Web Anomaly Misuse Intrusion Detection Framework for SQL Injection Detection

Shaimaa Ezzat Salama, Mohamed I. Marie, Laila M. El-Fangary & Yehia K. Helmy
Information System Department,
Faculty of Computers and Information
Helwan University, Cairo, Egypt

***Abstract*—Databases at the background of e-commerce applications are vulnerable to SQL injection attack which is considered as one of the most dangerous web attacks. In this paper we propose a framework based on misuse and anomaly detection techniques to detect SQL injection attack. The main idea of this framework is to create a profile for legitimate database behavior extracted from applying association rules on XML file containing queries submitted from application to the database. As a second step in the detection process, the structure of the query under observation will be compared against the legitimate queries stored in the XML file thus minimizing false positive alarms.**

Keywords-SQL injection; association rule; anomaly detection; intrusion detection.

I. INTRODUCTION

Database-driven web applications have become widely deployed on the Internet, and organizations use them to provide a broad range of services to their customers. These applications, and their underlying databases, often contain confidential, or even sensitive, information, such as customer and financial records. However, as the availability of these applications has increased, there has been a corresponding increase in the number and sophistication of attacks that target them. One of the most serious types of attack against web applications is SQL injection. In fact, the Open Web Application Security Project (OWASP), an international organization of web developers, has placed SQL injection attack (SQLIA) at the top of the top ten vulnerabilities that a web application can have [1]. Similarly, software companies such as Microsoft have cited SQLIAs as one of the most critical vulnerabilities that software developers must address [2]. As the name implies, this type of attack is directed toward database layer of the web applications. Most web applications are typically constructed in a two- or three-tiered architecture as illustrated in Fig.1 [3].

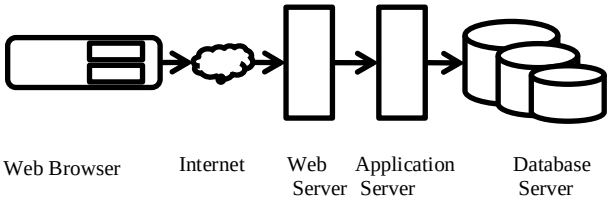

Figure 1. three-tiered architecture

SQLIA is a type of code-injection attack in which an attacker uses specially crafted inputs to trick the database into executing attacker-specified database commands. SQLIAs can give attackers direct access to the underlying databases of a web application and, with that, the power to leak, modify, or even delete information that is stored on them. The root cause of SQLIAs is insufficient input validation [4, 5]. SQLIAs occur when data provided by a user is not properly validated and is included directly in a SQL query [6]. We will provide a simple example of SQLIA to illustrate the problem.

*Select * from users where user_name='' & name & '' and password='' & pass & ''*

The previous example works well if the user supplies valid user name and password. But the problem arises when malicious user exploits the invalidated input and changes the structure of the query to achieve one or more of the different attack intents [4, 7]. The structure of the query will be altered if the user_name attribute have the following value: ' or 1=1 --. The full text of the previous query becomes:

*Select * from users where user_name='' or 1=1*

The injected code will delete the password constraint through the use of SQL comment - - and makes the condition of the query always evaluate to true.

One mechanism to defend against web attacks is to use intrusion detection systems (IDS) and especially network intrusion detection systems (NIDS). IDS use misuse or anomaly or both techniques to defend against attacks [8]. IDS that use anomaly detection technique establish a baseline of normal usage patterns, and anything that widely deviates from it gets flagged as a possible intrusion. Misuse detection technique uses specifically known patterns of unauthorized behavior to predict and detect subsequent similar attempts. These specific patterns are called signatures [8,9].

Unfortunately, NIDS are not efficient or even useful in web intrusion detection. Since many web attacks focus on applications that have no evidence on the underlying network or system activities, they are seen as normal traffic to the general NIDS and pass through them successfully [7, 10, 11].

NIDS are mostly sitting on the lower (network/transport) level of network model while web services are running on the higher (application) level as illustrated in Fig. 2 [11].

