7th International Conference on Research Developments in Applied Science, Engineering & Management
Mahratta Chamber of Commerce, Industries and Agriculture, Tilak Road, Pune (India)
Conference World
15th -16th June 2019 www.conferenceworld.in ISBN : 978-93-87793-90-3

Security Framework for Detecting the SQL Injection and Cross Site Scripting Vulnerabilities

[1]Vamsi Mohan V, [2]Dr. Sandeep Malik

[12]Department of Computer Science,
[12]School of Engineering and Technology, Raffles University, Neemrana, India

ABSTRACT - *From the last two decades, SQL Injection Attacks (SQLIA) and Cross-site Scripting Attacks (XSSA) became prominent of taint-based vulnerabilities which has caused many security breaches in the digital industry. SQL Injection (SQLI) executes malicious code in the form of SQL statement at database end whereas in the case of Cross-site scripting, the attacker targets to execute the malicious code at the web browser (client) side. In this paper, we are trying to propose a security framework to detect SQLI and XSS attacks. Regression Neural Network (RNN) introduced to identify and control the dynamic systems.*

Index Terms— Cross Site Scripting (XSS), SQL Injections (SQLI), Static Analysis, Regression Neural Network (RNN), Dynamic Analysis.

I. INTRODUCTION

In the last couple of decades, attacks against the Web applications and web sites have increased attention from security professionals due to frequent attacks. No matter how much strong firewalls installed and result sets configured. It is because of not following secured coding practices, attackers are able to trespass into your systems. The two primary techniques are SQL Injection and Cross Site Scripting attacks. These are attacks directly targets to the central servers.

Administrators can control the server-side code. However, the client-side code runs at the client side in their machines. According to K.Z.Snow, and et.al., (2016) compared with C# and Java commonly used by the server, JavaScript code often regards string data as executable code by eval or other API. Also, most of the modern frameworks, JavaScript use third party script code, which is hidden and difficult to control.

Hydra et. al. (2014), analysed XSS attacks and found 0.9% DOM-XSS only till the year 2013. Vogt et. al., (2007) proposed a static analysis and dynamic information flow tracing method to reduce the risk of XSS. In their proposed method, they focused on sensitive information like user cookies, potential information leakage, and not on tracking unreliable data. Ra is a plug in for Firefox browser for detecting DOM-XSS, which is useful for detecting black box fuzzing. SpiderMonkey JavaScript is modified and introduced a tool called Dominator for detecting DOM-XSS based vulnerability tracking.

II. IMPACT OF SQL INJECTION VULNERABILITY

SQL Injection (SQLI) is a most prevalent injection attack that executes malicious SQL code at database level to exploit the confidential information. Attackers use SQL injection technique to bypass the application security authentication. Once, attackers get the control on the database they can perform any operations to steal or destroy the sensitive data. SQL operations like add, edit, delete or drop tables are possible, once attackers break the system. The OWASP (Open Web Application Security Project) top 10 security risks for the year 2019, claims as "Injection" attacks are the number one threat to web application security.

Before targeting the SQL Injection attacks, the attackers first identify the vulnerable user inputs within the web applications and injects in the form of SQL query to the database. These malicious content or input is known as malicious payload and plays a key role in the injection attack. Sometimes, attackers can perform the SQL commands toattack andacquire the sensitive information like user credentials, credit card information and impersonate the user logins. If the attackers impersonate the Admin users, the impact of damage is more by running the operating system commands, which is more dangerous and cause serious consequences.

Example for SQL Injection

The below example depicts, how an attacker can grab the user details without proper authentication. It is a sample example of authenticating user credentials. The example database has a table named employee with the following columns: username and password.

```
# Define POST variables

username = request.POST['username']
password = request.POST['password']

# SQL query vulnerable to SQLi
query = "SELECT * FROM employee WHERE username='" + username + "' AND
password='" + passwordd + "'"

# Execute the SQL statement
database.execute(query)
```

In the above example, the input fields are likely to be vulnerable. By using the SQL command in the query and alter the actual statement. Attackers can use a single quote after the input and set the password field to password' OR 1 = 1, which is always true. As the result, the SQL query runs on the database is: SELECT * FROM employee WHERE username='username' AND password='password' OR 1=1'. Due to OR 1=1 statement, which is always true, it returns all the records of the employee table.

Some new special characters for considerations:

* (asterisk) is a symbol for the SQL query to return all records for the selected database table

= (equals) is a notation for the SQL database to only return values that match the searched string

' (single quote mark) is used to inform the SQL database where the search string starts or ends

Attackers can also comment out the partial part of the SQL statement to gain control on the execution.

```
-- All database
' OR '1'='1' --
' OR '1'='1' /*

-- Access (using null characters)
' OR '1'='1' %00
' OR '1'='1' %16
```

As discussed in the previous sections, Cross-site Scripting (XSS) attack works at the client or browser side. The Attacker targets to execute the malicious scripts in the web browser of the victim by including malicious code in a legitimate web page. When the victim clicks the URL or visits the page, it imports the malicious script to the victim's browser. Sometimes, web applications uses unsensitised user input which may also causes vulnerability. XSS attacks are primarily through the scripting languages like JavaScript, VBScript, Flash, ActiveX and even CSS for web pages.

