IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.102 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 13, Issue 3, March 2024

DOI: 10.17148/IJARCCE.2024.13381

Detecting and Removing Web Application Vulnerabilities with SQL Injection Prevention

Dr.J.Jeyaboopathiraja[1], N.Mithun[2]

[1]Assistant Professor, PG & Research Department of Computer Science, Sri Ramakrishna College of Arts & Science

[2]PG Student, PG & Research Department of Computer Science, Sri Ramakrishna College of Arts & Science

Abstract: Cross site scripting vulnerability is one of the most widely spreaded and existed vulnerability in today's web application. So this project focuses on implementing cross site scripting vulnerability scanner to find cross site scripting vulnerability in a web application .This paper is useful for security researchers to find cross site scripting vulnerability in less time and to get accurate results. Although a large research effort on web application security has been going on for more than a decade, the security of web applications continues to be a challenging problem. An important part of that problem derives from vulnerable source code, often written in unsafe languages like PHP. Source code static analysis tools are a solution to find vulnerabilities, but they tend to generate false positives, and require considerable effort for programmers to manually fix the code. We explore the use of a combination of methods to discover vulnerabilities in source code with fewer false positives. We combine taint analysis, which finds candidate vulnerabilities, with data mining, to predict the existence of false positives. This approach brings together two approaches that are apparently orthogonal: humans coding the knowledge about vulnerabilities (for taint analysis), joined with the seemingly orthogonal approach of automatically obtaining that knowledge (with machine learning, for data mining). Given this enhanced form of detection, we propose doing automatic code correction by inserting fixes in the source code. Our approach was implemented in the WAP tool, and an experimental evaluation was performed with a large set of PHP applications. Our tool found 388 vulnerabilities in 1.4 million lines of code. Its accuracy and precision were approximately 5% better than PhpMinerII's and 45% better than Pixy's.

Keywords: DenSet, cancer, BCCD, Filtering, blood cells

I. INTRODUCTION

Since its appearance in the early 1990s, the World Wide Web evolved from a platform to access text and other media to a framework for running complex web applications. These applications appear in many forms, from small home-made to large-scale commercial services (e.g., Google Docs, Twitter, and Facebook). However, web applications have been plagued with security problems. For example, a recent report indicates an increase of web attacks of around 33% in 2012. Arguably, a reason for the insecurity of web applications is that many programmers lack appropriate knowledge about secure coding, so they leave applications with flaws. However, the mechanisms for web application security fall in two extremes. On one hand, there are techniques that put the programmer aside, e.g., web application firewalls and other runtime protections. On the other hand, there are techniques that discover vulnerabilities but put the burden of removing them on the programmer, e.g., black-box testing, and static analysis.

This paper explores an approach for automatically protecting web applications while keeping the programmer in the loop. The approach consists in analyzing the web application source code searching for input validation vulnerabilities, and inserting fixes in the same code to correct these flaws. The programme is kept in the loop by being allowed to understand where the vulnerabilities were found, and how they were corrected. This approach contributes directly to the security of web applications by removing vulnerabilities, and indirectly by letting the programmers learn from their mistakes. This last aspect is enabled by inserting fixes that follow common security coding practices, so programmers can learn these practices by seeing the vulnerabilities, and how they were removed.

To predict the existence of false positives, we introduce the novel idea of assessing if the vulnerabilities detected are false positives using data mining. To do this assessment, we measure attributes of the code that we observed to be

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.102 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 13, Issue 3, March 2024

DOI: 10.17148/IJARCCE.2024.13381

associated with the presence of false positives, and use a combination of the three top-ranking classifiers to flag every vulnerability as false positive or not. We explore the use of several classifiers: ID3, C4.5/J48, Random Forest, Random Tree, K-NN, Naive Bayes, Bayes Net, MLP, SVM, and Logistic Regression. Moreover, for every vulnerability classified as false positive, we use an induction rule classifier to show which attributes are associated with it. We explore the JRip, PART, Prism, and Ridor induction rule classifiers for this goal. Classifiers are automatically configured using machine learning based on labeled vulnerability data.