In this paper, we propose a framework that combines the two IDS techniques, misuse and anomaly detection techniques, to defend against SQLIA. The main idea of Web Anomaly Misuse Intrusion Detection (WAMID) framework is to create a profile for web application that can represent the normal behavior of application users in terms of SQL queries they submit to the database. Database logs can be used to collect these legitimate queries provided that these logs are free of intrusions. We then use an anomaly detection model based on data mining techniques to detect queries that deviates from the profile of normal behavior. The queries retrieved from database log are stored in XML file with predefined structure. We choose XML format because it is more structured than flat files, more flexible than matrices, simpler and consume less storage than databases.

Association rules will be applied to this XML file to retrieve relation between each table in the query with each condition in the selection part. These rules represent the profile of normal behavior and any deviation from this profile will be considered attack. In order to better detect SQLIA and to minimize false positive alerts, WAMID framework as a second step uses misuse technique to detect any change in the structure of the query. Malicious users sometimes don't change the selection clause but add another SQL statement or add specific keywords to the initial query to check the vulnerability of the site to SQLIA or to perform inference attack. Such types of attack are detected in the second step of the detection process. By comparing the structure of the query under test with the corresponding queries in the XML file the previous malicious actions will be detected.

The rest of the paper will be organized as follows: in section II discusses previous work, section III

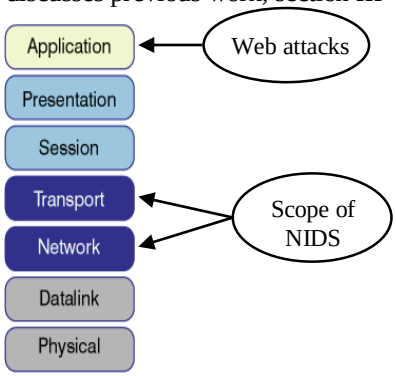

Figure 2. Scope of NIDS

provides a detailed description about the framework and its components. Anomaly and misuse algorithms and a working example will be presented in section IV. Section V concludes the paper and outlines future work.

II. Literature Review

Different researches and approaches have been presented to address the problem of web attacks against databases. Considering SQLIA as top most dangerous attacks, as stated in section I, there has been intense research in detection and prevention mechanisms against this attack [4, 5, 12]. We can classify these approaches into two broad categories: a) one approach is trying to detect SQLIA through checking anomalous SQL query structure. b) another approach uses data dependencies among data items which are less likely to change for identifying malicious database activities. In either of two categories, different researchers take advantage of the benefit of integrating data mining with database intrusion detection in order to minimize false positive alerts, minimizing human intervention and better detect attacks [13]. Moreover, Different intrusion detection techniques are used either separately or together. Different work used misuse technique others used anomaly or mixes the two techniques.

Under the first category and without using data mining technique, Lee et al. in [10] and Low et al. in [14] developed a framework based on fingerprinting transactions for detecting malicious transactions. They explored the various issues that arise in the collation, representation and summarization of this potentially huge set of legitimate transaction fingerprints. Another work that applies anomaly detection technique to identify anomalous database application behavior is presented by Valeur et al. in [15]. It builds a number of different statistical query models using a set of typical application queries, and then intercepts the new queries submitted to the database to check for anomalous behavior.

A general framework for detecting malicious database transaction patterns using data mining was proposed by Bertino et al. in [16, 17] to mine database logs to form user profiles that can model normal behaviors and identify anomalous transactions in databases with role based access control mechanisms. The system is able to identify intruders by detecting behaviors that differ from the normal behavior of a role in a database. Kamra et al. in [18] illustrated an enhanced model that can also identify intruders in databases where there are no roles associated with each user. It employs clustering techniques to form concise profiles representing normal user behaviors for identifying suspicious database activities. Another approach that checks for the structure of the query to detect malicious database behavior is the work of Bertino et al. in [19]. They proposed a framework based on anomaly detection technique and association rule mining to identify the query that deviates from normal database application behavior.

The problem with this framework is that it produces a lot of rules and represents the queries in very huge matrices which may affect tremendously on the performance of rule extraction. Misuse detection technique have been used by Bandhakavi et al. in [20] to detect SQLIA by discovering the intent of a query dynamically and then comparing the structure of the identified query with normal queries based on the user input with the discovered intent. The problem with this approach is that it has to access the source code of the application and make some modifications to the java virtual machine.

