Application of a Modified C4.5 Learner’s Algorithm to SQL Query Classification

E. E. Ogheneovo and P. O. Asagba
Department of Computer Science,
University of Port Harcourt
Port Harcourt, Nigeria
1 edward_ogheneovo@yahoo.com, 2 pasagba@yahoo.com
08038591480

Corresponding Author

ABSTRACT

SQL injection attack is an attacking technique whereby a malicious user input well-crafted data to the username or password field of a Web browser without proper validation. This type of attack is very rampant in recent times according to Open Web Application Security Provider [26], which categorized this attack as the second most dangerous attack on Web application after cross-site scripting (XSS) attack. In this paper, we applied a modified form of the C4.5 Learner’s Algorithm, a decision tree classifying algorithm to classify SQL queries as benign or malicious. Our result shows that the modified version of the algorithm was able to classify the data to a reasonably acceptable level despite the criticisms that the C4.5 algorithm is only efficient in handling small data set and that it can only handle numeric data.

Keywords: Learner’s Algorithm, SQL injection attacks, SQL query, input validation.

1. INTRODUCTION

SQL injection attack is an attacking technique whereby a malicious user input well-crafted data to the username or password field of a Web Browser without proper validation. SQL injection attack is usually included in a dynamically generated SQL statement and treated as SQL code [18]. For database dependent websites, SQL injection attacks are often exploited by attackers since they are easy to find and penetrate [2] [5], through SQL injection attacks. An SQL injection attack (SQLIA) is a type of attack on web applications that exploit the fact that input provided by web clients is directly included in the dynamically generated SQL statements [25]. SQLIA is one of the foremost threats to web applications [10] [12]. SQL injections are the second most serious web application vulnerabilities after cross-site scripting [26].

One reason for the wide spread of this form of attack is that they are easily exploited in the web by attackers. The root cause of SQLIAs is insufficient input validation. SQLIAs occur when data provided by a user is not properly validated and is included in an SQL query [11] [20] [14]. In such a vulnerable application, an SQLIA uses malformed user input that alters the SQL query issued in order to gain unauthorized access to a database and extract or modify sensitive information [21]. SQL injections have been used to extract customer and other information from e-commerce databases, or bypass security mechanisms [21]. This injection typically occurs through web form and associated scripts that do not perform appropriate input validation.

In this paper, we applied a modified form of the C4.5 decision tree learner’s algorithm to classify queries into legitimate or malicious class. The C4.5 learner’s algorithm was developed by [23]. It was an improvement to the IDE3 algorithm. The algorithm is based on Hunt’s model. We modified the algorithm to suite the purpose to which it was intended. Although the algorithm has been criticized by some authors by claiming that it cannot be used for large data set. What we did was to modify the algorithm and applied it for the SQL queries. The algorithm was able to classify these queries although there were a couple of errors resulting from misclassification but the percentage was minimal (0.001%).

2. SQL INJECTION ATTACKING TECHNIQUES

A number of techniques were proposed by [13], which attackers used to attack web sites and databases in order to gain access to people’s information. This grouping is what they called SQLIA types. These SQLIA types or techniques are most times often combined or used sequentially. The major groupings are explained below.
Tautologies: This SQLIA technique uses injection code in one or more conditional statements in such a way that whenever the query is executed, the result will always be evaluated to true. In this type of injection attack, an attacker uses an injectable field that is used in a query's WHERE condition. Transforming the condition into a tautology will make all the rows in the database table targeted by the query to be returned. However, for this injection type to work, an attacker must consider two important issues i) the injectable / vulnerable parameters ii) the coding constructs that evaluate the query results. Hence, an attacker is successful when the command either display all records returned or performs some action if at least one record is returned.

