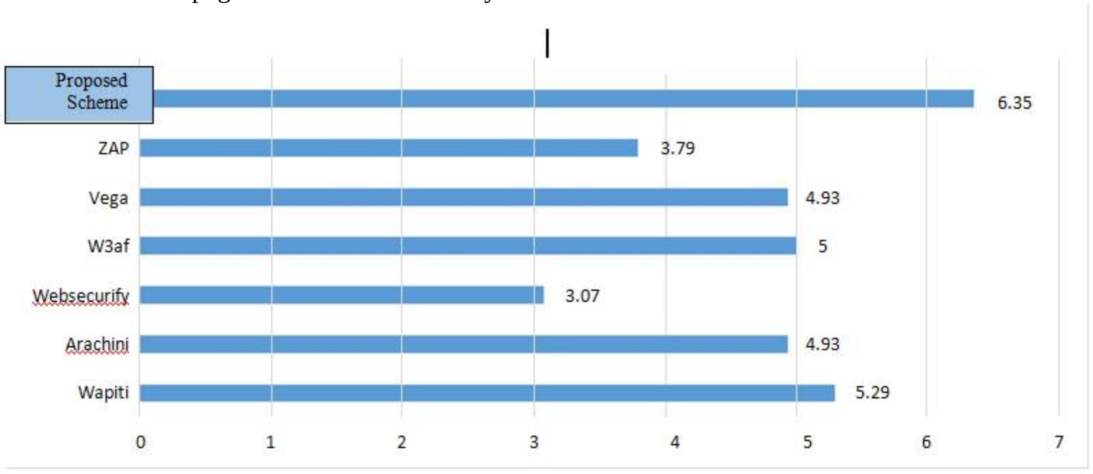

Chart 1: Weighted Average for the Proposed Scheme using Augmented Algorithm and Comparison with others

VII. Conclusion and Future Work

Conclusion about the Augmented algorithm : The proposed Augmented algorithm is extensive in the execution of its detection mechanism against web application vulnerabilities. The proposed Augmented algorithm reports more vulnerabilities and presents a proficient manner while reporting discovered vulnerabilities. However since the proposed Augmented algorithm did not scan 100% of the existing vulnerabilities. There is need to increase the algorithm crawling component in order to ensure that it executed “deep” crawling. In addition the results presented shows that the proposed algorithm needs to be optimised to do the scanning in a short period of time. More research is needed to come up with a sophisticated algorithm that has the capacity to detect more vulnerability.

Suggestions for Further Research

i. Suggest a solution to the vulnerabilities discovered

After analyzing the reports of all the tools used in this study, none of the tools sampled have suggested has a remedy for the vulnerabilities reported. In my own opinion, it would be prudent to suggest the way the source code should be structured to fix the vulnerabilities detected.

ii. Improved fuzzing component

Since the Augmented algorithm fails to detect all the vulnerabilities, there is a need to use a more advanced logic in the fuzzing component of the algorithm to get getter results. The fuzzing component is responsible for firing the necessary inputs to determine whether vulnerabilities exist or not. The fuzzing logic used by Ck AppScan should be improved further to increase the detection accuracy.

iii. Reduced scanning time

The scheme developed by the researcher takes a long period of time to scan a web based application. For this reason, it is important to improve the overall scanning mechanisms of the Augmented algorithm and reduce the scanning time without compromising on the detection accuracy.

References

1. Alssir, F. T., & Ahmed, M. (2012). Web Security Testing Approaches: Comparison Framework. In Proceedings of the 2011 2nd International Congress on Computer Applications and Computational Science (pp. 163-169). Springer Berlin Heidelberg.
2. Antunes & Vieira (2012). Defending against web application vulnerabilities. Computer, (2), 66-72.
3. Bau, J., Bursztein, E., Gupta, D., & Mitchell, J. (2010). State of the art: Automated black-box web application vulnerability testing. In Security and Privacy (SP), 2010 IEEE Symposium on (pp. 332-345). IEEE.

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Website: www.ijircce.com

Vol. 5, Issue 5, May 2017

4. Chen, S. (2014). wavsep. Available: http://sectooladdict.blogspot.com/2014/02/wavsep-web-application-scanner.html. [Accessed 09 July 2015.]
5. Dessiatnikoff, A., Akrout, R., Alata, E., Kaaniche, M., & Nicomette, V. (2011). A clustering approach for web vulnerabilities detection. InDependable Computing (PRDC), 2011 IEEE 17th Pacific Rim International Symposium on (pp. 194-203). IEEE.
6. Dougherty, C. (2012).Practical Identification of SQL Injection Vulnerabilities. 2012. US-CERT-United States Computer Emergency Readiness Team. Citado na, 34. . [Accessed: 08th June 2015]
7. Doupe, A., Cova, M., & Vigna, G. (2010). Why Johnny can't pentest: An analysis of black-box web vulnerability scanners. In Detection of Intrusions and Malware, and Vulnerability Assessment (pp. 111-131). Springer Berlin Heidelberg. [Accessed: 10th June 2015]
8. Fonseca, J., Vieira, M., & Madeira, H. (2014). Evaluation of Web Security Mechanisms using Vulnerability & Attack Injection. Dependable and Secure Computing, IEEE Transactions on, 11(5), 440-453.
9. Granville, K . (2015).Nine Recent Cyber-attacks against Big Businesses. New York Times [online] Available from http://www.nytimes.com/interactive/2015/02/05/technology/recent-cyberattacks.html?_r=1. [Accessed 08 July 2015.]
10. Howard, M., LeBlanc, D., & Viega, J. (2010). 24 deadly sins of software security [electronic book]: Programming flaws and how to fix them. New York: McGraw-Hill.
11. Jovanovic, N., Kruegel, C., & Pixy, E. K. (2010). A Static Analysis Tool for Detecting Web Application Vulnerabilities (Short Paper). In Proceedings of the 2006 IEEE symposium on Security and Privacy, Washington, DC, IEEE Computer Society (pp. 258-263).
12. Kalman., G. (2014). Ten Most Common Web Security Vulnerabilities.[online] Available from: http://www.toptal.com/security/10-most-common-web-security-vulnerabilities [Accessed 08 July 2015.]
13. Kals, S., Kirda, E., Kruegel, C., & Jovanovic, N. (2014). A web vulnerability scanner. In Proceedings of the 15th international conference on World Wide Web (pp. 247-256). ACM.
14. Khoury, N., Zavarsky, P., Lindskog, D., & Ruhl, R. (2011). Testing and assessing web vulnerability scanners for persistent SQL injection attacks. In Proceedings of the First International Workshop on Security and Privacy Preserving in e-Societies (pp. 12-18). ACM.
15. McQuade, K. (2014). Open Source Web Vulnerability Scanners: The Cost Effective Choice?. In Proceedings of the Conference for Information Systems Applied Research ISSN (Vol. 2167, p. 1508). [Accessed: 18th June 2015]