Halfond et al. in [21] developed a technique that uses a model-based approach to detect illegal queries before they are executed on the database. In its static part, the technique uses program analysis to automatically build a model of the legitimate queries that could be generated by the application. In its dynamic part, the technique uses runtime monitoring to inspect the dynamically-generated queries and check them against the statically-built model. The system WASP proposed by Wiliam et al. in [22] tries to prevent SQL Injection Attacks

by a method called positive tainting. In positive tainting, the trusted part of the query (static string) is not considered for execution and masked as tainted, while all other inputs are considered. The difficulty in this case is the propagation of taints in a query across function calls especially for the user defined functions which call some other external functions leading to the execution of a tainted query. Different other researches followed the same approach in detection of anomalous SQL query structure in [23, 24].

Researches that belong to the second category of detection which depends on data dependencies are [25, 26, 27, 28]. The work that is based on mining sequential data access patterns for database intrusion detection was proposed by Hu et al. in [25, 26]. Transactions that do not comply with rules generated from read and write sequence sets are identified as malicious transactions. Srivastava et al. offered a weighted sequence mining approach [27] for detecting database attacks. The advantage of the work presented by YiHu et al. in [28] is the automatic discovery and use of essential data dependencies, namely, multi-dimensional and multi-level data dependencies, for identifying anomalous database transactions.

The contribution of this paper is a framework that combines anomaly and misuse detection technique in order to better detect SQLIA. This framework uses association rules with anomaly technique to build the normal behavior of application users and detecting anomalous queries. Moreover, misuse technique is used to check the structure of the query to detect any malicious actions that cannot be detected using anomaly detection technique.

III. THEORETICAL FRAMEWORK

WAMID framework is a database intrusion detection that aims to detect SQLIA at real-time, before queries execution at the database. This is why this framework should run at the database or application server depending on the architecture of web application as depicted in fig. 1. In order to detect all possible attempts of SQL injection, WAMID framework combines the two detection techniques: anomaly and misuse. It depends in the detection of SQLIA on determining the malicious changes that occurred in the SQL query structure. The key idea of our framework is as follows. We build a repository containing set of legitimate queries submitted from the application user to the database. This repository is a set of training records. We then use an anomaly detection approach based on data mining technique to build a profile of normal application behavior and indicate queries that deviates from this normal behavior.

In a second step in the detection process, the framework checks for the presence of dangerous keywords in the query if the latter passes the test of anomaly detection step. We need this step because sometimes the intent of the attacker is to identify the security holes in the site or to infer the structure of the database through the error message returned from the application and this type of SQLIA is called inference [4, 29]. This type of attack cannot be detected through anomaly technique because it doesn't require change in the conditions of the original query but it will be discovered if the structure of the query is compared against its corresponding query in the repository file.

Based on what previously stated we learn that the framework act in two phases: training phase and detection phase. In the training phase the repository file will be created and normal behavior of the application is built. In the detection phase, the framework uses the anomaly and misuse techniques to discover any SQLIA. In the following subsections we will provide a detailed explanation of the framework, its components and how it works.

A. Training Phase

During the training phase the training records are collected from the queries the application send to the database. The source for obtaining these query traces is the database log provided that the latter is free of intrusions. The training phase flow is illustrated in Fig. 3. The challenge here is that to efficiently encode these queries in order to extract useful features from them and accordingly build the application fingerprint. Unlike approach provided in [19], we choose to encode the queries in XML file. The encoding scheme provided by Bertino et al. in [19] result in a large, dense, sparse matrices which may effect on the mining algorithm. XML is more structured than flat files, is supported by query tools like XQuery and XPath to extract data [30]. It is simpler and consumes less space than relational databases and more flexible than matrices.

It is important to identify accurately the structure of the XML file that will represent the features extracted from the query that will contribute in building the application fingerprint. Consider the following query:

Select SSN, last_name from employee where first_name='Suzan' and salary>5000

The encoding scheme of the previous query in XML file is illustrated in Fig. 4. The main advantage of XML format is that nodes may be duplicated upon need. For example the number of project_attribute" node may differ from one "Query" node to another depending on the query itself. This is why it is more suitable to store queries than databases while maintaining flexibility and simplicity.

