Preventing Structured Query Language (SQL) Injection Attacks using PHP Data Object and Prepared Statement

Taylor, Onate Egerton
Rivers State University, Port Harcourt, Nigeria.
tayonate@yahoo.com

Ezekiel, Promise Sochima
Rivers State University, Port Harcourt, Nigeria.
ezekielpromise27@gmail.com

Emmah, Victor Thomas
Rivers State University, Port Harcourt, Nigeria.
vikae2solo@yahoo.com

Abstract

SQL injection is a type of security exploit in which the attacker adds Structured Query Language (SQL) code to a Web form input box to gain access to resources or make changes to data. This paper presents a method for preventing SQL injection attacks on e-commerce websites. This is achieved by using PHP Data Object (PDO) and Prepared Statement for database connection, inserting, updating, selecting and web input forms. The methodology adopted in this paper is Agile Methodology, which follows planning, requirements analysis, designing, coding, testing and documentation in parallel during the stage of production process. A penetration test was being carried out on some websites in which one was fully implemented using PHP Data object and prepared statements and the other was built using a normal PHP script to know if they are injectable or not. The result of the penetration carried out showed that the website which was implemented using PHP Data Object and Prepared Statement was not visible for SQL Injection attack while others which were created using normal PHP connection were visible and injectable to SQL Injection attack. The system was fully prevented from SQL injection attacks using PHP Data Object and Prepared Statement.

Keywords: *SQL Injection, PHP Data Object, Prepared Statement, E-commerce*

1. Introduction

The internet system is an innovative web based solution useful for organization and companies to improve sales of their goods and services. Online and virtual environment is an enormous network of millions of computer allowing communication throughout the world. Internet is platform for carrying out electronic commerce, which consist of the buying and selling of product and services over the global network (internet) and other computer networks. The amount of trade conducted electronically has grown extraordinarily since the advent of internet. A wide variety of commercial activities conducted in this manner spurring and drawing on innovation in electronic funds transfer (EFT), Supply chain management (SCM), internet

marketing, online transaction processing (OTP), electronic Data interchange (EDI), inventory management system and automated data collection system.

SQL Injection Attack (SQLIA) is a type of vulnerability that target database connected to web applications where by malicious codes are inserted, processed and executed. This vulnerability exists when the web application does not perform a proper input validation. A poorly designed web application can be attacked by inserting malicious code in order to get access to the database. There are several places where users can input data in web applications, each of which can provide a SQL injection attack opportunity that can lead to loss of confidentiality, integrity and market value of an organization (Kumar and Pateriya, 2012). Detection of SQL injection before it happens is difficult. Several cases of such illegal intrusion is carried out by the attacker using valid user credentials or by using built-in characteristics of database application such as virulent modification of current SQL queries of web application that are penetrating the delicate parts of the changed databases (Johari and Shanna, 2012). This paper presents a prevention of SQL injection attacks on e-commerce using PHP Data Object and Prepared Statement for database connection, inserting, updating, selecting and web input forms.

2. Related works

SQL Injection is a technique in which attacker injects an input query in order to change the query and illegally gain the access of the database. SQL Injection allows attackers to create, read, update, alter, or delete and modify query in the back-end database and it also allow attackers to access sensitive information such as social security numbers, credit card number and other financial data. When any vulnerability present in web applications then the error is generated. Attacker takes an advantage of this error message as it displayed by the web server depicts the type of database structure that has been used (Bhanderi and Rawal, 2015).

Wei et al. (2007) proposed “Preventing SQL Injection Attacks in Stored Procedures” and through this they provide a novel approach to shield the stored procedures from attack and detect SQL injection from websites. Their method includes runtime check with subsequent application code analysis to eliminate vulnerability to attack. The key method behind this vulnerability attack is that it modifies the composition of the original SQL statement and identifies the SQL injection attack.

Bandhakavi et al. (2007) proposed “CANDID: Preventing SQL Injection Attacks using Dynamic Candidate Evaluation”. In this work, they exhibit a novel and powerful scheme for automatically transforming web applications to render them safe against all SQL injection attacks. The technique for detecting SQL injection is to dynamically mine the programmer-intended query structure on any input, and detect attacks by comparing it against the structure of the actual query issued. We propose a simple and novel mechanism, called CANDID, for mining programmer intended queries by dynamically evaluating runs over benign candidate inputs. This mechanism is theoretically well founded and is based on inferring intended queries by considering the symbolic query computed on a program run. Their approach has been implemented in a tool called CANDID, that retrofits Web applications written in Java to defend them against SQL injection attacks. They report extensive experimental results that show that our approach

performs remarkably well in practice.

