Detecting Intrusions in Web Applications Using CLTT and SQL Parser

Pukale S.J., Nandgaonkar S.S.

Abstract—Web application allows a set of website users to submit and retrieve data dynamically to and from a database server over the Internet using their web browser. Generally, Databases system is central to today's websites. And it store data useful for websites to deliver desired content to website visitors as well as pass information to customers, suppliers, employees. Data like User credentials, financial, online Trading payment information, organization statistics, market review may all be resident within a database. But all this websites are all highly susceptible to various types of web attacks. The drawback of existing models in detecting intrusions in multitier web applications has motivated this paper to present a novel approach against web attacks. In this paper, we propose a novel concept of character level taint tracking and SQL Lexer Token analysis for detecting web attacks.

Index Terms—Character Level taints tracking, SQL Lexer, web attacks, Intrusion Detection.

I. INTRODUCTION

Web applications are simply computer programs allowing website end user to submit and retrieve data to and from a database over the Internet using any preferred web browser. In order to tackle with large amount of data web application are moved towards client server architecture.

![Fig.1 Three Tier Architecture](image)

At start the end user requests a web page either a static or dynamic through web browser. Request is passed to web server,web server handle this request by invoking requested web application. Finally, the web application passes these requests to database. After verifying the privileges the web application server issues the Structured Query Language (SQL) requests to the database server. Where the database server fulfill this requests by allowing some standard database operation like update, storage, deletion on data depending upon parameters embedded within SQL String. Generally, web applications communicate with database system by means of Dynamic queries generated by the web applications Programming Logic. A query string contains the SQL keyword, commands and HTTP request parameters. If the received query string infected it will cause data leakage thus security of web application is prime concern. Generally web based attack are network born which arise because of the fields available for user to input. Which allow SQL query string statements to pass through and query the desired database server directly. Firewalls and various intrusion detection mechanisms provide little bit or no defense against full-scale web based attacks. Thus there is a necessity that an effective and easy to deploy tool for preventing existing vulnerable web applications against web attack.

II. LITERATURE SURVEY

Snort [1]: Martin Roesch proposed a new open source IDS which makes use of handcrafted rules to detect known attacks. Christopher Kruegel and Giovanni Vigna introduced [2] first anomaly based detection system especially designed for the detection of web-based attacks. In which attack is detection is mainly carried out by simple pattern-matching techniques to the contents of HTTP requests. Giovanni Vigna, William Robertson, Vishal Kher, Richard A. Kemmerer [3] proposed an integrated concrete approach which is based on stateful analysis of multiple event streams. On the basis of State Transition Analysis method it analyzes a sequence of activities that an attacker to find braches in the system and such sequence of activities is called signature actions. Where signature actions mean set of minimum possible actions needed to perform successful attack. Marco Cova, Davide Balzarotti, Viktoria Felmetser, and Giovanni Vigna [4] introduced an novel approach which is based upon detailed analysis and characterization of the internal state of a web application, by means of a variety of large no number of anomaly models. Giovanni Vigna, Fredrik Valeur, Davide Balzarotti, William Robertson, Christopher Kruegel,Engin Kirda [5] proposed detection system purely anomaly based and which is composed of web-based anomaly detector, a reverse HTTP proxy, and a SQL query anomaly detector. Yi Xie and Shun-Zheng Yu [6] proposed an approach which is entirely based upon concept of hidden semi Markov model used to briefly describe and characterize browsing behaviours of web user in order to detect web application associated DDOS attack. Andreas Kind, Marc Ph. Stoecklin, and Xenofontas Dimitropoulos [7] described an approach in which different traffic feature were simulated with the help of histogram. By matching histogram patterns deviations from the created models is detected.

III. SYSTEM ARCHITECTURE

Normally in conventional web server architecture network traffic from both normal and malicious user is intermixed at web server level thus it is not feasible to detect as well as separate out attacked session. Thus we follow container based session separated web server architecture with which it is feasible to segregate traffic session wise and information flow is being separated sessionwise. Such architecture also
enables us to map all web server requests with its associated database queries. Architecture enables us to capture the web requests along with resulting Dynamic SQL queries and hence possible to evaluate them in the individual session container [9]. Filters are placed at both sides one at web server layer and another at database layer which detect abnormality by capturing and analyzing the traffic at both the end.

![Fig.2 Container Based Session Separated Architecture](image)

**IV. BACKGROUND**

A query string is a set of characters passed as input to a browser where browser pass it to a Database program in order to recover specific information from a database. In terms of Internet world a query string is also called an HTTP string. It is simply set of characters which is input in the address bar of a dynamic Web site with which a user makes a request for data according to certain criteria. Web application programming logic makes use of parameter passed within HTTP query string and generates SQL query statements further passed to database server. We formulate a simple mathematical model as follows. SQL Statement=Web Application Program Logic (HTTP Request Query string Parameter) Web application programming logic is nothing but a static function which accepts HTTP Query string parameter and generates SQL Query statements. Consider static web application code that generate following query as follows

```sql
query = "SELECT * FROM users WHERE user_id = "user_ID + ""
```

In above logic user_ID is derived from HTTP web request parameter. If user enters 1 then resulting dynamic query becomes

```sql
query = SELECT * FROM users WHERE user_id = '1'
```

And modified query becomes

```sql
SELECT * FROM users WHERE user_id = '1' DROP TABLE users; --
```

and such query will cause little bit harm to database by deleting some part of database.

