JUHD / Vol. 3, No.1, March 2017: pp 282- 287
DOI: 10.21928/juhd.20170315.14, e-ISSN: 2411-7757, p-ISSN 2411-7765

Online Database Intrusion Detection System Based on Query Signatures

Alaa Khalil Jumaa
Technical College of Informatics
Sulaimani Polytechnic University
alaa.alhadithy@spu.edu.iq

Awezan Aso Omar
College of Commerce
University of Sulaimani
Kurdistan Region of Iraq, Iraq
Awezan_aso@yahoo.com

Abstract— SQL injection (SQLI) is a major type of attack that threatens the integrity, confidentiality and authenticity or functionality of any database driven web application. It allows the attacker to gain unauthorized access to the back-end database by exploiting the vulnerabilities within the system in order to commit an attack and access resources. Database Intrusion Detection System (DIDS) is the defense against SQLI that is used as a detection and prevention technique to protect any database driven web application. In this paper a proposed system is presented to protect the web application from SQLI. This proposed system uses a new technique of signature- based detection. It depends on secure hash algorithm (SHA-1), which is used to check the signature for the submitted queries and to decide whether these queries are valid, or not. The proposed system can distinguish and prevent hacking attempts by detecting the attacker, blocking his/her request, and preventing him/her from accessing the web application again. The proposed system was tested using Sqlmapproject attacking tool. Sqlmapproject was used to attack the web application (built using PHP and MySQL server) before and after protection. The results show that the proposed system works correctly and it can protect the web application system with good performance and high efficiency.

Keywords— SQL injection, Web application, Database Intrusion Detection System, Hash function, Sqlmapproject.

I. INTRODUCTION

INTERNET, an environment for the world that offer the ability to communicate at an easy manner, with all of the glory… the internet is bless and curse at the same time, two faces of the same coin. Web applications are used as a revolutionary solution for communication by any organization or company that represents smooth accessibility to users and clients over the world via the internet. Web applications correspond to user's input by interacting with the database behind it and output pertinent data for the user. Confidential and critical information usually conserved in the back-end database such as financial records or medical information which are considered to be sensitive data that are desired by attackers. [1].

The SQLIA (structured query language injection attack) can be formed by inserting or "injection" a SQL query by inputting data from the client to the web application at the back-end database. It take advantages of the vulnerability within the system's security policy to manage an effective SQL injection attack that can gain unauthorized access by granting administrative privileges and misleading the SQL query logically to execute commands that deviate from the programmer's original intent and serve the attacker's objective to accessing and reading sensitive data from the database, modify database data by inserting updating or deleting table records, like having a chance to practice administrative functionalities on the database such as locking down the DBMS or claiming the right to see the content of a given file existing on the DBMS file system or in extreme cases commanding the operating system. In another situation SQL injection can constitute and change the outcome of predefined SQL commands execution. [2].

Many approaches exist for a user to input data into a web application, so performing a poor input validation creates the vulnerability that offers a chance for committing SQLI attack and creating a passage to the back-end database without proper authorization which defiantly leads to the loss of secrecy (confidentiality) and integrity of the system and finally negatively altering the market value of the organization [3].

An SQL injection attack occurs when an attacker causes the web application to generate SQL queries that are functionally different from what the user interface programmer intended. For example, from the following SQL statement

SELECT * FROM `users` WHERE username = username and password = password

The user will input his/her legitimate login information to both fields (username and password) and the input data will go through authentication process to validate the login attempt by fetching relative SQL query to a table called users.

The query will be sent to the database to be executed. The values of both username and password are provided from input by the user. Suppose the valid input is:

User = admin

Password=123

The query will be translated as:

Query="SELECT * FROM `users` WHERE username = 'admin' and password = '123'

If this couple of values are to be found in the table users the query will be evaluated as true and the user will be authenticated. But what an attacker would do is by using SQLIA he/she will be able to trick the query logically to execute a command that differ from what is originally intended by the original programmer.

