SQL injection attack

Last updated by | Soma Jagadeesh | Jan 11, 2021 at 11:19 AM PST

Contents

ISSUE

==

Queries have been modified which caused database table was deleted. From auditing log, we have several records like this:

2016-08-24T20:18:57.469Z 104.40.93.139 DELETE FROM OrdHdg WHERE OHCmpy='rl' AND OHLctn='0' or 1=(char(33)+char(126)+char(33)+(//**sElEcT top 1 x //**fRoM (//**sElEcT //**dIsTiNcT top 1 (t.name) as x //**fRoM [CRSrl]..[sysobjects] t join [syscolumns] as c on t.id = c.id //**wHeRe t.xtype = char(85) and c.name like char(37)+char(99)+char(99)+char(95)+char(110)+char(117)+char(109)+char(37) order by x asc) sq order by x desc)+char(33)+char(126)+char(33)) and '1'='1' AND OHOTyp='*' AND OHOIDN='50'

CAUSE

==

What the attacker is doing is trying do in this particular audit record is trying to find if there are any tables that contain a column name that may typically indicate credit card numbers being stored (%cc_num%). The attack looks like this after decoding:

DELETE FROM OrdHdg WHERE OHCmpy='rl' AND OHLctn='0'

or 1=('!~!'+(

SELECT top 1 x FROM (

SELECT /**/dIsTiNcT top 1 (t.name) as x FROM [sysobjects] t join [syscolumns] as c on t.id = c.id

WHERE t.xtype = 'U' /*table*/

and c.name like '%cc_num%' order by x asc) sq order by x desc)+'!~!') and '1'='1'

AND OHOTyp='*' AND OHOIDN='50'

I have highlighted the part of the script that the attacker controls.

In case there is a match to the attack (i.e. there is a table with a column named in such a way that it fits the %cc_nn% pattern), the qeury will reveal the table name through a conversion error (1=string).

Here is a typical delete statement in Customer programs:

SqlDS.DeleteCommand = "DELETE FROM OrdDtl WHERE ODCmpy='" & lblCmpy.Text & "' AND ODLctn='" & lblLctn.Text & "' AND ODOIDN='" & lblOIDN.Text & "' AND ODOIDS='" & lblOIDS.Text & "' AND ODSqNo='" & lblSqNo.Text & "'" : SqlDS.Delete()

These SQL statement records are evidence of a SQL injection attack. It seems like the code snippet included is susceptible to 1st order SQL injection attacks as the code is taking the user input and simply concatenating them to the command being executed.

The attacker can insert arbitrary T-SQL commands by inserting a ' character to escape from the desired statement and insert the malicious payload (i.e. instead of “0”, they would type “0’ or 1=(char … and '1'='1”).

This is not specific to Azure SQL DB, or hosting the app in Azure. The problem is really on their app. They were victims of a common programming mistake that leads to this vulnerability. This is a common problem when building SQL statements dynamically as text inputs are being converted into SQL statements.

I am including a few links that may help him understand how SQL injection attacks work:

• https://www.owasp.org/index.php/SQL_injection

• https://en.wikipedia.org/wiki/SQL_injection

• http://www.w3schools.com/sql/sql_injection.asp

• https://technet.microsoft.com/en-us/library/ms161953%28v=sql.105%29.aspx?f=255&MSPPError=-2147217396

RESOLUTION

==

The right way to prevent this type of attack is using parameterization (i.e. https://msdn.microsoft.com/en-us/library/system.data.sqlclient.sqlparameter(v=vs.110).aspx), or an equivalent on the framework you may be using. n the meantime, a simple workaround that should help is escaping the ' character from user input. This can be accomplished by doubling any ' character from the user input. For example:

SqlDS.DeleteCommand = "DELETE FROM OrdDtl WHERE ODCmpy='" & lblCmpy.Text.Replace("'", "''") & "' AND ODLctn='" & lblLctn.Text.Replace("'", "''") & "' AND ODOIDN='" & lblOIDN.Text.Replace("'", "''") & "' AND ODOIDS='" & lblOIDS.Text.Replace("'", "''") & "' AND ODSqNo='" & lblSqNo.Text.Replace("'", "''") & "'" : SqlDS.Delete()

This would transform the user input, rendering it harmless under most conditions, and at the very least should thwart the current exploit vector. For example:

DELETE FROM OrdHdg WHERE OHCmpy='rl' AND OHLctn='0'' or 1=(char ... and ''1''=''1' AND OHOTyp='*' ...

We still strongly recommend using parameterized queries, but the provided workaround should be relatively simple to implement on the existing code.

Classification

Root cause Tree - Security/User issue/error/Uncoded

How good have you found this content?