INTERNATIONAL JOURNAL OF INNOVATIVE RESEARCH & DEVELOPMENT

ISSN 2278 – 0211 (Online)

SQL Injection: Attacking & Prevention Techniques

Raghavendra Babu Kalaati
SRM University, India
Balika J. Chelliah
SRM University, India
Dr. J. Jagadeesan
SRM University, India

Abstract :
This article speaks about the improvement in the development process of the application to avoid the SQLIA attack at the preliminary level, especially at the source code itself. We discuss about the possible methods to attack an SQLIA providing the pseudo code for better understanding. A prototype of the solution ie consolidating the validation into a single custom component termed as Injection Box control (IBC) that takes care of all the possible preventive measures for controlling the SQLIA at the source code level itself.

Key words: *SQLIA, IBC, Net Sparker, MSSQL, Vulnerable*

1. Introduction

Hacker injects SQL Commands into database through web portals and tries to gain access to the database for malicious purpose (i.e. steal, view or wipeout confidential information present in the database). In short the consequence of SQL Injection attack (SQLIA) is Loss of Confidential, authentication and Integrity (alteration of sensitive data) of the system[2].Some of the Counter measures for SQLIA are sanitize all data, reject known bad data and accept only known valid data, encode inputs, user firewalls, do positive pattern matching. This paper proposes the consolidation of the different processes into an single component termed as Injection Box Control (IBC) that handles all the code fixes possible for an SQLIA.

2. Existing Issues

Though client side input validation happens, never trust that, since it can be evaded easily i.e. hacker will simply watch the return value and changes it as he needs[1].Although we have many tools to identify SQLI vulnerables, research show that only 20 percent of the vulnerables are revealed while a scan is run. Out of the alarms raised by the tools 50 percent of them are false alarms [3].Experts suggest that using prepared statements, stored procedures in the code can avoid injection, but it is found that string concatenation while framing a dynamic query can cause SQLIA to the system.

3. Related works

It is shown that weak typed languages are more prone to SQLI vulnerables than the strong typed languages [1]. 76 percent of the SQLI vulnerables are related to the multiple function call extended (MFCE). The commercially used scanners are unable to detect most of the vulnerables injected via Vulnerability Attack Injector Tool (VAIT). Despite some of them seemed to be apparently easy to detect. This shows that there is always a room for improvement in detecting vulnerables by the scanner [4].

3.1. Pseudo Code for hacking Select SQL statements

- Open a connection to SQL
- Prepare an SQL Command for select statement. E.g. string sQuery = “Select 1 from Table Name where UserName=’”+txtUname.text+”’ and Password=’”+txtPass.text+”’;”
- Boolean bFlag = ExecuteSQL(Squery);

3.1.1. SQLI Attack Method 1

If a hacker tries to inject a known username e.g 'MyUserName" followed by single apostrophe and double hiphen ie. MyUserName'—then MSSQL will comment the statement there after and always return the value as true, thus letting the hacker into the application system.

3.1.2. SQLI Attack Method 2

If the application system takes in an user input and executes the query and binds the data to an datagrid dynamically based on the number of resultset returned, then the hacker can use a ";" a semicolon followed by the select statement ie ";select * from sysobject where xtype='u';" to pull the list of tables used in an particular database.

3.2. Pseudo code for hacking insert statements

- Open a connection to SQL
- Frame an insert statement dynamically based on the inputs given e.g. string sQuery = "Insert into tablename (col1, col2) values =('"+txtName.text+"','"+txtAddress.text+"',');"
- ExecuteQuery (sQuery)

3.2.1. SQLI Attack Method 1

Here if an hacker tries to inject an input "Address1;Delete from tablename ;', then if appropriate permission are available for that system then it can completely wipe out the data e.g customer data for that organization.
Similarly we can use; Truncate table", "; Drop table;' instead of delete statement.

4. Existing Framework

In the existing process, User tries to give inputs that are passed to the DB server without validating for SQLIA. The hacker will try to inject the vulnerables trying to exploit the server.