If the attacker use the following user data as an input to the web application:

Username= admin
Password= anything' OR '1' = '1

The query will be manipulated to

SELECT * FROM `users` WHERE username='admin' and password=' anything' OR '1' = '1'

This is a tautology type of injection that will trick the system by submitting a query that evaluates to true for every row on the table (anything' OR '1' = '1) and the attacker will be granted access rights as an authenticated user [4].

Another sort of attack is piggy backed attack that injects a second malicious query for execution, now assume an unauthorized user will input (anything) as a username and ('; drop table xyz --) as password in the login form the output query would be:

SELECT * FROM `users` WHERE username='admin' and password=' '; drop table info --'

The first query will not return any rows and then the query delimiter (";") will be recognized and executed by the underlying database hence deleting the table info from the system. Sometimes a malicious user go even further in his/her attack for example shutting down the DBMS by inserting ';SHUTDOWN; -- into the Username or Password fields then it will produce the following query that result in shutting down the database[5] :

Query="SELECT * FROM `users` WHERE username ='admin' and password = ''; SHUTDOWN; --'

According to the open web application security project (OWASP) ranked SQLI as first of top ten list at 2010, regarding the ease of exploitability and severity of impact.

II. PROBLEM STATEMENT

Database security has been a dialectical issue for many years; behind any web application there is a core database that stores valuable and sensitive information that is presumed to be potential targets for hackers that are trying to intrude their way into gaining financial benefit or espionage or other reasons.

The most dangerous attack technique to be considered is SQL injection. The protection of database from SQLIA may seem like an easy treat, by simply using firewall and applying some input sanitization and restriction techniques with the use of static queries, SQLIA is avoidable. But this strategy fails to accomplish security measures as hackers are always inventing new attack methods that outsmart the system's security policy.

DIDS are presented to protect database systems from SQLIA. Though there are many types and methods which have been presented by researchers but a perfect DIDS does not exist yet. This paper offers a new proven technique to apply database intrusion detection system by using signature based method detection and secures hash algorithm SHA-1 to protect databases from SQL injection attack.

III. RELATED WORK

In order to detect and prevent SQL injection attack many researchers had developed a verity of methods over time, since the first public discussions of SQL injection started around 1998[6].

Chung et al. proposed a misuse detection system called (DEMIDS) which was meant for relational database systems [7].

Lee et al. took advantage of real time data to serve intrusion detection. Data objects were flagged with time-stamps that drew assumptions about update rates that are unknown to intruders [8].

Low et al presented (DIDAFT) that can detect anomalous accesses to the database. This approach distinguishes legitimate access by finger printing their constituent SQL statement [9].

Sharma et al. proposed DIDAR also signature based detection but in real time along with damage control and an auto recovery feature; they built a model for authorized quires for every user derived from currently executing query transactions and later use that model to detect the illegal transactions [10].

Kemalis et al endowed a prototype called **(SQL-IDS).** This approach employs a specification that defines the intended syntactic structure of SQL queries that are produced and executed by the web application and at the same time observes the applications for an execution of query that deviates from the specifications [11].

Then, Randhe et al proposed that a reverse proxy is deployed between the client and the server, it sanitizes applications by using data cleaning algorithm and message digest algorithm, using this method enables the detection of both SQLI and CSS attacks [12].

Ali et al. built a prototype (**SQLIPA)** which is a simple approach yet a strong one to block SQL injection attacks concentrating on the authentication of web driven database. They calculated a hash value of all username and passwords of the system to improve the authentication process [13].

Hidhaya et al proposed a method using a Reverse proxy and MD5 algorithm to search for SQL injection in URL's in user input, by using grammar expression rules. The system showed

significant improvement in eliminating SQLIA on standard tested applications [14].

Swamy et al presented an authentication technique for web applications by encrypting the login data (username, password) by using SHA-3 algorithm to abolish bypass login attempts [15].

Mehta et al created a scheme, (**SQLshield**) that modifies the user input data before the SQL query is executed in the database server by deploying a randomization technique that makes it impossible for the execution outcome of SQL query to deflect from its programmer intended execution [16].

Latha et al presented an efficient method that the detection of SQL injection is done by tampering with the input features of query strings, analysis of query relating to the suitability for both static and dynamic manipulation of user queries [17].