The XML file illustrated in fig. 4 stores the projection attributes, the from clause and the predicate clause in a more detailed way. It is not important to identify the value of the integer or string literal it is important to determine that there is an integer or string literal or there is another attribute in the right hand side. Another file that should be created during the training phase is the signature file that will be used during the misuse detection phase. As stated before this file contains suspicious keywords that may be considered a sign of SQLIA.

Keywords like for example single quote, semicolon, double dash, union, exec, order by and their hexadecimal representation in order to prevent the different evasion techniques [31]. The important step in the training phase is to build the profile representing the application normal behavior. We will apply association rules [32] on the XML file to extract rules that represent the normal behavior of application users. Different approaches have been proposed to apply association rules on XML data. We direct the reader to [33-35] for an in-depth survey of these approaches. The rules extracted represent

relationship between each table in the query with each predicate in the selection clause.

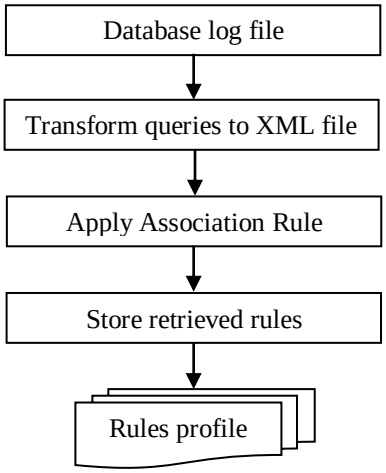

Figure 3. training phase flow

```
<Queries>
<Query id=1>
<command> select </command>
<project_attribute> SSN </project_attribute>
<project_attribute> last_name </project_attribute>
<From> employee </From>
<LHS_condition> first_name </LHS_condition>
<RHS_condition> string Literal </RHS_condition>
<logical_operator> and </logical_operator>
<LHS_condition> salary </LHS_condition>
<RHS_condition> Integer Literal </RHS_condition>
</Query>
</Queries>
```

Figure 4. representation of query in XML file

This is based on an observation that the static part of the query is the projection attribute and the part that is constructed during execution is the selection part [19]. We here add another item to the static part which are the tables in the from clause. We try to make relation between the static part and the dynamic part and extract rule with support and confidence of such relation. Any query that will not match rules extracted and stored in the rules profile will be considered attack. More details about how the rules are extracted are provided in the following subsection.

B. Anomaly Detection Phase

In the previous subsection, we illustrated how the benign queries are collected and captured in XML file in a form enabling the framework from creating the database behavior profile. We apply association rules on the XML file containing legitimate queries and extract rules that can describe the normal behavior of application users. The idea behind building the profile rule is to apply one of association rules algorithms on previously created XML file to extract relation between each table in the query with each selection attribute excluding the literals. Thus the rules extracted have the following format:

$$\text{From} \longrightarrow \text{LHS}$$

$$\text{From} \longrightarrow \text{RHS}$$

Recall the example of employee first name and salary so the rules extracted from this query are:

$$\text{employee} \longrightarrow \text{first\_name}$$

$$\text{employee} \longrightarrow \text{salary}$$

The rules that exceed the minimum support and confidence will be stored in rules profile. These rules represent the profile of how the application behaves normally. Fig. 4 illustrates the flow of detection phase of the framework in general including the anomaly technique. In a typical database application, the input supplied by the user construct the where clause of the query. Meanwhile, the projection clause and the from clause remain static at the run time. So we create a relation between the static and the dynamic part of the query and any change in the where clause by attackers that cannot be derived from the rules profile will be announced as SQLIA. We decided to choose the tables in the from clause from the static part of the query instead of the projection attributes because the former is more general and contain the latter and thus generating less rules and make it easier in comparison. Lets return to our query in the previous subsection and change it a little bit: select SSN, last_name from employee where first_name=' "& fname &" ' and salary> " & empsal. If the attacker needs to retrieve all values from employee table then the following code will be injected to form this new query:

Select SSN, last_name from employee where first_name='' or 1=1 - -

Before executing this query, rules should be extracted first and compared to the rules in the rules profile. The relation between tables and attributes will be compared against rules stored in the profile rules file. The two relations under test from the previous example are:

$$\text{employee} \longrightarrow \text{first\_name}$$

$$\text{employee} \longrightarrow 1$$

The first relation exists in the rules profile but no such rule match the second relation. So the query is announced as anomaly query.