Ensuring that the code correction is done correctly requires assessing that the vulnerabilities are removed, and that the correct behavior of the application is not modified by the fixes. We propose using program mutation and regression testing to confirm, respectively, that the fixes function as they are programmed to (blocking malicious inputs), and that the application remains working as expected (with benign inputs). Notice that we do not claim that our approach is able to correct any arbitrary vulnerability, or to detect it; it can only address the input validation vulnerabilities it is programmed to deal with. This paper also describes the design of the Web Application Protection (WAP) tool that implements our approach. WAP analyzes and removes input validation vulnerabilities from programs or scripts written in PHP 5, which according to a recent report is used by more than 77% of existing web applications. WAP covers a considerable number of classes of vulnerabilities:
SQL injection (SQLI), cross-site scripting (XSS), remote file inclusion, local file inclusion, directory traversal and path traversal, source code disclosure, PHP code injection, and OS command injection. The first two continue to be among the highest positions of the OWASP Top 10 in 2013, whereas the rest are also known to be high risk, especially in PHP. Currently, WAP assumes that the background database is MySQL, DB2, or PostgreSQL. The tool might be extended with more flaws and databases, but this set is enough to demonstrate the concept. Designing and implementing WAP was a challenging task. The tool does taint analysis of PHP programs, a form of data flow analysis. To do a first reduction of the number of false positives, the tool performs global, inter procedural, and context-sensitive analysis, which means that data flows are followed even when they enter new functions and other modules (other files). This result involves the management of several data structures, but also deals with global variables (that in PHP can appear anywhere in the code, simply by preceding the name with global or through the $_GLOBALS array), and resolving module names (which can even contain paths taken from environment variables). Handling object orientation with the associated inheritance and polymorphism was also a considerable challenge. The main contributions of the paper are: 1) an approach for improving the security of web applications by combining detection and automatic correction of vulnerabilities in web applications; 2) a combination of taint analysis and data mining techniques to identify vulnerabilities with low false positives; 3) a tool that implements that approach for web applications written in PHP with several database management systems; and 4) a study of the configuration of the data mining component, and an experimental evaluation of the tool with a considerable number of open source PHP applications. The table stores the extracted target links from the target website

http://192.168.43.70/dvwa/instructions.php
http://192.168.43.70/dvwa/vulnerabilities/brute/
http://192.168.43.70/dvwa/vulnerabilities/fi/?page=include.php
http://192.168.43.70/dvwa/vulnerabilities/sqli_blind/
http://192.168.43.70/dvwa/vulnerabilities/upload/
http://192.168.43.70/dvwa/security.php

The table that stores the testing target links from the tested extracted links

[+] TESTING FORM IN http://192.168.43.70/dvwa/instructions.php
[+] TESTING FORM IN http://192.168.43.70/dvwa/vulnerabilities/brute/
[+] TESTING IN [PARAMETER] http://192.168.43.70/dvwa/vulnerabilities/fi/?page=include.php

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.102 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 13, Issue 3, March 2024

DOI: 10.17148/IJARCCE.2024.13381

[+] TESTING FORM IN http://192.168.43.70/dvwa/vulnerabilities/sqli_blind/
[+] TESTING FORM IN http://192.168.43.70/dvwa/vulnerabilities/upload/
[+] TESTING FORM IN http://192.168.43.70/dvwa/security.php

The table that stores the vulnerability links in the tested target links

[+] XSS VULNERABILITY IN [FORM] http://192.168.43.70/dvwa/vulnerabilities/xss_r/
[+] XSS VULNERABILITY IN [FORM] http://192.168.43.70/dvwa/vulnerabilities/xss_s/

II. RELATED WORKS

With the increase in size and complexity of software more focus is needed on Quality Assurance. This Research work has presented a systematic review which discuss on approaches of combining Static QA and Dynamic QA techniques. As we know that static analysis (Analyzing program without execution) and dynamic analysis (Analyzing the system by running it) playing important role in improvement in software QA. The objective of combining these two QA approaches was according to Quality Assurance process quality and product quality. The Purpose of combining the approaches was integration and separation.