Parchand et al. provided a database detection system and gave preventive measures to avoid or reduce future attacks. A data mining algorithm is used to detect abnormal transactions by structuring a data dependency miner of a banking database system. Their approach extracts read-write dependency rules to be used later for identifying suspicious transactions and also come to the conclusion whether the read-write transaction are violations or done without permission [18].

Souissi et al introduce a categorization-based detection system which supply a structured zone to evaluate, identify, classify and present a defense mechanism against advanced attacks. Their approach contributes in simplifying complicated rule expressions and alert management using a modular design and instinctive rules defined with with a strong expression language. It has the ability to learn from previous attack detecttions and it is not focused on the attack itself instead it is concentrated on attack category; this property helps to sum up defense mechanisms and automates response [19].

Kar et al presents an approach for real-time detection of SQL injection attacks using transformation and resemblance measures. Performing as a database firewall, they proposed a system named (**SQLiDDS**). In a reference hash table the MD5 hash value of each structure is calculated and stored separately which assist the avoidance of repeating the computation of similar incoming query at run-time. They examined the (WHERE) clause only and ignore the (INSERT) queries which was based on two compelling observations made at the time of research [20].

IV. Proposed Database Intrusion Detection System

The proposed system uses a new technique for a signature-based method to detect intrusions, the detection takes place by going through two stages. First stage (offline stage) is building the SQL queries profile by extracting the SQL keywords for each query in the system (eliminate all other words in the query), and these extracted queries are encrypted using SHA-1 producing signatures of safe query. These signatures are saved in the text file called QUERYPROFILE.

The second stage (online stage) is the detection stage, this phase is done by taking query input from user (possible attacker) , extract the SQL keywords, and produce an SHA-1 signature of this query (SQL Keywords) , then the product signature is compared to the signatures in the QUERYPROFILE audit file, if a match found then the program lets the query pass and classify it as a valid query, if not then it distinguish the query as an intrusion, stops the query from executing, producing an alert and getting the IP address of the attacker and prohibit that user from entering the web-site again by blocking his IP.
(Figure 1) show the general architecture for the proposed system.

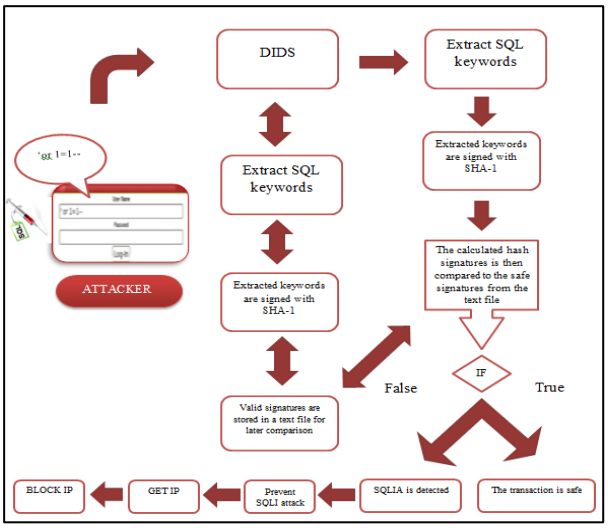

Fig 1. General architecture of the proposed system

In the first stage (offline audit file), the program was written in JAVA language which is used to create a signature for all system queries. All the queries work in the website must be entered into this program and all extracted signatures will be saved in the offline audit file called QUERYPROFILE.txt. Figure (2) shows the flowchart for this program.

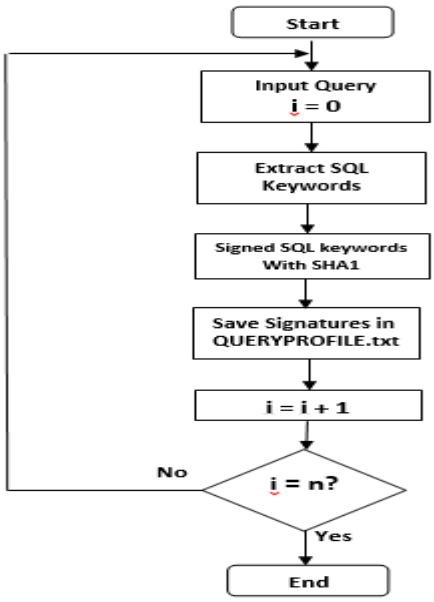

Fig 2. Flowchart for creating queries signature

Where n = maximum number of queries, and i= 1, 2, 3 … n.