For example

Let us consider a scenario where an attacker submits " ' or 1=1--" for the login input field. In this case, the input submitted for the other field becomes irrelevant since anything after the ("--") will be regarded as comments and will be ignored (-- is a comment sign in SQL). The resulting query will therefore look like this:

```
SELECT accounts FROM users
WHERE login=" or 1=1-- AND password=" AND pin=
```

The code injected in the conditional (OR 1=1) transforms the entire WHERE clause into a tautology. Because this condition is a tautology, the query will evaluate to true for each row in the table and returns all of them [13] [16]. The following are a sample list of inputs that form tautologies:

- ' or 1=1--
- " or 1=1--
- 1=1--
- ' or 'a'='a
- " or "a"="a
- ') or ('a='a

These inputs can be given in their hex encoded equivalent also. For example

' or 1=1-- can be given as 0x27206F7220313D31

Union Queries: This form of attack is related to tautologies except that it allows access to different tables than the ones originally involved in the query. In union-query attacks, an attacker exploits a vulnerable parameter to change the data set returned for a given query. This way, an attacker tricks the application such that data will be returned from a different table returned for a given query. This way, an attacker exploits a vulnerable parameter to change the data set originally involved in the query. In union-query attacks, an attacker uses an injectable field that is used in a query's WHERE condition. Transforming the condition into a tautology will make all the rows in the database table targeted by the query to be returned. However, for this injection type to work, an attacker must consider two important issues i) the injectable / vulnerable parameters ii) the coding constructs that evaluate the query results. Hence, an attacker is successful when the command either display all records returned or performs some action if at least one record is returned. Let us consider a scenario where an attacker submits " ' or 1=1--" for the login input field. In this case, the input submitted for the other field becomes irrelevant since anything after the ("--") will be regarded as comments and will be ignored (-- is a comment sign in SQL). The resulting query will therefore look like this:

```
SELECT accounts FROM users
WHERE login=" or 1=1-- AND password=" AND pin=
```

The code injected in the conditional (OR 1=1) transforms the entire WHERE clause into a tautology. Because this condition is a tautology, the query will evaluate to true for each row in the table and returns all of them [13] [16]. The following are a sample list of inputs that form tautologies:

- ' or 1=1--
- " or 1=1--
- 1=1--
- ' or 'a'='a
- " or "a"="a
- ') or ('a='a

These inputs can be given in their hex encoded equivalent also. For example

' or 1=1-- can be given as 0x27206F7220313D31

However, if there is no login equal to " ", the first query returns null but the second query will return data from the "CreditCards" table. In this case, the database would return column “cardNo” for account 123. The database takes the results of the two queries UNION them and return a result. This way, the value for the cardNo is displayed along with the account information of the victim(s) of the attacker [11] [16].

Piggy-Backed Queries: These are malicious queries added to be executed in addition to the original query. In this form of attack, the attackers are not interested in modifying the original intended query but instead they try to add new distinct queries to the original query. The implication of this action is that the database receives multiple queries. The first or main query which is the intended query would be executed normally while subsequent queries would be executed alongside the main query. This type of attack is often very harmful especially when the attack is successful. Once the attacker succeeds, he can insert virtually any type of SQL command including stored procedures (stored procedures are routines stored in the database and run by the database engine). These can be user-defined or default procedures in addition to the main or original query and have them executed. Let us consider a situation in which an attacker input " ; drop table user--" into the password field, the application generates the query shown below:

```
SELECT accounts FROM users
WHERE login='eddy' AND pass='; drop table user--'; AND pin=123
```

On completion of the execution of the first query, the database recognizes the query delimiter (";") and execute the second injected query. This will cause the users table to be dropped from the database resulting in a loss of valuable data [13] [16].

Stored Procedures: This form of SQLIA often executes stored procedures in the database. Stored procedures are routines stored in the database engine. These could be user-defined procedure or default procedures. Most databases have standard set of stored procedures that extend the functionality of the database and allow for interaction with the operating system. What an attackers does is to determine which backend database is in use and the inject commands to execute the stored procedures provided by that database, including procedures are often written in special scripting languages. They can contain other vulnerabilities such as buffer overflows that allow attackers to run arbitrary code on the server and escalate their privileges [9] [17] [13]. In order to execute a stored procedures with SQLIA, the attacker simply injects " ; SHUTDOWN; -- " into either the username or password field. This will cause the stored procedure to generate the query shown below:

```
SELECT amounts FROM user
WHERE login='eddy' AND password=''; SHUTDOWN; -- AND pin=
```
However, the first query is executed normally while the second query which is the malicious (injected) query is executed and causes the database to shutdown. Stored procedure may allow the attacker to interact with the targets beyond the database, like the underlying Operating System, for example,

xp_cmdshell is a built-in extended stored procedure that allows the execution of arbitrary command lines.