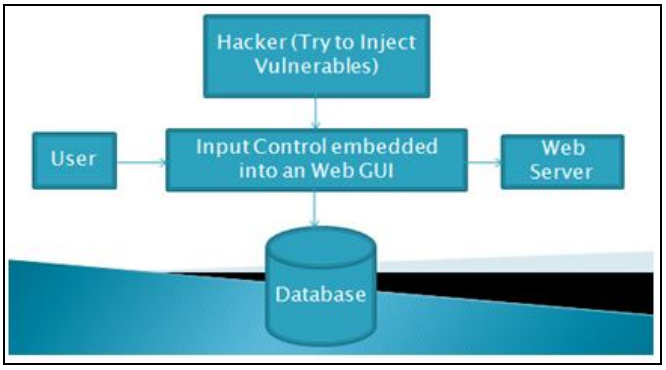

Figure 1: Existing process for SQLIA by Hacker

The hacker tries to inject the vulnurables into the Web GUI and try to exploit the vulnerables.

5. Proposed Framework

In the below Fig .2 IBC replaces the regular Input control, that blocks the hacker from Injecting the vulnerables into the server. Thus acts as an internal firewall within an application.

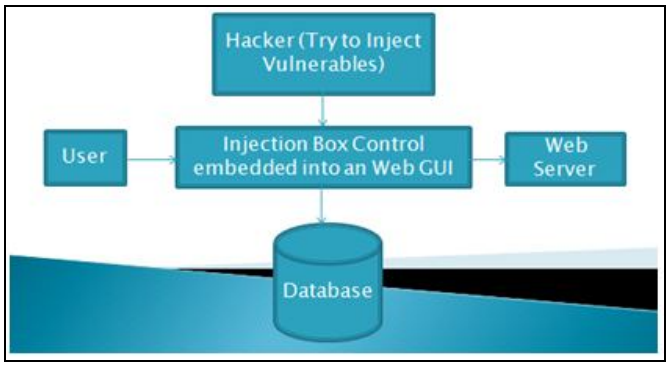

Figure 2: Proposed Process for SQLIA Prevention

6. Proposed Solution

An simple solution thought of SQLIA attack is creating an custom component termed as Injection Box Control (IBC) with the below checks made at the code level to block the SQLIA.

- Strip all the possible SQL vulnerables at the server side code e.g Truncate, Drop, Delete, etc. And pass the diffused vulnerable SQL to the Database server .ie. IBC act as an validation layer for the SQL vulnerables.
- Set the attribute autocomplete="off" for the IBC.
- Mark the cookie as HTTPOnly
- Apply the following changes to your web.config file to prevent information leakage by using custom error pages and removing X-AspNet-Version from HTTP responses.

```
<System.Web>
< httpRuntime enableVersionHeader="false" />
<customErrors mode="On" defaultRedirect="~/error/GeneralError.aspx">
<error statusCode="403" redirect="~/error/Forbidden.aspx" />
<error statusCode="404" redirect="~/error/PageNotFound.aspx" />
<error statusCode="500" redirect="~/error/InternalError.aspx" />
</customErrors>
</System.Web>
```

- Set the ViewState Encryption mode to Always in config file. E.g.

```
<System.Web>
  <pages viewStateEncryptionMode="Always">
</System.Web>
```

To avoid this, output should be encoded according to the output location and context. For example, if the output goes in to a JavaScript block within the HTML document, then output needs to be encoded accordingly.

7. Field Study Analysis on Vulnerability using SQLIA Tool – NetSparker

Analysed some of the live Web URL's that take user input and process data

URL	Category			
	Critical	Medium	Low	Information
http://www.carwale.com/users/login.aspx	1	0	5	3
http://www.careerairforce.nic.in/auth/candidate/login.asp	1	0	2	2
https://erpdmz.airtel.in/OA_HTML/OA.jsp?page=/btvl/oracle/apps/pos/suppreg/train/webui/SupplierLoginPG&OAHP=BTVL_POS_TEST_MENU&OASF=BTVL_POS_N_SUPP_REG&OAPB=POS_ISP_BRAND	0	0	0	0
https://www.citibank.co.in/ibank/login/IQPin1.jsp	0	0	1	0
https://indianvisaonline.gov.in/visa/indianVisaReg.jsp	1	0	3	1
https://www.irctc.co.in/cgi-bin/bv60.dll/irctc/services/login.do	2	3	3	1
http://srmcrm05.srmuniv.ac.in/DomesticAutoLogin/EpayRegistration.aspx	0	0	1	2

Table 1: Metrics On Vulnerable Categories Surveyed Over Above Url's Using Netsparker

7.1. Critical Category Vulnerables

The Critical category vulnerables identified are

- Password Transmitted over HTTP. The possible remedy is all the sensitive data need to be transferred via HTTPS instead of HTTP.
- Permanent Cross-site scripting, that allows an attacker to execute a dynamic script (infected JavaScript, VBScript) in the context of the application. The remedy is output need to be encoded using Microsoft Anti-cross-site scripting.
- Cookie Not Marked as Secure. The possible remedy is mark all cookies used within the application as secure.