There are two program in the second stage (online detection), these programs used to detect the intrusion (SQL injection), prevent it from access the DB system and also from connecting to the website again. Figure (3) show the flowchart for the combined programs.

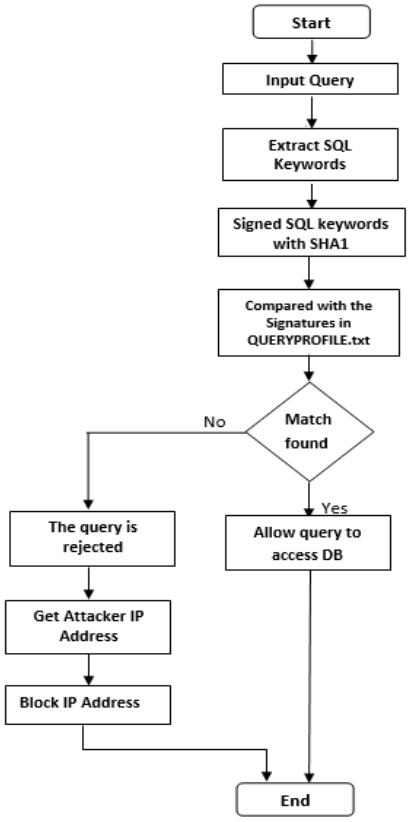

Fig 3. Flowchart for checking input queries

Fist program written in JAVA language and converted to JAR file. This file are injected in the web pages and it will be invoked directly before the input query tries to access the database system. This program extracts the SQL keywords from the input query, signs it using SHA-1 algorithm and compares the result signature with the signatures in the QUERYPROFILE,txt audit file. According to this comparison the system can decide that this query is a SQL injection query or a normal query.

The second program written in BASH-Shell language from the Linux OS. This program is invoked after the system classifies the input user as an intrusion, this program then takes the IP address for the input user and prevents him from accessing the website again by blocking this IP.

V. PROPOSED SYSTEM IMPLEMENTATION AND RESULTS

The experiments for the proposed system performed on a notebook CPU core i7 2.4-GHz and 8 GB memory, Apache_2.4.10 HTTP web server and MySQL-5.0.12 server are installed under Linux-Debian-8.3 Operating System. The Students Attendance Website was used in this experiment. This website is used in two cases: the first case applies the attack tools to the non-protected website (name of site is AttSytem), and for the second case the attack tools are applied to the proposed protection system (name of site is AttSytem1).

The first test for the proposed system is a traditional test, it can be done by a malicious input for SQL injection like use " OR '1' = '1' ". Figure (4) and (5) shows that the SQL injection is working and the malicious user can access the website (AttSytem) by using a fake username and password. Figure (6) and (7) show that the malicious user cannot access the protected website (AttSytem1) and the system detects him/her and give the alert about this malicious attack.