C. Misuse Detection Phase

In a second step in the detection process and after the anomaly detection phase, comes the role of misuse detection. The need to this step comes from the fact that SQLIA doesn't only change the conditions in the query but it also may provide information about the database schema or check the vulnerability of the application to SQL injection. This is done through adding to the query some keywords that may change the behavior of the query or return information about the database through database errors without changing the predicates of the query. In such case, the anomaly detection phase will not be able to discover such attack. For example consider the following query:

*Select * from employee where SSN=10*

If the attacker just adds a single quote at the end of the query, this will result in error message that may inform the attacker that the site is vulnerable to SQLIA. Another example of attack is just adding the keyword "order by" to the query without changing the selection attributes like:

*Select * from employee where SSN=10 order by 1*

Trying to execute this query several times will give attacker information about the number of attributes in the table. This is why this step is needed in the detection process. Moreover, the framework doesn't announce the query as anomaly just by finding these keywords in the query because it may be part of the legitimate query itself resulting in false positive alarm. This is why the framework checks for the structure of the query under test with the corresponding query stored in XML file. The detection phase flow of the framework in Fig. 4 illustrates this process. These suspicious keywords are stored in file called "forbidden keywords". This file contains SQL keywords like single quote,

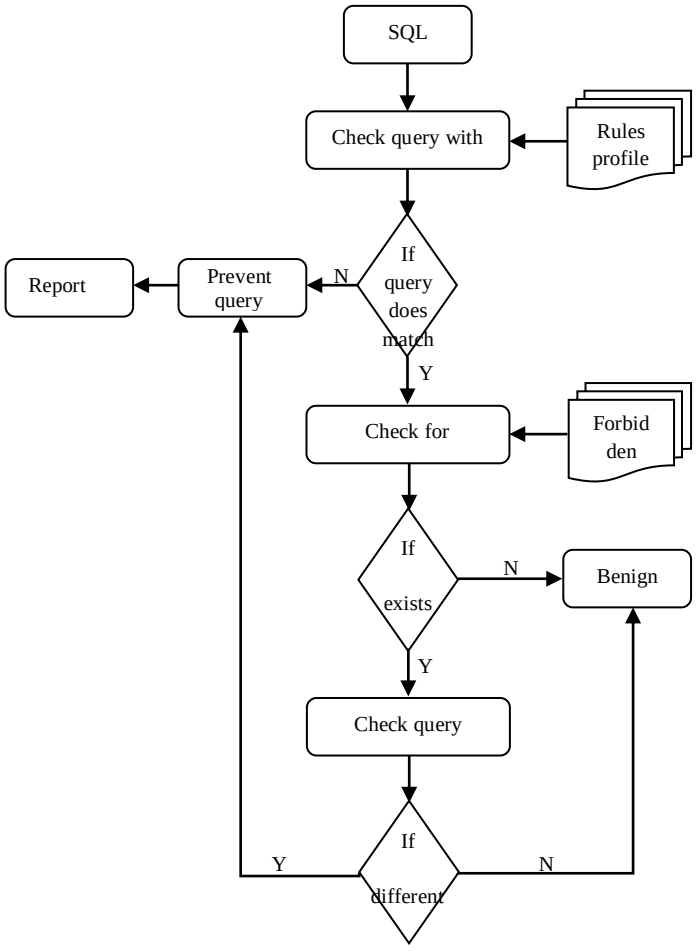

Figure 5. anomaly misuse detection flow phase

semicolon, union, order by, exec and their hexadecimal representation to avoid the different evasion techniques. After confirming the existence of one or more of these keywords, we use XQuery to retrieve queries from XML file with the same projection attributes and same from clause. Then comparison is done between query under test and the queries retrieved by XQuery from XML file. If there is no match then the query is announced anomaly.

IV. Algorithm And Working Example

In this section we present algorithms for anomaly and misuse detection. In addition, we provide a working example illustrating how the WAMID framework performs the detection.