Taint analysis tools like CQUAL and Splint (both for C code) use two qualifiers to annotate source code: the *untainted* qualifier indicates either that a function or parameter returns trustworthy data (e.g., a sanitization function), or a parameter of a function requires trustworthy data (e.g., *mysql_query*). The *tainted* qualifier means that a function or a parameter returns non-trustworthy data (e.g., functions that read user input).

Finding cross site scripting vulnerability in manual way is very difficult and takes lots of hours to find out so automated vulnerability scanners are used. There are some vulnerability scanners are available but those scanners are giving only false positive results and some scanners are attached with some malicious malwares and backdoors so it is very unsafe to use those scanners. To that end, this work features a model that analyzes source code and uses data from the Solution Knowledge Database (SKB) to create feature sets for recognizing a set of code patterns. Specifically, source code is represented using an Abstract Syntax Tree [5], which provides the ability to extract statements. This work is based on KM Trajectory Service Frame work [7] which is framed to reduce the dependency on human resources in software development organizations. Knowledge Champions and KM team members have to contribute their time in creation and maintenance of solution knowledge base. This provides functional KM solutions for software process improvement. This work proposes the extraction of code sets from project repositories to present to the user a set of methods that supplies code segment for each requirement.

III. METHODOLOGY

The drawbacks which are faced during existing system can be eradicated by using the proposed system. The main objective of the proposed System is to get a accurate result and malware free. The proposed system is to create our own tool to find Cross-site scripting vulnerabilities and to get accurate results. so that finding cross-site scripting vulnerability time is reduced and we can prevent some malicious activities by using other scanners. This paper explores an approach for automatically protecting web applications while keeping the programmer in the loop. The approach consists in analyzing the web application source code searching for input validation vulnerabilities, and inserting fixes in the same code to correct these flaws. The programmer is kept in the loop by being allowed to understand where the vulnerabilities were found, and how they were corrected. This approach contributes directly to the security of web applications by removing vulnerabilities, and indirectly by letting the programmers learn from their mistakes. This last aspect is enabled by inserting fixes that follow common security coding practices, so programmers can learn these practices by seeing the vulnerabilities, and how they were removed. We explore the use of a novel combination of methods to detect this type of vulnerability: static analysis with data mining. Static analysis is an effective mechanism

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.102 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 13, Issue 3, March 2024

DOI: 10.17148/IJARCCE.2024.13381

to find vulnerabilities in source code, but tends to report many false positives (non-vulnerabilities) due to its undesirability. To predict the existence of false positives, we introduce the novel idea of assessing if the vulnerabilities detected are false positives using data mining. To do this assessment, we measure attributes of the code that we observed to be associated with the presence of false positives, and use a combination of the three top-ranking classifiers to flag every vulnerability as false positive or not. The main contributions of the paper are: 1) an approach for improving the security of web applications by combining detection and automatic correction of vulnerabilities in web applications; 2) a combination of taint analysis and data mining techniques to identify vulnerabilities with low false positives; 3) a tool that implements that approach for web applications written in PHP with several database management systems; and 4) a study of the configuration of the data mining component, and an experimental evaluation of the tool with a considerable number of open source PHP applications.