Thomas and Williams (2007) proposed "Using Automated Fix Generation to Secure SQL Statements", they covered techniques to remove vulnerabilities from code. They proposed an automated method for removing SQL injection vulnerabilities from code by converting plain text SQL statements into prepared statements. Prepared statements restrict the way that input can affect the execution of the statement. An automated solution allows developers to remove SQL injection vulnerabilities by replacing vulnerable code with generated secure code.

Subburaj and Varsha (2016) proposed "preventing Structured Query Language (SQL) Injection Attacks in Mobile Applications" The technique used is a "program Vaccine" in the Intrusion Detection System that sanitizes the user input queries. All queries passing through the application will enter this program before being allowed to access the database. The "vaccine" is a program written entirely in Python and works only with SQLite databases. Since SQLite databases are created and stored locally, they are used in mobile applications. They are also lightweight and do not require an external interface for interacting with the database. The python file for Vaccine is placed between the mobile application and the database. When the application generates a query, it is passed through Vaccine. The entered query is checked against a set of predefined rules in the blacklist. If any of those characters from the blacklist are present in the query, it is blocked from the database and an error message is displayed.

Merlo et al. (2007) proposed "Automated Protection of PHP Applications against SQL-injection Attacks" the techniques covered an original method to protect application automatically from SQL injection attacks. The original approach combines static analysis, dynamic analysis, and automatic code reengineering to secure existing properties.

Kirti and Vishal (2015) proposed "Security Engine for prevention of SQL Injection and CSS Attacks using Data Sanitization Technique" here, Data Sanitization technique was used to develop perfect Security engine for SQL injection prevention and Cross site scripting attacks. In this method, user input is tested twice before entering the database. First one is to use "Reverse proxy technique" for sanitizing the input given by the user which may attack the database in future. Second, the input is sent to the proxy server before entering the application server, where data cleansing algorithm is triggered to make the user input sanitized. They have used three modules SQL Injection Preventer, Cross Site Scripting Preventer and Analysis Module. In the first two modules the client IP addresses are analyzed. If the attack persists from the same IP address more than three times, then the IP address is blocked or otherwise it is sent to the third module where analysis is done to find the intrusion attack.

3. Methodology

The design methodology used here is Agile Methodology, this methodology follows planning, requirements analysis, designing, coding, testing and documentation in parallel during the stage of production process. Customer involvement during the development process by getting their feedback improves the confidence of making changes; error free, and customer-oriented approach, agile methodologies are the best candidates for e-commerce systems that incorporate

the innovative and dynamic nature of the web. This approach is more efficient and powerful to ensure every piece of functionality is delivered early in the development stage and improves throughout the life of the web application.