Consider the following command:

exec master..xp_cmdshell 'net1 user'

This command will provide a list of all users on the machine. Since SQL server is normally running as either the local 'system' account, or a 'domain user' account, an attacker can do a great deal of harm.

The following are the list of extended Stored Procedure that can be used.

- **xp_cmdshell** – Used to run commands as the SQL server user on the database server, e.g., exec master xp_cmdshell 'dir' obtain a directory listing of the current working directory of the SQL Server process
- **xp_regXxx** – Used to manipulate on registry keys. The following are the listing of the xp_regXxx
  - xp_regaddmultistring
  - xp_regdeletetempkey
  - xp_regenumkeys
  - xp_regdelevale
  - xp_regread
  - xp_regremovemultistring
  - xp_regwrite

For example, we can have the following execute statement:

exec xp_regreadHKEY_LOCAL_MACHINE,‘SYSTEM\CurrentControlSet\Services\lanmanserver\parameters’, ‘nullsessionshares’

- **xp_servicecontrol** – Used to start, stop, pause and continue services e.g., exec master..xp_servicecontrol ‘start’, ‘schedule’
- **xp_availablemedia** – Reveals the available drivers on the machine.
- **xp_dirtree** – Allows directory tree to be obtained
- **xp_enumdsn** – Enumerates OBC data sources on the server
- **xp_loginconfig** – Reveals information about the security mode of the server
- **xp_makecab** – Allows the user to create a composed archive of files on the server (or any file the server can access)
- **xp_netsec_enumdomains** – Enumerates domains that the server can access
- **xp_terminate_process** – Terminates a process, given its PID
- **xp_OACreate, sp_OAMethod and sp_OAGetProperty** – These system stored procedures can do everything as ASP script can do

**Inference:** In this form of attack, the attackers often try to attack a site that is well secured. The attacker does this modifying the original query to recast it in the form of an action that is executed based on the answer to true/false question about data values in the database. This way, the attacker injects commands into the site and then observes how the function/response of the web site changes. The attacker then watches the behaviours of the site by noting when the behaviour is the same or when it behaves funny. The attacker can then infer/deduce the parameters that are vulnerable to attack. Also, the attacker would know other information in the database such as data values. The attackers does this by modifying the original query to recast it in the form of an action that is executed based on the answer to true/false questions about data values in the database.

There are two forms of inference attacks:

- The first form is identifying injectable parameters using blind injection. For example, a user can inject two different commands into the login field. The first case is “legalUser’ and 1=0” and the second is “legalUser’ and 1=1”. These two injection commands result in the following query:

  \[
  \text{SELECT accounts FROM users WHERE login= 'legalUser' and 1=0 - -' AND password= ' ' AND pin=0}
  \]

  \[
  \text{SELECT accounts FROM users WHERE login= 'legalUser' and 1=1 - -' AND password= ' ' AND pin=0}
  \]

  In the first scenario, the application is secured and the input for login is validated correctly. In this case, both injections would return login error messages, and the attacker would know that the login parameter is not vulnerable. The second scenario shows an example of insecure application and the login parameter is vulnerable to injection. The attacker submits the first injection and because it always evaluate to false, the application returns a login error message. At this point, the attacker does not know if this is because the application validated the input correctly and blocked the attack attempt or because the attack itself caused the login error. The attacker then submits the second query, which always evaluates to true. However, if there is no login error message, then the attackers know that the attack went through and that the login parameters are vulnerable to injection [3] [24] [13].
The second way inference based attack can be used to perform data extraction. This is a timing-based inference attack used to extract a table name from the database. In this attack, the following type of query is injected into the login parameters:

```
'legalUser' and ASCII(SUBSTRING((select top 1 name from sysobjects), 1,1)) > X
WAITFOR 5 - - ''
```

This produces the following query:

```
SELECT accounts FROM users WHERE login='legalUser' and ASCII(SUBSTRING ((select top 1 name from sysobjects), 1,1)) > X WAITFOR 5 - - 'AND password= ' AND pin=0
```

In this attack, the SUBSTRING function is used to extract the first character of the first table’s name. Using a binary search technique, the attacker can then ask a series of questions about this character. In this case, the attacker is asking if the ASCII value of the character is greater-than or less-than-or-equal-to the value of X. If the value is greater, the attacker knows this by observing an additional 5 seconds delay in the response of the database. The attacker can then use a binary search by varying the value of X to identify the value of the first character.