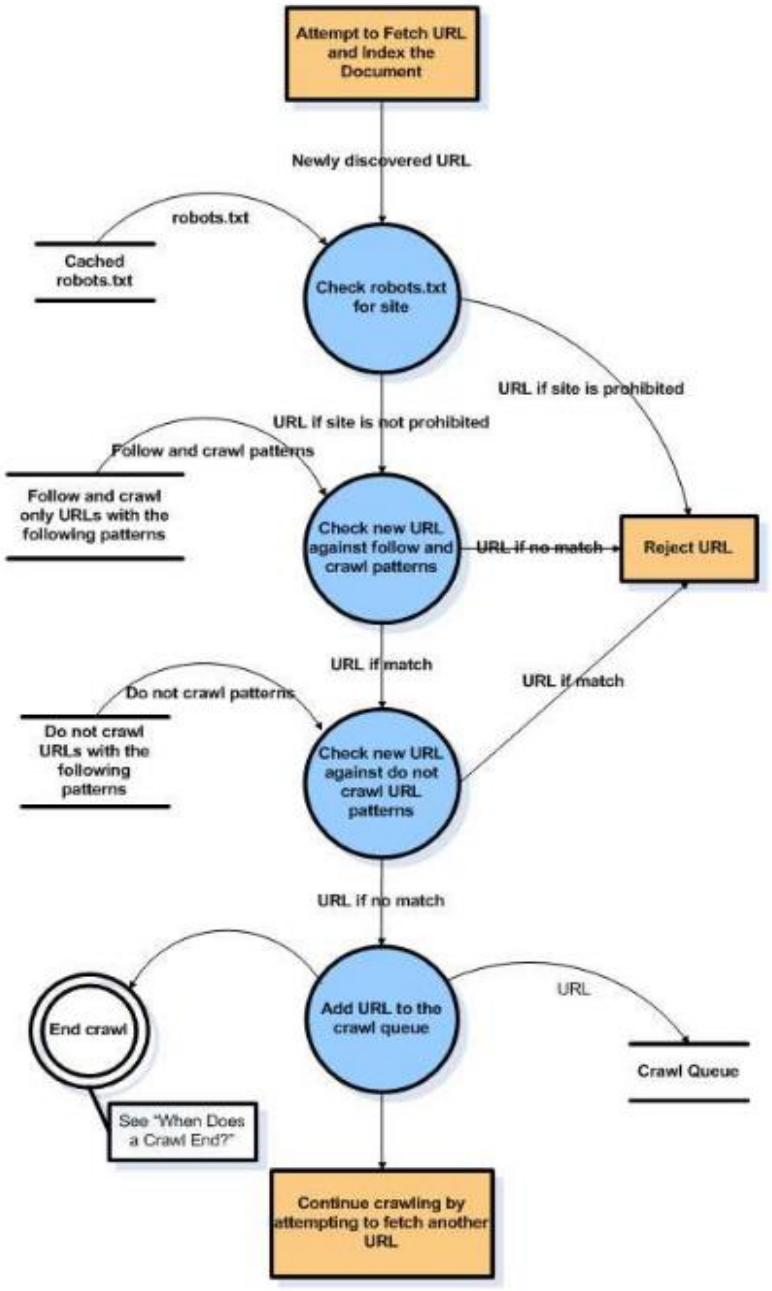

Figure-1 System Architecture

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.102 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 13, Issue 3, March 2024