Thus approach of Character-level taint tracking [6] will help us to detect and block Web injection attacks. We employ mechanism in such way that before executing any SQL query statements it is passed to Tokenization. Where the SQL Lexer tokens are generated and then simply perform intensive check on Token generated. And before submitting query statements to Database we ensure that none other than literals either string or numeric literals used in query statements are tainted.

In the above case, if the user ID is 1, then the resulting dynamic SQL query will be

```sql
SELECT * FROM users WHERE user_id = '1'
```

Our proposed architecture will allow to pass this query since set of tainted characters occur only inside a one type of literal i.e. string literal. In case of malicious user architecture will simply restrict second query because it recognize that taint information is not limited to string or numeric literals but it is combination of Several SQL keywords, delimiter, Identifiers or special operators which are partially or fully tainted. Thus there is a necessity to build an infrastructure which will check every HTTP request parameter and associated invoked SQL query statements and finds an effective way to block any SQL query that indicates a web attack.

**V. IMPLEMENTATION**

In order to do an effective investigation of HTTP web request and dynamically generated SQL query string there is need to intercept and capture web traffic and Database traffic. In order to do an effective investigation of HTTP web request and dynamically generated SQL query string there is need to intercept and capture web traffic and Database traffic. This is achieved by putting a filter at both the side and generated traffic is simply logged. Proposed architecture is consisting of following main modules as shown in diagram below.

![Fig.3 Implementation](image)
Infrastructure Consist of

A. HTTP Web Request Capture

It is implemented as a Simple HTTP Servlet Filter that capture HTTP web request along with form field data. For each captured web request a new instance of “mark un-trusted” is called and further which is used to keep track of un-trusted parameter.

B. Mark Un- Trusted Data

Task assigned to this module is simply track, check and trace the source of un-trusted character within captured HTTP request. This is achieved by using java’s string related classes like String Buffer, String Concatenation, and String Tokenizer. For that purpose we need to modify java string related classes up to certain extent and make them enable to store un-trusted status per character.

C. Track Un-trusted Data Propagation

One of challenge in determining un-trusted data is how un-trustworthiness is propagated from one string to another string [6]. In order to deal with this challenge we adopt basic policy that along with the use of custom Java’s string related classes. We use a special dynamic data structure separate for individual string like array which keep track of whether individual character generated from known source or not.

D. SQL Query string Capture

One of the simplest and quickest ways to log SQL statements is through tools like P6spy. This is open source JDBC proxy tool awesome for its shear simplicity and ease of use. P6Spy acts as JDBC proxy framework that enables database traffic to be intercepted and manipulated without any code changes to existing web application source code. Its Distribution consists of P6Log, an application which logs all JDBC transactions for any Java based application.

E. Tokenization

Logged and intercepted SQL statements are passed to SQL Lexer in order perform Lexical Analysis. Output of Lexer is set of distinct Tokens along with type of Tokens. We are going to perform lexical analysis in order to recognize tokens in SQL statements which are not exact string or numeric literal which further enables us to check whether any character appearing in a Token is un-trusted or not [6].

The expression:

\[ x = \frac{3 + 2}{y} \]

When lexed, we get a series of tokens as follows:

```
My @tokens = (              
[OP => 'x'], [OP => '='], [OP => '('],    
[INT => '3'], [VAR => '+'], [INT => '2'],   
[ OP => ')'], [OP => '/'], [VAR => 'y'], );
```

With a proper grammar we were able to read this series of tokens its type and take actions based upon their values.

F. Detection Module

Having a ready made set of Un-trusted parameter along with SQL Token Detection module perform runtime analysis of web request and HTTP Parameter. It makes use of following Logic for detection of an attack.

Algorithm:

- Input: A set of parameter strings r in an captured HTTP request
- Input: An Intercepted SQL Statement string S.
- Output: A Boolean value indicating whether S is malicious or not.

Steps:
1. \( \Delta \leftarrow \) Set of LexerToken in S.
2. for Every LexerToken t in \( \Delta \)
3. Do
4. If Type of (t) ≠ string or numeric literal And is Tainted (t)
5. Then
6. Return True;
7. End If
8. End For.