Alternate Encoding: This form of attack is used in conjunction with other attacks. Alternate encoding does not provide any unique way to attack an application. Alternate encoding is a detection technique that exploits vulnerabilities that might not otherwise be exploitable using other techniques. These evasion techniques are often necessary because a common defensive coding practice is to scan for certain known “bad characters”, such as single quotes and comment operators. Examples of alternate encoding are:

```
0; exec (0x73587574 64 5f77 6e),
SELECT accounts FROM users WHERE login="legalUser" AND pin=0; exec (char(0x73687574646f776e))
```

Use of Comments: Most SQL implementations such as PL/SQL and T-SQL use ‘- -’ to indicate the start of a comment (‘#’ is also used occasionally). By injecting comments, attackers can truncate SQL queries without much effort. Let us consider an example of comment in SQL queries:

```
SELECT * FROM users WHERE username='eddy'
AND password='password'
```

By merely supplying ‘admin’ - - as the surname, the query is truncated, thus eliminating the password clause of the WHERE condition. This is because whatever appeared after the ‘- -’ is regarded as a comment and is therefore not executed. Also, because the attacker can truncate the query, the tautology attack presented earlier can be used without the supplied value being the last part of the query. Thus attackers can create queries such as

```
SELECT * FROM users WHERE username='anything' OR 1=1 AND password='irrelevant'
```

This is guaranteed to log the attacker in as the first record in the users table, often as an administrator. Once this happens, the user may have access to the whole database if logged in as an administrator.

Illegal/logically Incorrect Queries: This form of attack is considered as pre-attack preparation by an attacker. This attack is performed by entering some inputs which generates illegal or logically incorrect queries. The error messages will reveal the names of the tables and the columns that cause error. The attacker also comes to know about the application database used in the backend server. For example an attacker can inject a query like this:

```
SELECT * FROM users
WHERE login='derived' AND password=convert(select host from host)
```

Thus the injected query generated first tries to execute the column host from host table. The host table consists of the information about the users privileges. The query will try to convert the host column data to an integer. Since this is not a legal type conversion, the database server returns an error message [6]. Table 1 summarizes some of the features or attributes of the various types SQL injection attacks.
Table 1: SQL Injection Attacking Techniques and their Attributes

<table>
<thead>
<tr>
<th>Attacking Techniques</th>
<th>Attributes</th>
</tr>
</thead>
</table>
| **Tautology**        | i) ' or 1=1- -  
  ii) " or 1=1- -  
  iii) or 1=1- -  
  iv) ' or 'a'='a  
  v) " or "a"="a  
  vi) ') or ('a'='a  
  vii) ' or 2>1--  
  viii) 'greg' LIKE '%gr%'- -  |
| **Union**            | ‘UNION SELECT 1,1 - -  |
| **Piggy-backed queries** | i) '; show tables, - -  
  ii) '; shutdown, - -  
  iii) '; drop table - -  |
| **Stored procedure** | i) xp_cmdshell  
  ii) xp_regXXX  
  iii) xp_dirtree  
  iv) xp_servicecontrol  
  v) sp_XXXX |
| **Inference**        | i) '; /*!SELECT CONCAT ('1', '2') */ (MySQL) - -  
  ii) '; /*!SELECT '1' + '2' */ (MS SQL) - -  
  iii) ASCII(SUBSTRING((select …), 1,1)) > XWAITFOR 5 –  
  iv) SELECT …covert(select host from host) |
| **Use of comment**   | i) 'abc'- -  
  (ii) 'abc' # |
| **Alternate Encoding** | 0; exec (0x73587574 64 5f77 6e), SELECT accounts FROM users WHERE login=" AND pin=0;  
  exec (char(0x73687574646f776e)) |

### 3.1 Decision Tree and the C4.5 Learning Algorithm

Decision trees are very important tools for modeling and optimization of probabilistic multistage decision-making problems [7]. They constitute a qualitative representation of a problem structure, which provides the decision-maker with an optimum chronological sequence of decisions. Decision trees are a simple, but powerful form of multiple variable analyses. They provide unique capabilities to supplement, compliment, and substitute for multiple linear regressions, neural networks, and business intelligence [20]. Decision trees are very important tools for modeling and optimization of probabilistic multistage decision-making. They constitute a qualitative representation of a problem structure, which provides the decision-maker with an optimum chronological sequence of decisions [8].