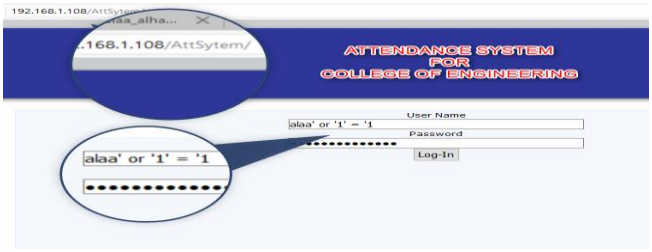

Fig 4. Malicious try to access non- protect website

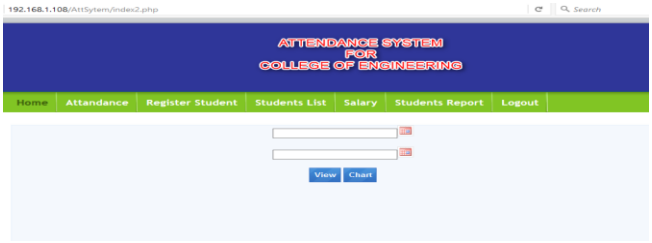

Fig 5. Malicious access the non- protect website

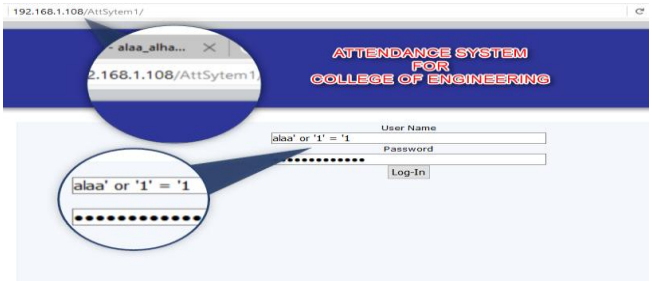

Fig 6. Malicious try to access protect website

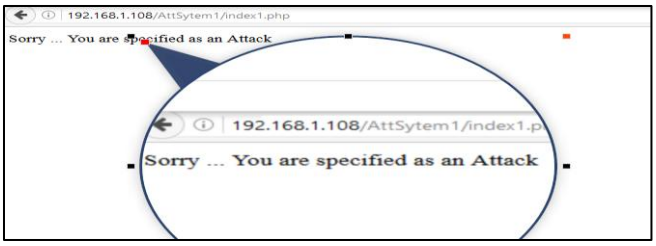

Fig 7. Malicious failed for accessing protect website

The second test for the proposed system implemented by *Sqlmapproject* package [21], this package was written by Python language and it was used as an attached tool for the website systems. At first this tool used to attack the website before protection ("AttSytem"). Figure (8) show how the *Sqlmapproject* tool try to attack the website.

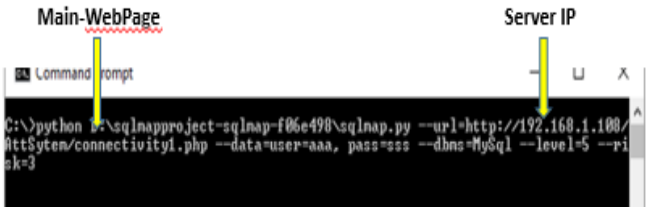

Fig 8. Sqlmapproject try to attack the non-protect website

From figure (9) it can be seen that the *Sqlmapproject* tool hacked the website system with the total 15366 HTTP requests and it needed about 361.0 seconds. Also it can be seen that the Operating System type, Webserver version and MySQL Server version are extracted.

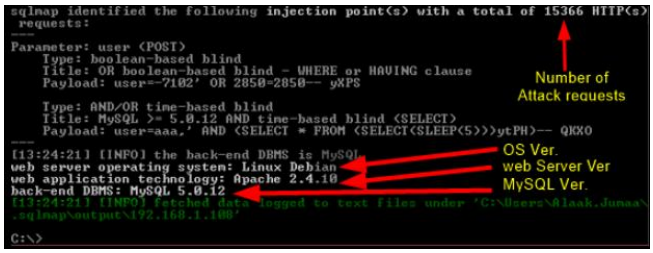

Fig 9. *Sqlmapproject* hacked non-protect website

From figures (10) and (11) it can be seen that the name of database are extracted too.

Fig 10. *Sqlmapproject* try to extract Database name

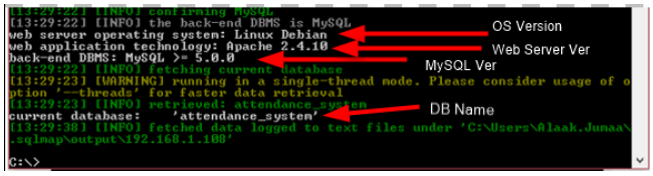

Fig 11. *Sqlmapproject* gained the Database name

Figure (12) shows that the USER table for this database (which includes username and the password) was hacked and the table contents are extracted