DOI: 10.17148/IJARCCE.2024.13381

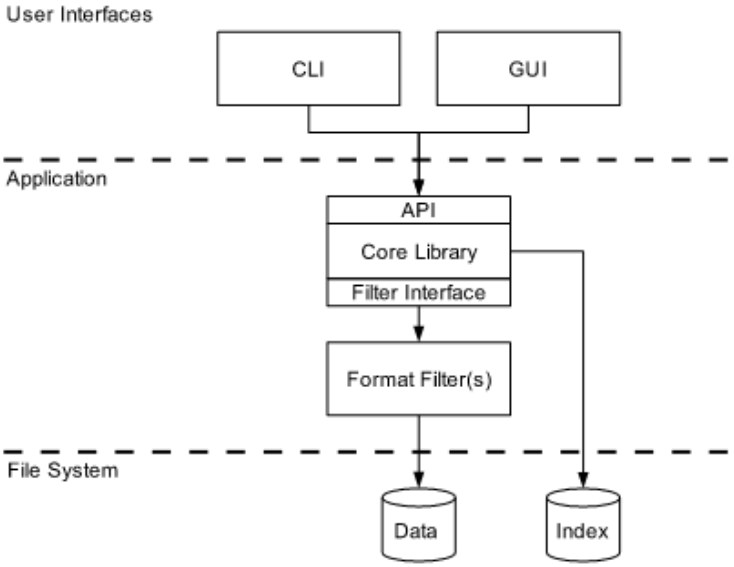

Figure-2 Design Framework

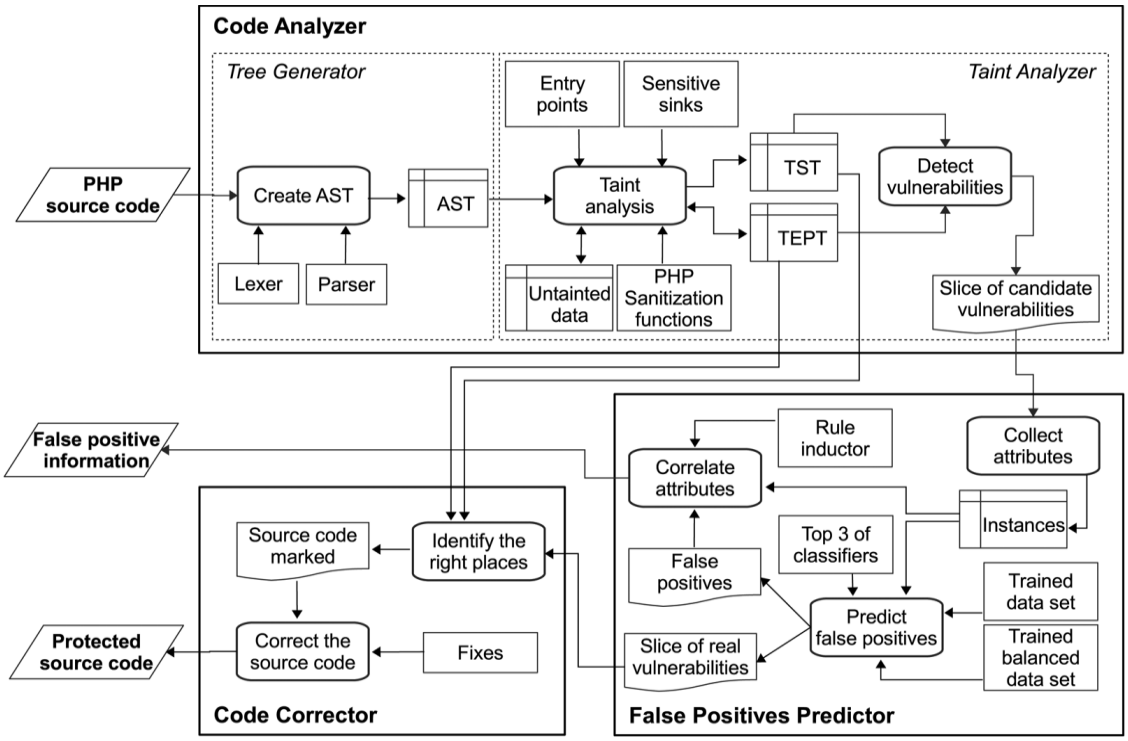

Figure-3 Overall Architecture

IV. RESULTS AND DISCUSSION

This paper presents an approach for finding and correcting vulnerabilities in web applications, and a tool that implements the approach for PHP programs and input validation vulnerabilities. The approach and the tool search for vulnerabilities using a combination of two techniques: static source code analysis, and data mining. Data mining is used to identify false positives using the top 3 machine learning classifiers, and to justify their presence using an induction rule classifier. All classifiers were selected after a thorough comparison of several alternatives. It is important to note that this combination of detection techniques cannot provide entirely correct results. The static analysis problem is

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.102 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 13, Issue 3, March 2024

DOI: 10.17148/IJARCCE.2024.13381

undecidable, and resorting to data mining cannot circumvent this undecidability, but only provide probabilistic results. The tool corrects the code by inserting fixes, i.e., sanitization and validation functions. Testing is used to verify if the fixes actually remove the vulnerabilities and do not compromise the (correct) behavior of the applications. The tool was experimented with using synthetic code with vulnerabilities inserted on purpose, and with a considerable number of open source PHP applications. It was also compared with two source code analysis tools: Pixy, and PhpMinerII. This evaluation suggests that the tool can detect and correct the vulnerabilities of the classes it is programmed to handle. It was able to find 388 vulnerabilities in 1.4 million lines of code. Its accuracy and precision were approximately 5% better than PhpMinerII's, and 45% better than Pixy's.