Decision tree is an effective tool for guiding a decision process as long as no changes occur in the dataset used to create the decision tree [11]. Decision trees provide unique insight into the problem of identifying malicious activities and can assist in the creation of technology-specific techniques to defend against attacks. The main advantage of decision trees over many other classification techniques is that they produce a set of rules that are transparent, easy to understand, and easily incorporated into real time techniques [19]. Decision trees are a simple, but powerful form of multiple variable analyses [15].

Decision trees are useful for classifying objects because of their hierarchical nature between the parent and children nodes. A decision tree is both a knowledge representation scheme and a method of reasoning about its knowledge [22]. They provide an elegant framework for studying the complexity of Boolean functions. The internal nodes of a decision tree are associated with tasks on the input, the branches leaving a node correspond to the outcomes of the associated task; and the leaves of the tree are labeled with output values. The main parameter in decision tree is its depth. That is, the maximum number of tests made over all inputs. Therefore, the decision tree complexity of a particular Boolean function is defined as the minimum of this parameter over all decision trees computing the function.

**C4.5 Learning Algorithm:** This algorithm was developed by [23]. The C4.5 algorithm is an improvement to the IDE3 algorithm. The algorithm is based on Hunt’s model. The C4.5 algorithm is also serially implemented just like the IDE3 algorithm. The C4.5 algorithm is used for a larger data set. C4.5 algorithm is primarily used for numeric data (both discrete and continuous data) based on some entropy. In C4.5 algorithm, the internal node is replaced with a leaf node using the process of pruning thus reducing the error rate. Like IDE3 algorithm, the data in C4.5 algorithm is sorted at every node of the tree in order to determine the best splitting attribute [4]. The algorithm uses gain ratio impurity method to evaluate the splitting attribute.
The C4.5 learning algorithm can be used to solve problems with harder domains that contain many numbers of possible values. These domains are:

- Discrete domain
- Continuous domain
- Complex domain

(i) Discrete Domain: This domain is commonly constructed by a set of nominal values. The set of nominal values has to be a finite set with values which are discrete. The term nominal means there is no ordering between the values, such as last names or colours. For example, an attribute with the values low, medium, or high.

(ii) Continuous Domain: Continuous domain is a subset of real numbers, used to measure differences between possible values. This type of domain has to be in discrete form by defining various numbers of threshold (intervals) for each possible class, for example, we can categorize people between certain age groups as children, teenagers or adults as shown below.

\[
\begin{align*}
\text{age} < 10 & \Rightarrow \text{children} \\
10 \leq \text{age} < 20 & \Rightarrow \text{teenage} \\
\text{age} \geq 20 & \Rightarrow \text{adult}
\end{align*}
\]

Literal attributes which are set of strings can have a discrete or continuous domain. In the case of continuous domain, it has to be categorized by defining various numbers of sets for each possible class. For example, we can categorize character literals as follows:

- Letter \( \{e, a, i, o, u\} \Rightarrow \text{vowel} \)
- Letter \( \{e, a, i, o, u\} \Rightarrow \text{consonant} \)

(iii) Complex Domain: This type of domain is used to implement a domain of an attribute belonging to another class of object. We processed the complex domains by calling all the complex domains by getting their attributes until every complex attribute is converted to a set of attributes with discrete or continuous domain.

Thus C4.5 learning algorithm deals with the numeric (integer and real) and continuous attributes using a discrete technique based on entropy. It is the principle of this algorithm that we employed in this paper. The modified algorithm is shown in figure 1.

3. MATERIALS AND METHOD

Materials
We used a Pentium® Dual-core 2.10GHz processor with 2GB RAM and 64-bit system architecture running Windows 7. We created sample database in MySQL server 5.0. The J2EE application bundled into Netbeans software was used. To match real world application, JSP (Java Server Page) was compiled into servlets. The Web server, database server, and clients were in same local machine. Although, the C4.5 learning algorithm deals with the numeric (integer and real) and continuous attributes using a discrete technique based on entropy and small dataset, we modified the algorithm to suit our objective in the work. The modified algorithm is shown in Figure 1. Since we are interested in the algorithm to be able to classify a query as malicious or benign, there are two possible outcomes. Thus we set one (1) for TRUE and zero (0) for false respectively.