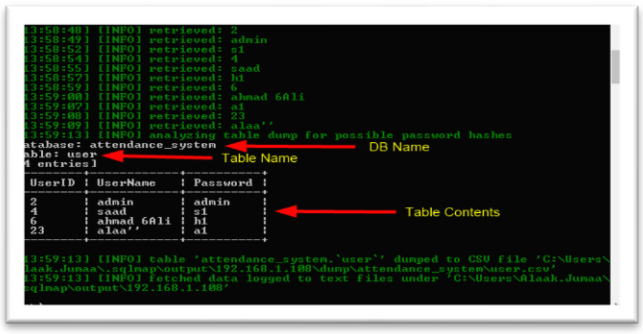

Fig 12. *Sqlmapproject* gained the USER tables contents

Figure (13) shows how the *Sqlmapproject* tool tries to access the website ("AttSytem1") which is protected by the proposed system, and Figure (14) shows that the attacking tools failed to access the protect website. The proposed system will check all the request queries used by attacking tools and prevent them from accessing the database system. In this test the *Sqlmapproject* tool used about 240000 HTTP requests with total time 6137.0 seconds.

Fig 13. Sqlmapproject try to attack the protect website

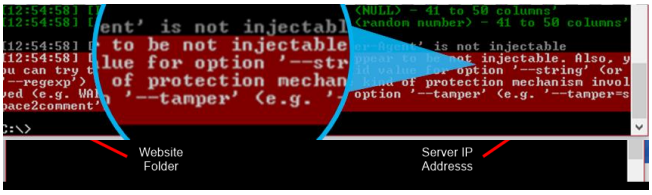

Fig 14. *Sqlmapproject* attack failed to access protect website

Figures (15) and (16) shows the needed time and number of HTTP requests used by *Sqlmapproject* for accessing the non-protected and protected websites.

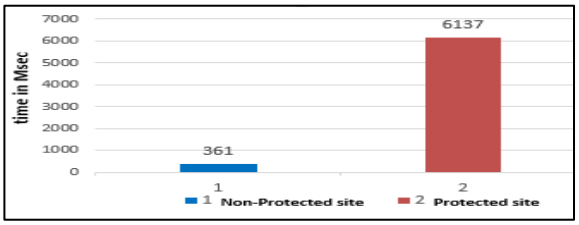

Fig 15. *Sqlmapproject time used to attack non-protected and protected site*

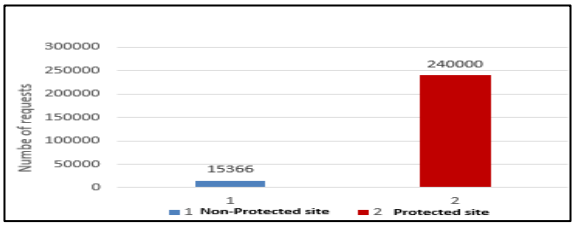

Fig 16. *Sqlmapproject request used to attack non-protected and protected site*

Figure (17) show the average access time which are needed to access both non-protected and protected website. From this figure it can be seen that the protected website was responding a little slower than the non-protected website, this difference is not effect for the system performance because it is so little time (in Milliseconds).

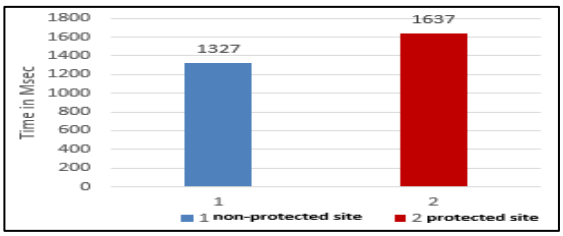

Fig 17. *Average access time for non-protected and protected website*

From these implementations and tests the results show that the proposed system was able to detect all the intrusions injected by *Sqlmapproject* and it can protect the website and the database system with a good performance and high efficiency.

VI. CONCLUSION

In this paper a new technique was proposed for a database intrusion detection system. The proposed system works as an online detection system, it can detect all types of SQL injection attacks and prevent them from accessing the website again. Experimental results show that the proposed system solution is feasible in terms of efficiency and completeness. Furthermore, the website access time is not affected too significantly when the proposed system is used. In future work, this system needs to be developed to protect the websites from internal attacks, this can be done by extracting database-user behavior form the user log file and builds the Intrusion Detection System by using Data Mining or Neural Network techniques.