A. Anomaly detection algorithm

Algorithm anomaly_detection( )
Input: rules profile, query under test
Output: True if query is intrusion, false otherwise
Begin

Extract relation between tables and selection attributes
Store extracted relations in query_relation
/* query_relation is array to store relations*/
For each relation r in query_relation
If (r is found in rule profile)
Score=score+1
If score=length (query_relation)
Return false
Else
return true
End

B. Misuse detection algorithm

Algorithm misuse_detection( )
Input: forbidden keywords file, query under test, XML file
Output: True if query is intrusion, false otherwise
Begin
For each keywords k in forbidden keywords
If k not exists in query
Return false
Else
Use XQuery language to extract relevant queries from XML file
If query structure doesn't match any retrieved queries
Return True
Else
Return false
End

C. Working example

In order to provide better understanding of the anomaly and misuse detection in WAMID framework, we provide in this subsection example of the flow of detection either anomaly or misuse in this framework. The following represents example of queries submitted from application to database:

- Select product_name, description from product where product_id=?
- Select product_name, description from product where salary<?
- Select * from product where product_name=? order by product_name

- Select product_name, description from product where salary<? and category_id=?

The representation of the previous queries in XML file is illustrated in Fig. 5.

```
<Queries>
<Query id=1>
<command> select </command>
<project_attribute > product_name </project_attribute>
<project_attribute > description      </project_attribute>
<from> product </from>
<LHS>product_id </LHS>
<RHS> Integer_literal </RHS>
</Query>
<Query id=2>
<command> select </command>
 <project_attribute > product_name </project_attribute>
<project_attribute > description      </project_attribute>
<from> product </from>
<LHS> salary </LHS>
<RHS> Integer_literal </RHS>
</Query>
<Query id=3>
<command> select </command>
<project_attribute > * </project_attribute>
<from> product </from>
<LHS> product_name </LHS>
<RHS> string_literal </RHS>
<order by> product_name</order by>
</Query>
<Query id=4>
<command> select </command>
<project_attribute > product_name </project_attribute>
<project_attribute > description      </project_attribute>
<from> product </from>
<LHS> salary </LHS>
<RHS> Integer_literal </RHS>
<logical_operator> and </logical_operator>
<LHS> category_id </LHS>
<RHS> Integer_literal </RHS>
</Query>
```

Figure 6. XML file representing queries

After applying association rule algorithm like for example Apriori on this XML file, the resulting rules will stored in rules profile file in Fig. 6.

In the following we will provide sample of malicious and legitimate queries.

- Select product_name, description from product where product_id=5’

The first step in the framework is to identify relation between tables and selection attributes in the query.

Product ⟶ product_id

Second, the framework searches in the rules profile for this relation. It already exists. But this is not the end of the detection flow. The second step is to check for suspicious keywords in the query. The query already contains one of the suspicious keywords which is single quote.

So XQuery language is used to extract queries from the XML file with same projection attributes and same from clause. By comparing the structure of the query under test and query returned from the XML file we will find that query shouldn’t contain the single quote and thus it is announced as anomaly.

- Select product_name, description from product where product_id=1 or 1=1- -

The value 1 for the product_id may be right or maybe wrong anyway we have here an injected code to retrieve data of all products. First we extract relations.

Product ⟶ product_id relation 1
Product ⟶ 1 relation 2

By searching in the rules profile we find a rule for the first relation but no rule for the second relation so the query is announced immediately anomaly.

- Select product_name, description from product where product_id=1 order by 1- -

As we previously stated there is a rule matching the relation 1 in the previous example.

By examining the query against the forbidden keywords file we find two keywords: order by and double dash. By examining the original query in the XML file we find that this query is anomaly because it doesn’t contain order by or double dash.

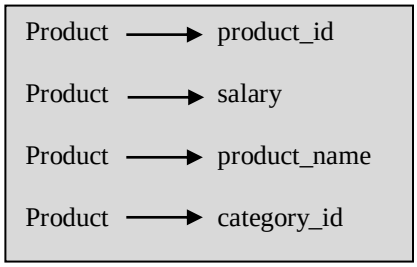

Figure 7. Extracted rules from XML file

- Select * from product where product_name=’food’ order by product_name

The extracted relation from this query is:

Product ⟶ product_name

This relation exists in the rules profile. And also one of the forbidden keywords exists so the structure of the query should be examined. After examining the structure of the query the framework identifies that the query is legitimate.