Method
In our experiment, we used the AMNESIA Testbed [11]. Although the AMNESIA Testbed contains a very large number of queries, we decided not to use all. First we selected some of these queries randomly and we then further optimize them to ensure that we are working with a reasonable set of queries. We then calculated the percentage of reduction of these queries before applying our modified C4.5 decision tree learner’s algorithm. Table 2 shows the Percentage of reduction after query optimization.

<table>
<thead>
<tr>
<th>Subject</th>
<th>No of queries Before optimization</th>
<th>No of queries After optimization</th>
<th>% of reduction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bookstore</td>
<td>320</td>
<td>63</td>
<td>19.6%</td>
</tr>
<tr>
<td>Portals</td>
<td>467</td>
<td>77</td>
<td>16.5%</td>
</tr>
<tr>
<td>EmplDir</td>
<td>295</td>
<td>36</td>
<td>12.2%</td>
</tr>
<tr>
<td>Classifieds</td>
<td>280</td>
<td>27</td>
<td>9.6%</td>
</tr>
<tr>
<td>Events</td>
<td>315</td>
<td>23</td>
<td>7.3%</td>
</tr>
<tr>
<td>Checkers</td>
<td>436</td>
<td>43</td>
<td>9.9%</td>
</tr>
<tr>
<td>Office Talk</td>
<td>214</td>
<td>24</td>
<td>11.2%</td>
</tr>
<tr>
<td>Total</td>
<td>2327</td>
<td>293</td>
<td>Avrg = 10.9%</td>
</tr>
</tbody>
</table>

The total number of queries we used before and after optimizing them and the percentage of reduction of these queries are presented in Table 2. Before optimization, the total number of queries collected was 2327 while the number reduced to 293 after optimization indicating an average of 10.9%. These classifications were determined based on the attributes of each query that we used in the experiment.
3.1 A Modified Version of C4.5 Decision Tree Learner’s Algorithm

In this algorithm, we used the principle of the work of [23] in C4.5 learning algorithm which can solve problems in harder domains that contain a number of possible values. A modified version of C4.5 decision tree learner’s algorithm is shown in Figure 1.

Modified Algorithm: Decision Tree Learner for Malicious Attacks
Learner (s, t, d)
1. Input: s: SQL statement
t: tree specification
   attrs: list of attributes
   attr\_target: target attributes, and
   C[attr\_target]: list of target attribute categories
d: depth of branch
2. Select malicious query from categories
   malicious ← queries with malicious attributes
3. Output: A Boolean value indication whether s is malicious or no
4. select s
5. If attrs found then
   6. return leaf\_classified (malicious)
   7. If attrs = ø
   8. choose attr\_best to split s
   9. If ø criteria c \( \notin \) list of Stopping Criterion: c = true
      10. then return leaf\_majority (malicious)
      11. for c \( \notin \) C[attr\_best]
      12. Instantiate tree specification t’ excluding attribute attr\_best
      13. If s \( \neq \) ø
         14. then node\_child ← learner (s, t, d+1)
      15. set node\_child as child of node\_new at c
      16. else
      17. return leaf\_no\_Attribute (benign)
      18. endif
      19. endif
   20. endif
   21. endif

Fig. 1: A modified version of C4.5 decision tree learner’s algorithm.

Although, it has been widely argued that the C4.5 learner’s algorithm is efficient for small dataset, we decided to use this algorithm since it deals with all forms of attributes whether discrete or continuous. As a result of the modification made to the algorithm, we were able to use it for a large dataset as earlier reported in our work. We used this algorithm to train the data set (SQL statement) for the decision tree. The algorithm chooses the best attribute which differentiate the target attributes the most. When the learner chooses the best attribute attr\_best it divides the instances into subgroups so as to reflect the attribute categories of the chosen node at line 8.