3.1 Architectural Design

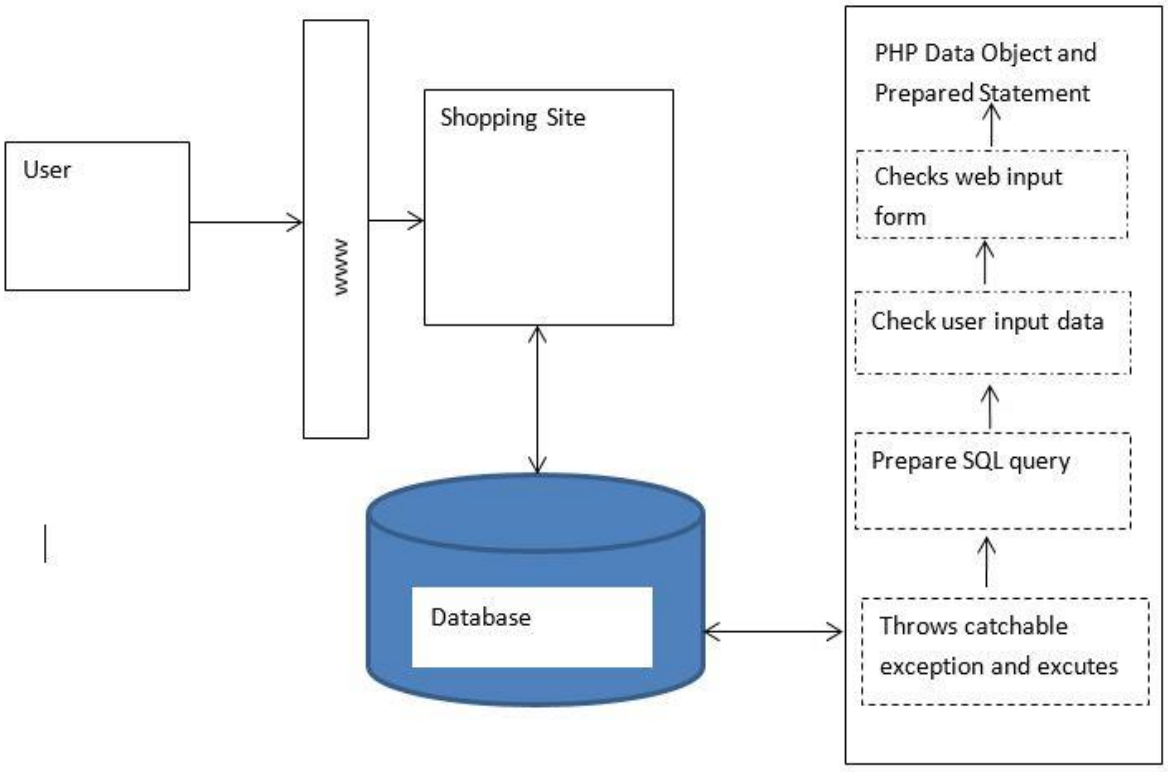

Figure 1: Architecture of the proposed system

This system architecture shows how PHP Data Object and Prepared Statements are being implemented in preventing SQL injection attacks. The system checks web input forms, checks users input data and prepares the SQL query inserted by the user. At the SQL Prepare stage, a statement template is being sent to the database server. The server performs a syntax check and initializes server internal resources for later use. The system throws catchable exception which displays an error message to the user when an invalid string or query has been sent to the database. During execution, the client binds parameter values and sends them to the server. The server creates a statement from the statement templates and bound values to execute it using the previously created internal resource.

4. Result and Discussion

In this paper, sqlmap which is an open source software used to detect and exploit database vulnerabilities and provide option for injecting malicious codes into them in testing the proposed system. Figure 2 contains the number of databases contained on our local server which is 31

databases. Figure 4 contains the number of databases in my local server that is vulnerable to SQL injection attack. A Penetration test was being carried out on our local server which brought out 30 databases which are vulnerable to SQL Injection attack. These 30 databases were vulnerable because of how they were connected and implemented. These 30 databases were all connected using MySQLi extension. The only database which is not injectable that is e-commerce was built using PHP Data Object and Prepared Statement for the connection to the database, insertion and selection of information were done using PDO. In Figure 5, further penetration test was still carried at on some database schema which produces results showing the numbers of tables, columns, user information and number of passwords cracked. These are all in figure 6, 7, 8 and 7.