A worst attack through Cross-site scripting attack is in the extreme cases, it may deface the web application not only targeting the visitors of the web application. In some cases, these vulnerable scripts change the content of the web application or redirects the browser to another unauthorized web site, which consists of malicious code.

Cross-site Scripting (XSS) vulnerabilities are much less dangerous than the SQLI attacks. Cross-site Scripting (XSS) vulnerabilities are much less dangerous than the SQLI attacks. Frontend scripts can be dangerous if misused or neglect as part of malicious content even though client-side scripting languages have very limited accessibility to the user's operating systems and the user files.

Malicious front-end scripts can access to the web objects including user cookies, which often stores session tokens. Through these cookies, attackers can impersonate the user and enter into the system to gain the sensitive information from the system. Some of the modern browsers use HTML5 APIs, which is actively to access user geo location, webcam, microphone and some sensitive files in the system. In accessing the user information, social engineering plays a major role.

Example of Cross-site Scripting (XSS)

The below example explains how the server-side pseudo code used to display the recent comment entered by the users on the web page

```
print "<html>"
print "<h1>Recent Comment by Users</h1>"
print database.latestComment
print "</html>"
```

The mentioned script takes the recent comment from the database and includes in the html page and assumes the comments are plain text than the html tags, which is vulnerable to Cross-site script attacks.

```
<script>PerformAttack();</script>

<html>
<script> PerformAttack();</script>
</html>
```

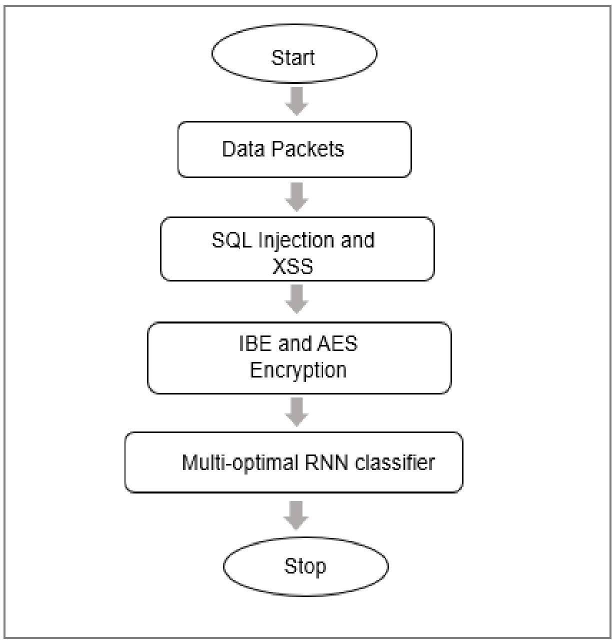
When the web page loads with the vulnerable code in the victim's browser, the attacker start executing the malicious script in most cases, the browser goes out of the victim's control. It is easy to test whether the web application is vulnerable to XSS and other vulnerabilities by running several licensed and open source scanners.

IV. SOLUTION PROPOSED FOR THE SQLI AND XSS DETECTION

This solution simulates the process to detect the SQL injection (SQLI) and Cross-site scripting (XSS) vulnerabilities in the web applications. In this proposal, we have selected the Advanced Encryption Standard (AES), which is faster than triple DES and performs iterative process. It uses 'substitution–permutation network'. It combines a series of linked operations, for which involve substitutions (replacing inputs by specific outputs) and others involve permutations (shuffling bits around).

We used IBE encryption (ID-based encryption, or identity-based encryption), a primitive of ID-based cryptography technique which is a public-key encryption where the public key of a user is unique information about the user identity (e.g. common details like user's email address or date of birth).

The sender, who accesses to the public parameters can encrypt a message using receiver's identity as a key. The decryption key obtained by the receiver from a central authority is a trusted system.



In the process of detection SQL injections and Cross Site Scripting, network data packets are collected, which can be attacked by SQL injections and cross site scripting. By using the multi-optimal RNN (Regression Neural Network), we encrypted the text using IBE (ID-based encryption, or identity-based encryption) and AES techniques.