Then it creates a separate tree branch for each of the chosen attributes. For each subgroup, the learner calls itself if there is no termination condition satisfied as shown in line 8, and lines 9 and 13. This is used as a stopping criterion whenever the target attribute is found to be true and thus classify the query as malicious otherwise the query is classified as legitimate. The algorithm can terminate by returning a node classifying legitimate target category if:

- All instances turn false
- There is no attribute left to classify as in line 7
- There exists a stopping criterion equal to true given the current statistics as in lines 9 and 13.
3.2 Design Model

The design tool we employed for this research work is unified modeling language (UML). The UML contains the SqlClassifierDTree which consist of the parser and the decision processor. The UML diagram also contains decision processor which classifies the query into expected value of valid or invalid query. It also contains the rule classifier that helps to evaluate the query and initialize the value to expected value through the expression rule.

The request processor processes the query using the attributes of valid queries by creating children nodes as the query passes through the request processor to ensure the validity or otherwise of the query. This is done by ensuring that the processor gets the required attributes by using the getRule and getQueryClassification methods. This process is iterated several times until the query is completely authenticated. Figure 2 shows a UML diagram showing the various classes and method for implementing the algorithm.

Fig. 2: A UML diagram showing the various classes and method for implementing the algorithm
4.0 RESULT
Using our modified C4.5 decision tree learner’s algorithm, we tested the optimized queries in order to know the number of legitimate (benign) and malicious queries. Table 3 shows the result of our experiment (number of legitimate and malicious queries and their percentages). In Table 3, the total number of optimized queries tested is 293 out of which 244 are legitimate signifying about 83.27% of the total queries used; while 49 queries are malicious signifying about 16.72 of the total queries used.

Table 3: Number of Legitimate and Malicious Queries and their Percentages

<table>
<thead>
<tr>
<th>Optimized Queries</th>
<th>Legitimate Queries</th>
<th>Malicious Queries</th>
<th>% of Legitimate Queries</th>
<th>% of Malicious Queries</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bookstore</td>
<td>63</td>
<td>54</td>
<td>9</td>
<td>85.7</td>
</tr>
<tr>
<td>Portals</td>
<td>77</td>
<td>62</td>
<td>15</td>
<td>80.5</td>
</tr>
<tr>
<td>EmplDir</td>
<td>36</td>
<td>31</td>
<td>5</td>
<td>86.1</td>
</tr>
<tr>
<td>Classifieds</td>
<td>27</td>
<td>20</td>
<td>7</td>
<td>74.1</td>
</tr>
<tr>
<td>Events</td>
<td>23</td>
<td>19</td>
<td>4</td>
<td>82.6</td>
</tr>
<tr>
<td>Checkers</td>
<td>43</td>
<td>37</td>
<td>6</td>
<td>86.1</td>
</tr>
<tr>
<td>Office Talk</td>
<td>24</td>
<td>21</td>
<td>3</td>
<td>87.5</td>
</tr>
<tr>
<td>Total</td>
<td>293</td>
<td>244</td>
<td>49</td>
<td>% Avg = 83.2</td>
</tr>
</tbody>
</table>

Figure 3 shows a graph of legitimate and malicious queries before and after optimization. In Figure 3, it can be seen that after optimization, the number of legitimate queries is 244 representing an average of 83.2% of the total queries considered while the number of malicious queries is 49 representing an average of 16.8% of the total queries considered. This is quite a high figure and this is the reason why SQL attacks are considered to be most prevalent attacks in Web application according to [27] report.

We further calculated the actual number of queries detected that are in each category of malicious queries based on the total number of optimized queries used for our experiment. Our result shows that of the query types we used based on the classification from the work of [12], “A Classification of SQL Injection Attacks and Countermeasures”, which are: tautology, logically incorrect, piggy-backed, stored procedure, inference, and alternate encoding, we got the result as shown in Table 4 (Actual number of malicious queries detected based on their classes).
Table 4: Actual Number of Malicious Queries Detected Based on their Classes