The intrusion detection module investigate for every web request parameter and check for each Un-trusted character which is a part of token of SQL statements S. Token which is not of type either string or numeric literal and Token is un-trusted then resulting Dynamic SQL Query string is Infected. To ensure that our infrastructure is effective in detecting intrusions we use standard attack dataset with separate list of attacked and legitimate web requests consisting of five most vulnerable web application’s named as Amnesia Test-suite.

VI. RESULTS

A. Detection Results

To test our infrastructure we used the Amnesia Test-suite consisting five Java-based most vulnerable web applications. We strongly believe that this is to be an appropriate web application for testing infrastructure as it allows us to fire a wide a range of request of different types. From Test-suite we
automatically submit web requests as the attack list and benign list through Perl based testing script. Detection logic crafted in Web application will trigger an attack exception after successful detection of a web attack. Same exception count is maintained by running script and attack counter is incremented. Detection results are logged on command prompt along with time consumed in between submission of request and detection which further enables us to determine computing overhead.

### TABLE I Detection Results

<table>
<thead>
<tr>
<th>Type</th>
<th>No of request</th>
<th>Detected/Allowed Accuracy In %</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Bookstore</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Attack</td>
<td>3063</td>
<td>3042</td>
</tr>
<tr>
<td>Legitimate</td>
<td>608</td>
<td>604</td>
</tr>
<tr>
<td><strong>employee Directory</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Attack</td>
<td>3497</td>
<td>3461</td>
</tr>
<tr>
<td>Legitimate</td>
<td>660</td>
<td>660</td>
</tr>
<tr>
<td><strong>Portal</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Attack</td>
<td>2968</td>
<td>2961</td>
</tr>
<tr>
<td>Legitimate</td>
<td>1080</td>
<td>1080</td>
</tr>
<tr>
<td><strong>Classifieds</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Attack</td>
<td>3346</td>
<td>3340</td>
</tr>
<tr>
<td>Legitimate</td>
<td>576</td>
<td>572</td>
</tr>
<tr>
<td><strong>events</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Attack</td>
<td>3002</td>
<td>2980</td>
</tr>
<tr>
<td>Legitimate</td>
<td>900</td>
<td>900</td>
</tr>
</tbody>
</table>

### TABLE II Performance Overhead

<table>
<thead>
<tr>
<th>Web Apps</th>
<th>Type</th>
<th>No of request</th>
<th>Time In seconds</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Bookstore</strong></td>
<td>Attack</td>
<td>3063</td>
<td>295</td>
</tr>
<tr>
<td></td>
<td>Legitimate</td>
<td>608</td>
<td>59</td>
</tr>
<tr>
<td><strong>employee Directory</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Attack</td>
<td>3497</td>
<td>350</td>
<td></td>
</tr>
<tr>
<td>Legitimate</td>
<td>660</td>
<td>63</td>
<td></td>
</tr>
<tr>
<td><strong>Portal</strong></td>
<td>Attack</td>
<td>2968</td>
<td>179</td>
</tr>
<tr>
<td></td>
<td>Legitimate</td>
<td>1080</td>
<td>73</td>
</tr>
<tr>
<td><strong>Classifieds</strong></td>
<td>Attack</td>
<td>3346</td>
<td>305</td>
</tr>
<tr>
<td></td>
<td>Legitimate</td>
<td>576</td>
<td>58</td>
</tr>
<tr>
<td><strong>events</strong></td>
<td>Attack</td>
<td>3002</td>
<td>298</td>
</tr>
<tr>
<td></td>
<td>Legitimate</td>
<td>900</td>
<td>69</td>
</tr>
</tbody>
</table>

### B. Performance Overhead

Our web application server as well as Linux Based Virtual machine as client is running on single physical machine having Intel Core i3 at 2.40GHz processor and 4 GB RAM. We measured the time required in between submission of request. And detection of request in terms of seconds and results are as follows.

### VII. CONCLUSION

There are few techniques which are meant for intrusion detection against multilayer web applications. Some of the technique use single IDS to detect and prevent web server from malicious request while some approach use combined approach to detect intrusions at both web and database level. In comparison with all surveyed approach this approach is having some additional detection capability. As it consider traffic at both level and perform combined analysis of web request HTTP parameter and SQL query string. Approach helps for runtime detection against variety of web based attack as well as from evaluation it is clear that it is very effective and efficient, stand alone tool providing compatibility with variety of database and easy to integrate with existing web application

### REFERENCES


AUTHOR’S PROFILE

Mr.S.J.Pukale pursuing his post graduation at Department of Computer Engineering, Vidya Pratishthan’s College of Engineering Baramati Pune, Maharashtra, India. He has completed his graduation from SSJCOE (Mumbai University) with first class.

Prof.S.S.Nandgaonkar working as Assistant Professor in Computer Engineering Department at Vidya Pratishthan’s College of Engineering Baramati, Pune Maharashtra, India. She has completed her graduation from WIT, Solapur (Shivaji University) with first class and distinction and M.Tech from COEP, Pune (Pune University) with first class and distinction.