V. Conclusion And Future Work

Database intrusion is a major threat to any organization storing valuable and confidential data in databases. This is increasingly more so as the number of database servers connected to the Internet increases rapidly. Existing network-based and host-based intrusion detection systems are not sufficient for detecting database intrusions. We have introduced a framework based on anomaly and misuse detection for discovering SQLIA. We have presented a new encoding technique for SQL queries in XML file in a way enabling the extraction of normal behavior of database application. We then used data mining technique for fingerprinting SQL statements and use them to identify SQLIA. This set of fingerprints is then used to match incoming

database transactions. If the set of fingerprints in the legitimate set is complete, any incoming transaction whose fingerprint does not match any of those in the legitimate set is very likely to be an intrusion. A second step in the framework is the misuse technique in which XQuery is used to match the incoming query with queries stored in XML file after ensuring that one or more of the suspicious keywords exist in the query.

We plan to perform experiments to apply this framework to identify its performance in detecting attacks and include comparisons to other approaches. This work may be extended to include detection against other attacks like cross site scripting.

REFERENCES

[1] [1] http://www.owasp.org/index.php, OWASP Top 10-2010 document

[2] M. Howard and D. LeBlanc, "Writing Secure Code", Microsoft Press, 2002

[3] Amit Kumar Pandey, "SECURING WEB APPLICATIONS FROM APPLICATION-LEVEL ATTACK", master thesis, 2007

[4] W.G.Halfond, J.Viegas, and A.Orso, "A classification of SQL-Injection Attacks and Countermeasures", in proceeding of the International Symposium on Secure Software Engineering (ISSSE), 2006

[5] Kindy, D.A.; Pathan, A.K, "A survey on SQL injection: Vulnerabilities, attacks, and prevention techniques", in proceedings of IEEE 15th International Symposium on Consumer Electronics (ISCE), 2011

[6] G. Wassermann and Z. Su, "An Analysis Framework for Security in Web Applications", In Proceedings of the FSE Workshop on Specification and Verification of Component-Based Systems (SAVCBS 2004), pages 70–78, 2004.

[7] San-Tsai Sun, Ting Han Wei and Stephen Liu, "Classification of SQL Injection Attacks", University of British Columbia : Sheung Lau Electrical and Computer Engineering, 2007

[8] S.Axelsson, "Intrusion detection systems: A survey and taxonomy", Technical Report, Chalmers Univ., 2000

[9] Marhusin, M.F.; Cornforth, D.; Larkin, H., "An overview of recent advances in intrusion detection", in proceeding of IEEE 8th International conference on computer and information technology CIT, 2008

[10] Lee, S. Y., Low, W. L., and Wong, P. y.: Learning Fingerprints for a Database Intrusion Detection System. In the Proceedings of the 7th European Symposium on Research in Computer Security, 2002

[11] C.J. Ezeife, J. Dong, A.K. Aggarwal, "SensorWebIDS: A Web Mining Intrusion Detection System", International Journal of Web Information Systems, volume 4, pp. 97-120, 2007

[12] N. Khochare, S. Chalurkar ,S. Kakade, B.B. Meshramm, "Survey on SQL Injection attacks and their countermeasures", International Journal of Computational Engineering & Management (IJCEM), Vol. 14, October 2011

[13] S. F. Yusufovna., "Integrating Intrusion Detection System and Data Mining", International Symposium on Ubiquitous Multimedia Computing, 2008

[14] Low, W. L., Lee, S. Y., Teoh, P., "DIDAFIT: Detecting Intrusions in Databases Through Fingerprinting Transactions", in Proceedings of the 4th International Conference on Enterprise Information Systems (ICEIS), 2002

[15] F. Valeur, D. Mutz, and G.Vigna, "A learning-based approach to the detection of sql injection attacks", in proceedings of the conference on detection of intrusions and Malware and vulnerability assessment (DIMVA), 2005