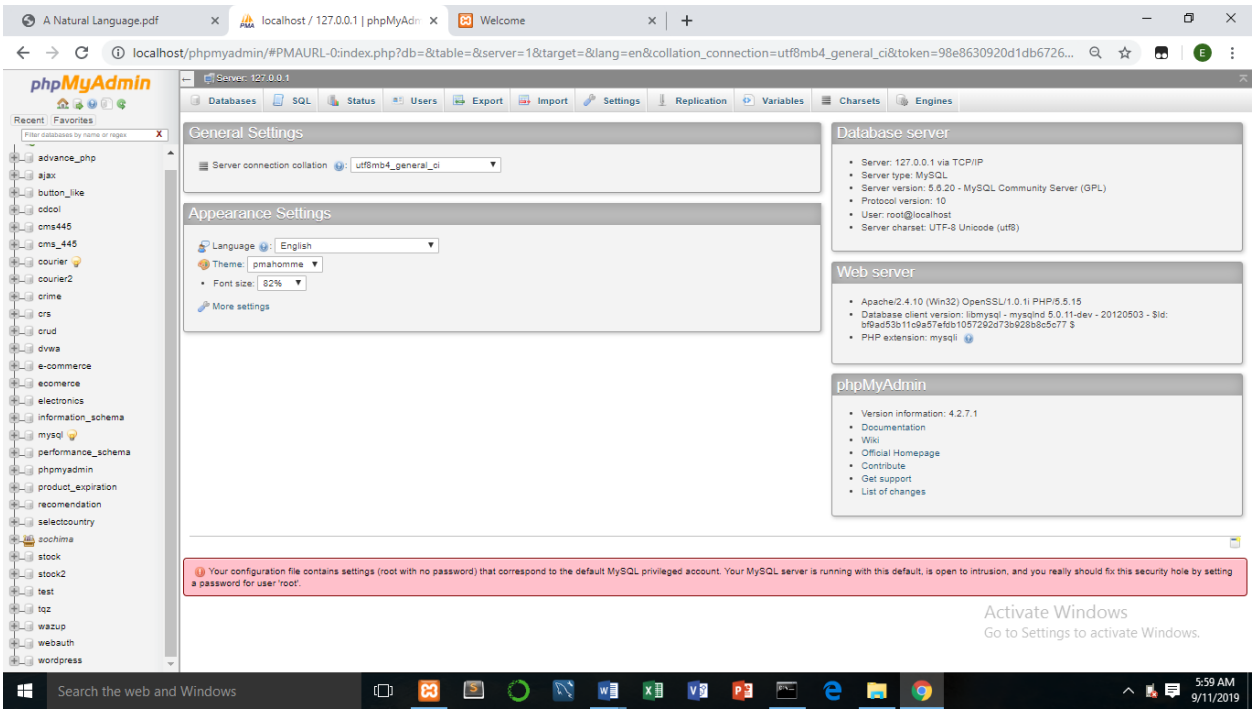

Figure 2: showing the number of database present on our local server in which one is built and connected using P.D.O and prepared statement. The name of the database is called e-commerce.