Figure -4 Classification of Sample

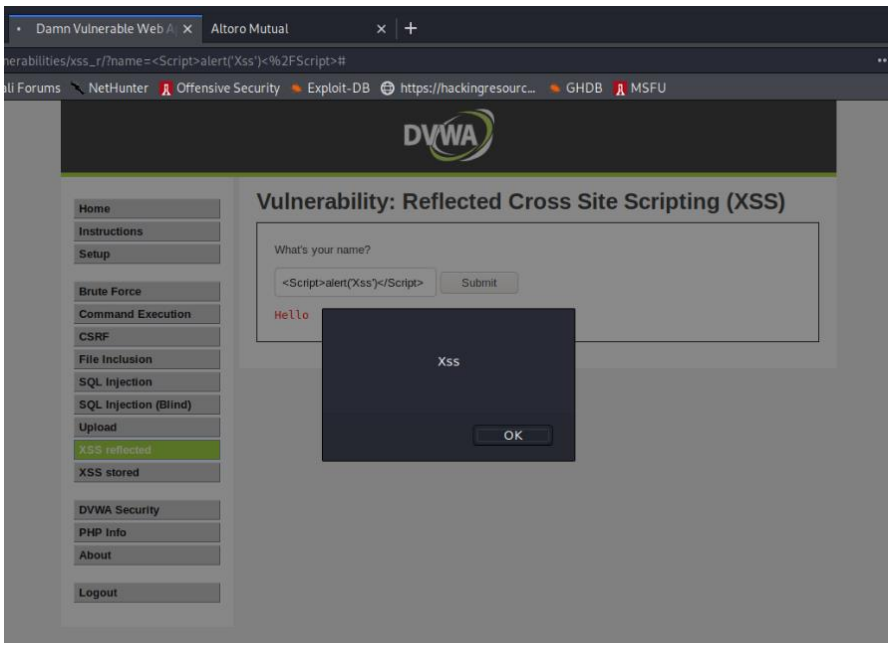

Figure -5 XSS in manual Testing

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.102 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 13, Issue 3, March 2024