V. PROPOSED APPROACH OF THE FRAMEWORK FOR DETECTING SQLIA AND XSS

1. Pseudo Code for AES

```
1. Initialize state=M
2. AddRound Key(state, & p[0])
3. for i=1 step 1 to 9
4. SubBytes (state)
      a. ShiftRows (state)
      b. MixColumns (state)
      c. AddRound Key (state, &p[i*4])
5. endfor
6. SubBytes(state)
7. ShiftRows(state)
8. AddRound Key(state, &p[40])
```

2. Pseudo Code for IBE

```
1. Input: Message m, receiver's public key QB
2. Output: U, C,tag
3. Set random u Zp
4. Compute U= u.G
5. Compute S( xs, ys) = u.QB
6. Generate (kENC, kMAC) =KDF(xs)
7. Encrypt C= ENC(m, kENC)
8. Generate tag = HMAC(C, kMAC).
```

3. Multi-objective optimization-based Regression Neural Network

RNN gives quick solution to approximation, regression, classification and fitting issues. RNN can also be used in system identification of dynamic systems and control of dynamic systems. In our proposal, we are integrating multi objective optimization, involves integration of objective formulation from dragon fly optimization (DA) and Genetic algorithm (GA). Integrating of optimization algorithm, crossover and mutation is used in this process.

Preliminary steps involved here are:

1. Collect Data
2. Separate into Training and Test Sets
3. Define a Network Structure
4. Select a Learning Algorithm
5. Set Parameters, Values, Initialize Weights
    a. Weight selection/ initialization carried out using Multiobjective optimization
    b. DA with genetic operators are employed here
6. Transform Data to Networks Inputs
7. Start Training, and Determine and Revise Weights
8. Stop and Test
9. Implementation: Use the Network with New Cases

4. Pseudo-codes of the DA-GA algorithm:

```
1.  Initialize the dragonflies population Xi (i = 1, 2, ..., n)
2.  Initialize step vectors ΔXi (i = 1, 2, ..., n)Xi (i = 1, 2, ..., n)
3.  while the end condition is not satisfied
4.  Calculate the objective values of all dragonflies
5.  Update the food source and enemy
6.  Update w, s, a, c, f, and e
7.  Calculate S, A, C, F, and E using above Equations
8.  Update neighboring radius
9.  if a dragonfly has at least one neighboring dragonfly
10. Update by Two point Crossover,
11. Calculate Mutation rate
12. else
13. Update the neighbor radius as new source
14. end if
15. Check and correct the chromosome based on theboundaries of variables
16. end while
17.
```

VI. CONCLUSION AND FUTURE WORK

In this paper, we tried to propose a framework for SQLIA and XSS detection considering AES and IBE encryptions. It can be further optimized. During the process of vulnerability detection, the proposed framework analyses web pages which may cause Cross-site scripting (XSS) and get the vulnerability logs. Using these logs, our method can generate automatically attack vectors, which is most significant for the project stakeholders to detect and clean the malicious or vulnerability code.

REFERENCES

[1] Brannigan, S., Smyth, N., Oder, T., Valencia, F., O'Sullivan, E., G¨uneysu, T., Regazzoni, F.: An investigation of sources of randomness within discrete Gaussian sampling. Cryptology ePrint Archive, Report 2017/298 (2017), http://eprint.iacr.org/2017/298

[2] C. Criscione, firing-range, Google, https://github.com/google/firingrange (2016).

[3] Google, Ra.2, Google, https://code.google.com/archive/p/ra2-dom-xssscanner/issues (2012).

[4] G. Wassermann and Z. Su, "Static detection of cross-site Scripting vulnerabilities," In Proceeding of the 30th International Conference on Software Engineering, (2008)May.

[5] I. Hydara, A. B. M. Sultan, H. Zulzalil, N. Admodisastro, Current state of research on cross-site scripting (xss) c a systematic literature review, Information & Software Technology 58 (2014) 170–186.

[6] I. Parameshwaran, E. Budianto, S. Shinde, H. Dang, A. Sadhu, P. Saxena, Dexterjs: robust testing platform for dom-based xss vulnerabilities, in: Joint Meeting, 2015, pp. 946–949.

[7] I. Parameshwaran, E. Budianto, S. Shinde, H. Dang, A. Sadhu, P. Saxena, Auto-patching dom-based xss at scale, in: Joint Meeting, 2015, pp. 272–283.

[8] K. Z. Snow, R. Rogowski, J.Werner, H. Koo, F. Monrose, M. Polychronakis, Return to the zombie gadgets: Undermining destructive code reads via code inference attacks, in: Security and Privacy (SP), 2016 IEEE Symposium on, IEEE, 2016, pp. 954–968.

[9] P. Vogt, F. Nentwich, N. Jovanovic, E. Kirda, C. Krgel, G. Vigna, Cross site scripting prevention with dynamic data tainting and static analysis., in: Network and Distributed System Security Symposium, NDSS 2007, San Diego, California, Usa, February - March, 2007.

[10] P. Saxena, S. Hanna, P. Poosankam, D. Song, Flax: Systematic discovery of client-side validation vulnerabilities in rich web applications., in: Network and Distributed System Security Symposium, NDSS 2010, San Diego, California, Usa, February - March, 2010.

[11] R. Gomez, DOMXSS Scanner, https://github.com/yaph/domxssscanner (2016).

[12] S. Lekies, B. Stock, M. Johns, 25 million flows later - large-scale detection of dom-based xss, in: ACM SIGSAC Conference on Computer & Communications Security, 2013, pp. 1193–1204.