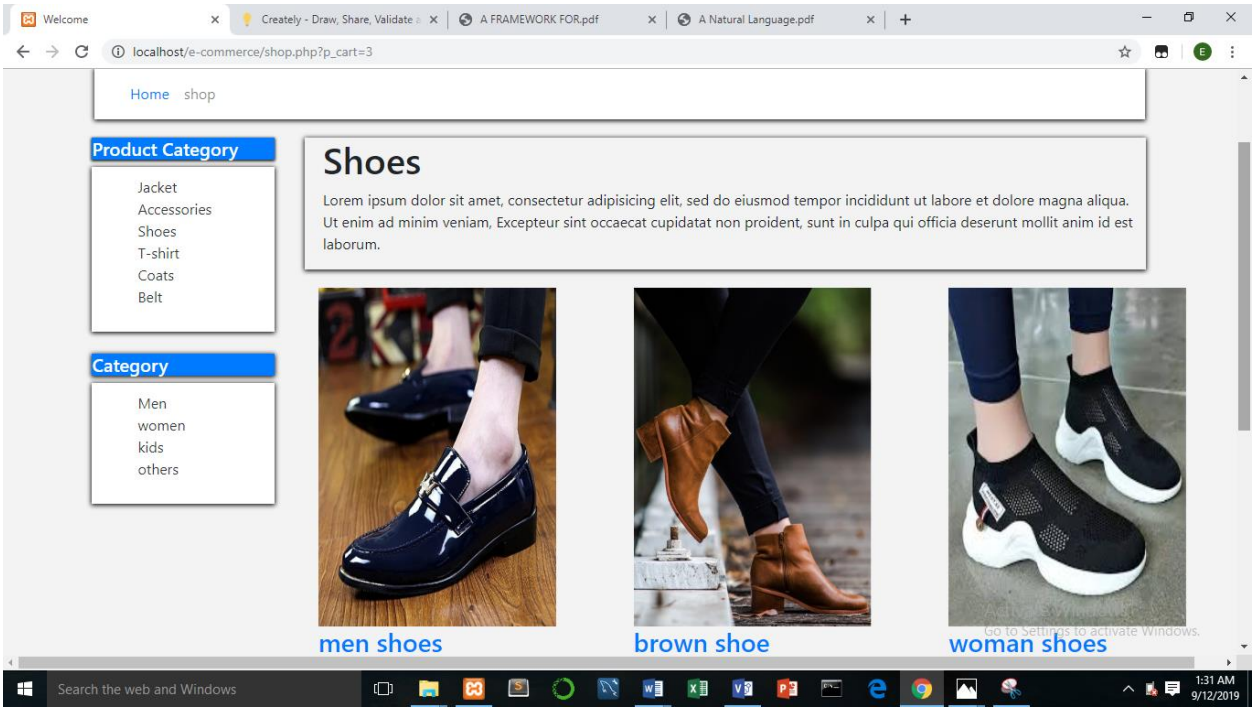

Figure 3: Showing shopping page of the e-commerce website built using PDO and Prepared Statement

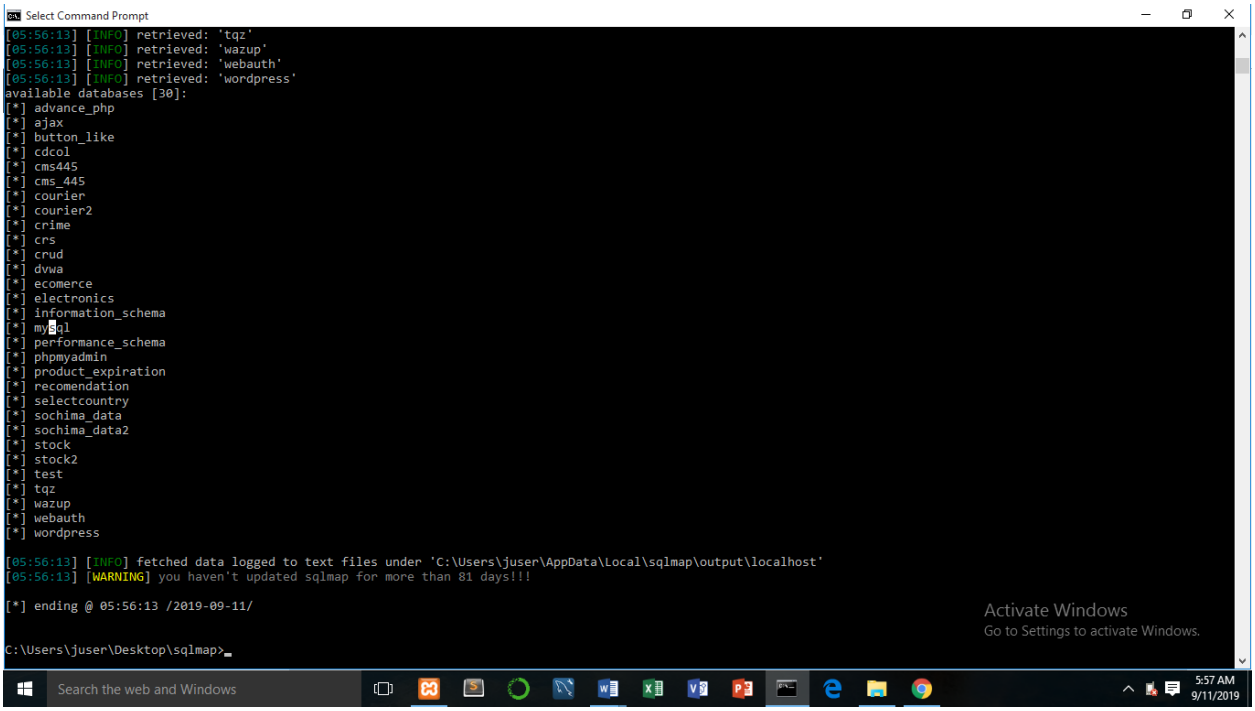

Figure 4: showing the number of database which are vulnerable to SQL Injection attack.

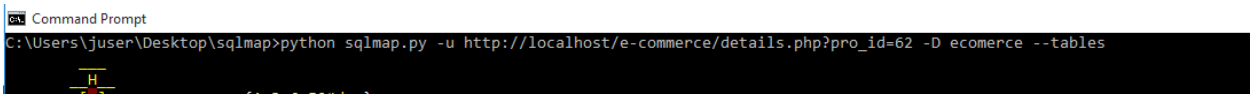

Figure 5: trying to get the number tables present in the database called e-commerce. This is one of the database which was not connected using P.D.O and Prepared Statement

Figure 6: Showing the number of tables present in the database called e-commerce.