<table>
<thead>
<tr>
<th>Class</th>
<th>Tautology</th>
<th>Logically Incorrect</th>
<th>Piggy-Backed</th>
<th>Stored Procedure</th>
<th>Inference</th>
<th>Alternate Encoding</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bookstore</td>
<td>5</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Portals</td>
<td>8</td>
<td>5</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>EmplDir</td>
<td>3</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Classifieds</td>
<td>5</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Events</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Checkers</td>
<td>4</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Office Talk</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>29</strong></td>
<td><strong>8</strong></td>
<td><strong>2</strong></td>
<td><strong>4</strong></td>
<td><strong>3</strong></td>
<td><strong>2</strong></td>
</tr>
</tbody>
</table>

Figure 4 shows a graph of the occurrence of the different types of SQLIAs. In Figure 4, we can see from the graph that the tautology queries are the most common form of SQL injection attacks as they constitute about 59.2% of the total number of queries used for this experiment before and after optimization. This indicates that most malicious SQL queries are the most common SQL injection attacks common in Web applications.

![Graph of SQLIAs](image)

**Fig. 4: A graph of the occurrence of the different types of SQLIAs**

4.1 Discussion
In Table 3, we can see at a glance that the most common SQL injection attack query is tautology which constitutes about 59.2% of the total malicious queries detected. This is closely followed by logically incorrect queries – which most times often emanates from syntax errors. This means that the intention of the users who supplied such queries may not necessarily be to constitute any malicious activity. But because they do not conform to the rules of the SQL statement that constitute legitimate queries, they are classified as malicious. Although, there were a couple of errors in our experiment which caused misclassification of queries most especially false positives, our algorithm has drastically reduced misclassification of queries to about 0.001%.

However, to solve the problem of false positives where legitimate queries may be classified as malicious queries thus preventing genuine users from having access to a web site; and false negatives where malicious queries are classified as legitimate queries, we ensured that our algorithm has only two values to work with, true (1) or false (0). This is to ensure that it has few variables to work with and thus reducing the time complexity of the algorithm.

5. CONCLUSION
SQL injection attack is an attacking technique whereby a malicious user input well-crafted data to the username or password field of a Web browser without proper validation. SQL injection attack is usually included in a dynamically generated SQL statement and treated as SQL code. SQL injection attacks are one of the most foremost threats to Web applications. It is an attacking technique which is used to pass SQL query through a Web application directly to the
database by taking advantage of insecure code’s non-validated input values. They pose a serious security threat to Web applications. They allow attackers to obtain unrestricted access to the databases underlying the applications and to the potentially sensitive information these databases contains.

In this paper, we applied a modified form of the C4.5 decision tree learner’s algorithm to classify queries into legitimate or malicious class. The C4.5 learner’s algorithm was developed by [23]. It was an improvement to the IDE3 algorithm. The algorithm is based on Hunt’s model. We modified the algorithm to suite our purpose. Although the algorithm has been criticized by some authors arguing that, it cannot be used for large data set. What we simply did was to modify the algorithm and applied it for the SQL queries. The algorithm was able to classify these queries although there were a couple of errors resulting from misclassification but the percentage was minimal (0.001%).

REFERENCES


Authors' Bio

Ogheneovo Edward Erhieyovwe received his B.Sc. in Computer Science/Mathematics from the University of Port Harcourt in 1997 and M. Tech. in Information Technology from the Federal University of Technology, Akure, in 2004 and his Ph.D. in Computer Science at the University of Port Harcourt, Port Harcourt, Nigeria, in 2013 where he currently lectures. Dr. Ogheneovo is a Certified Oracle Database Professional. His research interests are in the area of computer security, programming, software dependability, and machine learning. He has published several articles in Learned Journals both in Nigeria and Internationally. He is a member of the Nigerian Computer Society (NCS) and Professional Registration Council of Nigeria (CPN).

Prince Oghenekar Asagba had his B.Sc. degree in Computer Science at the University of Nigeria, Nsukka, in 1991, M.Sc. degree in Computer Science at the University of Benin in April, 1998, and a Ph.D degree in Computer Science at the University of Port Harcourt in March, 2009. He is a Senior Lecturer and a visiting lecturer to several Universities in Nigeria since 2010. His research interest include: Network Security, Information Security, Network Analysis, Modeling, Database Management Systems, Object-oriented Design, and Programming He has published several articles in Learned Journals both in Nigeria and Internationally. He has authored and co authored several books in Computer Science. He is a member of Nigeria Computer Society (NCS) and Computer Professional Registration Council of Nigeria (CPN).