DOI: 10.17148/IJARCCE.2024.13381

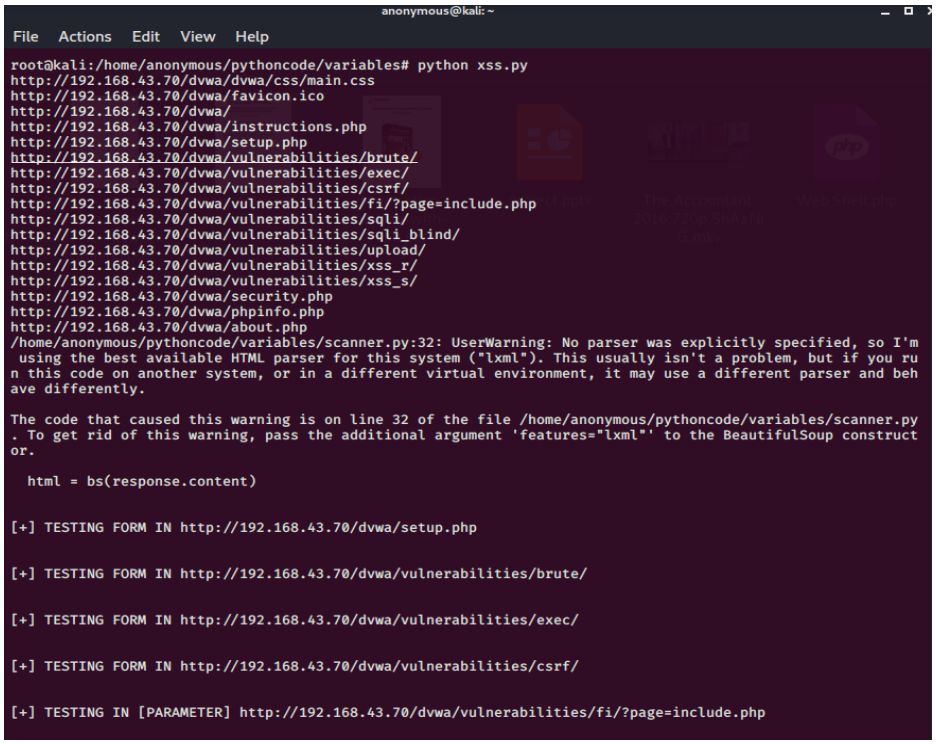

Figure -6 Extracted Links

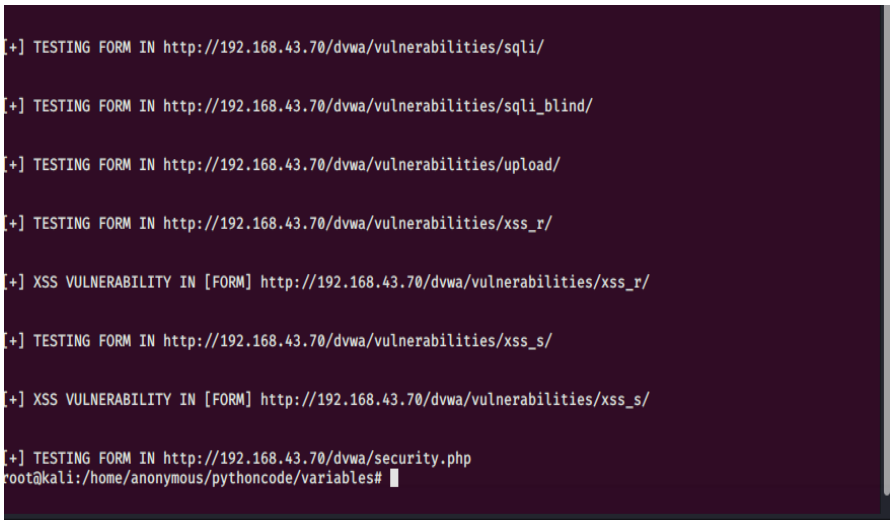

Figure -7 Vulnerabilities Scanner in XSS

V. CONCLUSION

In conclusion, our study the cross-site scripting vulnerability scanners help all category of website to scan for vulnerability without the help of the internet. The application was designed in a user friendly manner; therefore any user can make use of the application in an effective manner. The cross-site scripting vulnerability scanners help to find vulnerability in less time and get accurate results. The application was designed to scan the website without using internet, therefore the user won't suffer to scan the website due to not getting signal. Thus the application was tested and implemented in an effective manner to reach all categories of websites to scan without internet.

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.102 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 13, Issue 3, March 2024

DOI: 10.17148/IJARCCE.2024.13381

REFERENCES

1. B. McMahan et al., "Communication-Efficient Learning of Deep Networks From Decentralized Data", *Artificial Intelligence and Statistics Proc. PMLR*,vol. 10, no. 1, pp. 1273-82, 2017.
2. Napoleon D. and Praneesh M. “Detection of Brain Tumor using Kernel Induced Possiblistic C-Means Clustering”, volume no.3, issue no.9, pp 436-438, 2013
3. Symantec, Internet threat report. 2012 trends, vol. 18, Apr. 2013.
4. W. Halfond, A. Orso, and P. Manolios, “WASP: protecting web applications using positive tainting and syntax aware evaluation,” *IEEE Trans. Softw. Eng.*, vol. 34, no. 1, pp. 65–81, 2008.
5. T. Pietraszek and C. V. Berghe, “Defending against injection attacks through context-sensitive string evaluation,” in *Proc. 8th Int. Conf. Recent Advances in Intrusion Detection*, 2005, pp. 124–145.
6. X. Wang, C. Pan, P. Liu, and S. Zhu, “SigFree: A signature-free buffer overflow attack blocker,” in *Proc. 15th USENIX Security Symp.*, Aug. 2006, pp. 225–240.
7. J. Antunes, N. F. Neves, M. Correia, P. Verissimo, and R. Neves, “Vulnerability removal with attack injection,” *IEEE Trans. Softw. Eng.*, vol. 36, no. 3, pp. 357–370, 2010.
8. R. Banabic and G. Candea, “Fast black-box testing of system recovery code,” in *Proc. 7th ACM Eur. Conf. Computer Systems*, 2012, pp. 281–294.
9. Y.-W. Huang *et al.*, “Web application security assessment by fault injection and behavior monitoring,” in *Proc. 12th Int. Conf. World Wide Web*, 2003, pp. 148–159.