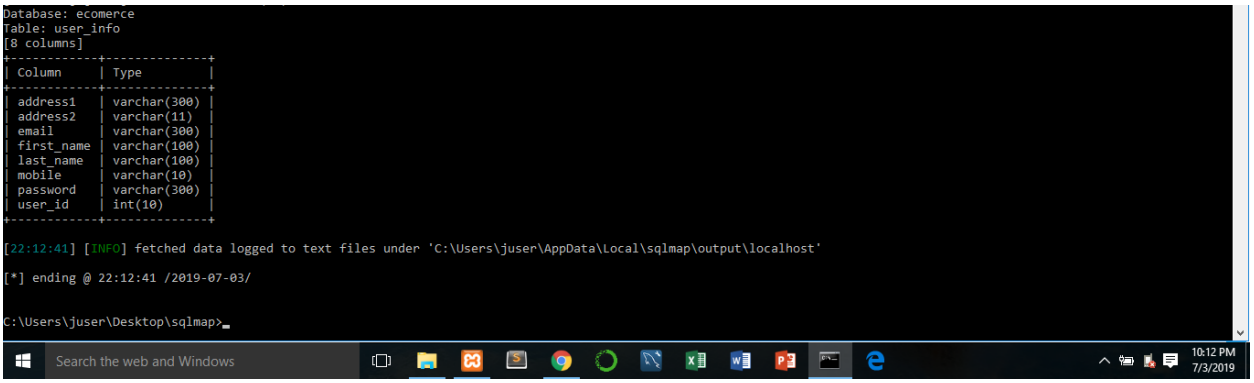

Figure 7: showing the number of columns present in in the database schema called e-commerce

Figure 8: showing the number of users in the system with their mail address

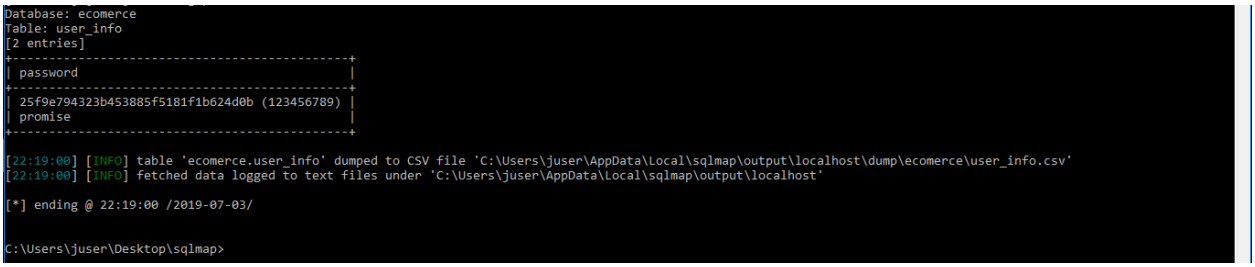

Figure 9: showing the number of password cracked

TABLE 1: Penetration tests Result carried out on three database schemas

Database Schema	Tables	Columns	User Information	Password Cracked
Ecommerce	6	8	2	2
Electronics	10	11	6	6
Crime	3	9	4	4
Stock2	12	6	5	5