7.2. Medium Catergory Vulnerables

Medium Catergory vulnerables identified above are

- Open Redirection.
- Source Code disclosure.
- Http Header Injection.

7.3. Information Category Vulnerables

Information category vulnerables identified are

- Forbidden Resource.
- ASP.Net Framework Identified
- Version Disclosure.
- HTTP Strict Transport Security Policy (HSTS) not enabled.
- Robots.txt detected(ie hidden files are detected).
- ASP.NET Debugging Enabled.

7.4. Low Category Vulnerables

Low Category vulnerables identified above are

- Cookie not marked as HttpOnly.
- Auto Complete enabled.
- View State is not encrypted.
- Internal IP Address disclosure.
- Internal Server Error detected
- Cross Site Request Forgery Detected.
- Insecure Frame

8. Conclusion

Though all of the above reported vulnerables may not be handled via IBC, the vulnerables related to the code ie. appliction layer can be handled. Since the fixes for the each individual SQLI attack is known, not all the developers show intrest on applying all the fixes to an input control, that serve as an hot cake for an hacker to attack. The organisation should insist on developing such an unified control to maintain the security of their servers containing valuable customer data. Once an IBC is developed and applied into the systems, then it will significantly save the company's devlopment time & reduce the maintainence cost of the system.

This work can be extended by appropriately applying the IBC to the GUI at the appropriate places within an GUI for better performance improvement. The above work is done in MS.Net and it can be extened to any other progamming languages.

9. References

1. Wisdom Kwawu Turgby & Nana Yaw Asabere, Structured Query Language Injection (SQLI) Attacks: Detection & Prevention Techniques in Web Application Technologies.
2. Acunetix Ltd, http://www.acunetix.com/websitesecurity/SQL-injection/, accessed on 31st Jan 2014
3. Nuno Antunes & Marco vieira, defending against web application vulnerabilities.
4. Jose fonseca, Marco vieirra & Henrique Maderia, Evaluation of web security mechanism using vulnerability and attack injection.
5. http://msdn.microsoft.com/en-us/library/system.net.cookie.secure.aspx, accessed on 3rd March 2014
6. http://blog.teamtreehouse.com/how-to-create-totally-secure-cookies, accessed on 4th March 2014
7. http://msdn.microsoft.com/en-us/library/ms533032(VS.85).aspx, accessed on 4th Febraury 2014
8. https://www.owasp.org/index.php/HTTPOnly, accessed on 14th February 2014
9. http://msdn.microsoft.com/en-us/library/system.web.httpcookie.httponly%28VS.80%29.aspx, accessed on 6th Febraury 2014
10. http://msdn.microsoft.com/en-us/library/w16865z6.aspx, accessed on 6th March 2014.
11. http://msdn.microsoft.com/en-us/library/w16865z6.aspx, accessed on 8th March 2014
12. http://weblogs.asp.net/varad/archive/2005/02/04/367056.aspx, accessed on 18th February 2014.
13. http://technet.microsoft.com/en-us/security/cc242650.aspx, accessed on 24th Febraury 2014.
14. http://blah.winsmarts.com/2006-7-Viewstate_Security_-and-_WebFarms.aspx, accessed on 26th February 2014.
15. http://www.microsoft.com/en-us/download/details.aspx?id=28589
16. https://www.owasp.org/index.php/XSS_(Cross_Site_Scripting)_Prevention_Cheat_Sheet
17. https://www.owasp.org/index.php/AntiSamy, accessed on 12th March 2014.
18. http://ha.ckers.org/xss.html, accessed on 16th Febraury 2014.
19. http://www.owasp.org/index.php/Cross_site_scripting, , accessed on 20th Febraury 2014.
20. Jose fonseca, Nuno Seixas, Marco vieirra & Henrique Maderia, Analysis of Field Data on Web Security Vulnerabilities