[16] Bertino, E., Kamra, A, Terzi, E., and Vakali, A, "Intrusion detection in RBAC-administered databases", in the Proceedings of the 21st Annual Computer Security Applications Conference, 2005

[17] Kamra A, Bertino, E., and Lebanon, G.,"Mechanisms for Database Intrusion Detection and Response", in the Proceedings of the 2nd SIGMOD PhD Workshop on Innovative Database Research, 2008

[18] Kamra A, Terzi E., and Bertino, E.,"Detecting anomalous access patterns in relational databases", the VLDB Journal VoU7, No. 5, pp. 1063-1077, 2009

[19] Bertino, E., Kamra, A, and Early, J., "Profiling Database Application to Detect SQL Injection Attacks", In the Proceedings of 2007 IEEE International Performance, Computing, and Communications Conference, 2007

[20] Bandhakavi, S., Bisht, P., Madhusudan, P., and Venkatakrishnan V., "CANDID: Preventing sql injection attacks using dynamic candidate evaluations", in the Proceedings of the 14th ACM Conference on Computer and Communications Security, 2007

[21] Halfond, W. G. and Orso, A , "AMNESIA: Analysis and Monitoring for Neutralizing SQL-Injection Attacks", in Proceedings of the 20th IEEE/ACM international Conference on Automated Software Engineering, 2005

[22] William G.J. Halfond, Alessandro Orso, and Panagiotis Manolios, "WASP: Protecting Web Applications Using Positive Tainting and Syntax-Aware Evaluation", IEEE Transactions on Software Engineering, Vol. 34, No. 1, pp 65-81, 2008

[23] Buehrer, G., Weide, B. w., and Sivilotti, P. A, "Using Parse Tree Validation to Prevent SQL Injection Attacks", in Proceedings of the 5th international Workshop on Software Engineering and Middleware, 2005

[24] Liu, A, Yuan, Y., Wijesekera, D., and Stavrou, A, "SQLProb:A Proxy-based Architecture towards Preventing SQL Injection Attacks", in Proceedings of the 2009 ACM Symposium on Applied Computing, 2009

[25] Hu, Y., and Panda, B., "A Data Mining Approach for Database Intrusion Detection", In Proceedings of the 19th ACM Symposium on Applied Computing, Nicosia, Cyprus ,2004

[26] Hu, Y., and Panda, B., "Design and Analysis of Techniques for Detection of Malicious Activities in Database Systems", Journal of Network and Systems Management, Vol. 13, NO. 3,2005

[27] Srivastava, A, Sural S., and Majumdar, AK., "Database Intrusion Detection Using Weighted Sequence Mining", Journal of Computers, vol.1, no. 4 ,2006

[28] Yi Hu; Campan, A.; Walden, J.; Vorobyeva, I.; Shelton, J, "An effective log mining approach for database intrusion detection", in proceedings of IEEE international conference on systems man and cybernetics (SMC), 2010

[29] David Litchfield, "Data-mining with SQL Injection and Inference",An NGSSoftware Insight Security Research, September 2005

[30] World Wide Web Consortium. XQuery 1.0: An XML Query Language (W3C Working Draft). http://www.w3.org/TR/2002/WDxquery-20020816, Aug. 2002.

[31] O. Maor and A. Shulman, "SQL Injection Signatures Evasion", White paper, Imperva, April 2004. http://www.imperva.com/application defense center/white papers/sql injection signatures evasion.html

[32] Han J., Kamber M., "Data Mining: Concepts and Techniques", Maurgan Kaufmann, 2nd edition, 2006

[33] Jacky W.W.Wan, Gillian Dobbie, "Mining Association Rules from XML Data using XQuery", in proceeding of ACM 2nd workshop on Australasian information security, Data Mining and Web Intelligence, and Software Internationalization, 2004

[34] Qin Ding, "Data Mining on XML Data", in Encyclopedia of Data Warehousing and Mining, 2nd edition, Vol. 1, ed. John Wang, IGI Global, 2008, pp. 506-510

[35] Qin Ding and Gnanasekaran Sundarraj, "Mining Association Rules from XML Data", in Data Mining and Knowledge Discovery Technologies, ed. David Taniar, IGI Global, 2008. pp. 59-71