Graphical Representation

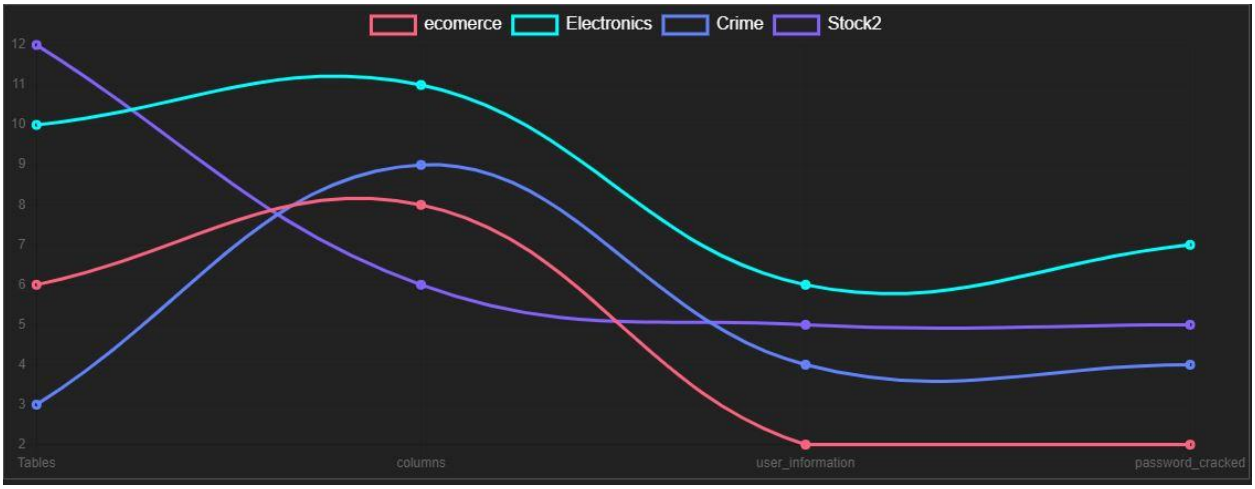

Figure 10 showing a graphical representation of the SQL Injection Penetration Test that was being carried out on 4 database schema which was vulnerable to SQL Injection.

5. Conclusion

In this paper, "Detecting data leaks via SQL injection prevention on E-commerce" was carried out as a result of attackers firing malicious SQL injection queries on e-commerce sites. SQL injection is a security treats in which attackers inserts some malicious codes into web application

forms or performing a blind attack which is being carried out on websites url (GET and POST). This paper talks about SQL injection and their preventive measures of an SQL injection attacks. It was being implemented using PHP Data Object with Prepared Statements in preventing SQL injection attacks on both web input forms and Url. Sqlmap is an open source software which was used in performing penetration test to see if the actual system was truly protected from SQL injection attacks. Further extension can be made to protect applications from SQL Injection and other attack vectors are by using a web application firewall (WAF) like BIG-IP ASM. Also, inserting a WAF into the software development lifecycle allows developers to design applications that are tested, and security policies created, that protect web sites from vulnerabilities like SQL Injection before there are released into production. Deploying a WAF is the first step in creating a protected environment for applications to stay at peak performance while holding vulnerabilities at bay. As future work, we recommend research to be carried out is by adding Run Time Automated Statement and Vulnerability Reinstatement on the system so as to make the system more robust.

References

Bandhakavi, S., Bisht, P. and Madhusudan, P. (2007). CANDID: Preventing SQL Injection Attacks using Dynamic Candidate Evaluation. *Proceeding of the ACM conference on computer and communication security.*

Bhanderi, A. and Rawal, N. (2015). A review and Detection mechanism for SQL injection attacks. *International Journal of Innovative Research in Science, Engineering & Technology*, 4(12), 12446-12452.

Ettore, M., Dominic., L. and Giuliano., A. (2007). Automated Protection of PHP Applications Against SQL-injection Attacks. *Software Maintenance and Reengineering, 11th European Conference IEEE CNF.* Retrieved from http://ieeexplore.ieee.org

Johari, R. and Shanna, P. (2012). A survey on web application vulnerabilities (SQLIA, XSS) exploitation and security engine for SQL injection. *Proceedings of the International Conference on Communication Systems and Network Technologies*, 453-458.

Kirti, R., Vishal, M. (2015). Security Engine for prevention of SQL Injection and CSS Attacks using Data Sanitization Technique. *Proceeding to international journal of innovative Research in computer and communication engineering*, 3(6).

Kumar, P. and Pateriya, P. (2012). A survey on SQL Injection attacks, detection and prevention techniques. *Proceedings of the 3rd International Conference Computing Communication and Networking Technologies*, 1-5.

Stephen, T., Laurie, W. (2007). Using Automated Fix Generation to Secure SQL Statements. *Software Engineering for Secure Systems IEEE CNF*. Retrieved November 6, 2007, from http://ieeexplore.ieee.org

Subburaj, R., Varsha, V. (2016). Preventing Structured Query Language (SQL) Injection Attacks in Mobile Applications. *Proceeding of International Journal of Control Theory and Applications* ISSN: 0974–5572 © International Science Press 9(40).

Wei, M., M., Suraj, K. (2006). Preventing SQL Injection Attacks in Stored Procedures. *Proceedings of the 2006 Australian Software Engineering Conference*.