ACKNOWLEDGMENT

We would like to give thanks to M. Saad Muhammed Abed for providing the website which is used in this work. The authors also gratefully acknowledge the reviewers whose help to improve the work by their comments and suggestions was very helpful..

REFERENCES

[1] Khan, D.R.P.M.M.S., A Survey of Sql Injection Countermesures. International Journal ofComputer Science & Engineering Survey (IJCSES) Vol.3.No, June 2012.

[2]OWASP. SQL Injection. Available from: https://www.owasp.org/index.php/SQL_Injection.

[3] Kumar, P. and R. Pateriya. A survey on SQL injection attacks, detection and prevention techniques. in Computing Communication & Networking Technologies (ICCCNT), Third International Conference on. 2012. IEEE.

[4] Kemalis, K. and T. Tzouramanis. SQL-IDS: a specification-based approach for SQL-injection detection. in Proceedings of the 2008 ACM symposium on Applied computing. 2008. ACM.

[5] w3resource. SQL Injection. 2016; Available from: http://www.w3resource.com/sql/sql-injection/sql-injection.php.

[6] Kerner, S.M., How Was SQL Injection Discovered?, in eSecurityPlanet. 2013

[7] Chung, C.Y., M. Gertz, and K. Levitt, Demids: A misuse detection system for database systems, in Integrity and Internal Control in Information Systems. 2000, Springer. p. 159-178.

[8] Lee, V.C., J.A. Stankovic, and S.H. Son. Intrusion detection in real-time database systems via time signatures. in Real-Time Technology and Applications Symposium, 2000. RTAS 2000. Proceedings. Sixth IEEE. 2000.

[9] Low, W.L., J. Lee, and P. Teoh. DIDAFIT: Detecting Intrusions in Databases Through Fingerprinting Transactions. in ICEIS. 2002..

[10] Sharma, A., DIDAR–Database Intrusion Detection with Automated Recovery. National Institute of Technology, 2007.

[11] Kemalis, K. and T. Tzouramanis. SQL-IDS: a specification-based approach for SQL-injection detection. in Proceedings of the 2008 ACM symposium on Applied computing. 2008.

[12] Randhe, K. and V. Mogal, Defense against SQL Injection and Cross Site Scripting Vulnerabilities, International Journal of Science and Research (IJSR), Volume 3 Issue 11, November 2014.

[13] Ali, S., S. Shahzad, and H. Javed, Sqlipa: An authentication mechanism against sql injection. European Journal of Scientific Research, 2009.

[14] Hidhaya, S.F. and A. Geetha, Intrusion Protection against SQL Injection Attacks Using a Reverse Proxy. SIPM, FCST, ITCA, WSE, ACSIT, CS & IT, 2012.

[15] Swamy, S., P. Kumar, and V. DEV, IMPR OVED AUTHENTICATION TECHNIQUE TO PROTEC T WEB APPLICATIONS. International Journal of Computer Science and Engineering (IJCSE) ISSN (P): p. 2278-9960.

[16] Mehta, P., J. Sharda, and M.L. Das. SQLshield: Preventing SQL Injection Attacks by Modifying User Input Data. in International Conference on Information Systems Security, Springer 2015..

[17] Latha, R. and E. Ramaraj, SQL Injection Detection Based On Replacing the SQL Query Parameter Values. International Journal of Advanced Trends in Computer Science and Engineering · August 2015

[18] Parchand, D. and H. Khanuja, Framework to Detect Malicious Transactions in Database System. International Journal of Computer Applications, 2015..

[19] Souissi, S. Toward a novel classification-based attack detection and response architecture. in Network of the Future (NOF), 2015 6th International Conference on the. IEEE, 2015..

[20] Kar, D., S. Panigrahi, and S. Sundararajan. SQLiDDS: SQL injection detection using query transformation and document similarity. in International Conference on Distributed Computing and Internet Technology. V, 2015..

[21] Sqlmapproject package avilible from: https://pypi.python.org/pypi/sqlmap