SQLi süstimine ASP.NET, ADO.NET ja MS SQL serveris

15 aastat tagasi - 14.04.2010 Autor: AM

(Arvutimaailm 4/10, veebis on autori täispikk versioon, lõpus lisandina RTF dokument originaalkujundusega)

? MS SQL Server on paljude veebirakenduste andmebaasimootor ja tänu oma rikkalikele programmeerimisvõimalustele väga sobilik SQLi süstimiseks. Kuidas sellest hoiduda?

! Järjest uuemates SQL Serveri versioonides on küll kasutusel järjest paremad tehnikad süstimise vältimiseks, kuid parim tõrje on ikkagi programmeerija hea töö.

SQLi süstimine, tuntud ka kui SQL INJECTION, on CWE/SANS 2010. aasta aruande (http://cwe.mitre.org/top25/) kohaselt küberkuritegevuses kasutatavatest tehnikatest teisel kohal.

MS SQL Server on paljude veebirakenduste andmebaasimootor ja tänu oma rikkalikele programmeerimisvõimalustele väga sobilik SQLi süstimiseks. Järjest uuemates SQL Serveri versioonides on küll kasutusel järjest paremad tehnikad süstimise vältimiseks (TEXTCOPY utiliidi keelamine, DDL päästikprotsessid, operatsioonisüsteemi poole pöörduvate süsteemsete protseduuride aktiveerimiseks tuleb ligipääs eraldi lubada jne.), kuid parim tõrje on ikkagi programmeerija poolt õieti ja teadlikult süstimist vältivate lausekonstruktsioonide koostamine.

SQL süstimine ise toimub nõnda, et teadlikult valitud sisendparameetrite koostamisega saab muuta algset, programmeerija poolt loodud baasipäringu loogika mõtet. Näiteks alljärgnev andmebaasi päring ASP.NET-is, kus tuleb sisestada kasutaja ID, ja kui selline kasutaja baasist leitakse, siis tehakse midagi, vastasel juhul väljastatakse veateade:

```
SqlCommand komm = new SqlCommand(@"SELECT TOP 1 name FROM dbo.users
WHERE [ID] =" + this.TextBoxUserId.Text.Trim());
Object o = komm.ExecuteScalar();
if (o.Equals(DBNull.Value))
 {
 throw new InvalidOperationException("Vale kasutaja");
 }
```

```
else
{
// tee midagi
}
```

Kui tabeli USERS väli ID on INT tüüpi ja sisestuskasti sisestatakse number 3, siis baasipäring, mida täitma asutakse, on

```
SELECT TOP 1 name FROM dbo.users WHERE ID = 3
```

ehk päringu loogika langeb kokku sellega, mida arendaja mõtles.

Süstija muudab aga baasipäringu loogikat, valides uued sisendaparameetrid, ütleme *3 OR 1=1*. Seega saadetakse andmebaasimootorile täitmiseks päring

```
SELECT TOP 1 name FROM dbo.users WHERE ID = 3 OR 1=1
```

Ehk siis algselt mõeldud ostingutingimuse loogika ID = 3 laieneb hoopis kujule ID = 3 OR 1=1, mis tähendab, et kui tabelis leidub mistahes kirjeid, siis üks kirje väljastatakse alati tänu muudetud loogikatingimusele OR 1=1, ehk 1=1 on alati tõene.

Andmete väljatõmbamiseks kasutatakse SQLi süstimises erinevaid läbistustestimisvahendeid, mille abil võrreldakse erinevust serverile normaalse parameetri ja muudetud parameetriga esitatud päringu vastustes.

Lihtne testmeetod selle erisuse väljatoomiseks on võrrelda näiteks baasipäringute

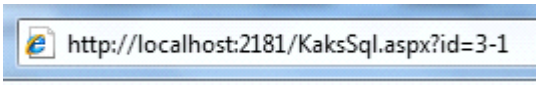
```
SELECT TOP 1 name FROM dbo.users WHERE ID = 3
```

ja

```
SELECT TOP 1 name FROM dbo.users WHERE ID = 3-1
```

esitamise teel erinevusi serveri poolt saadetud vastustes.

Baasipäringule etteantavat parameetrit saab kohandada näiteks sirvikul URL parameetrit muutes ning seejärel visuaalselt väljundi erinevust võrreldes:



Muuta saab ka HiddenField tüüpi muutujate sisu või SOAP päringute parameetreid.

Andmebaaside puistamise võtteid

-Tehete järjekorra alusel:

```
SELECT TOP 1 name FROM dbo.users WHERE ID = 3 - CASEWHEN
SUBSTRING(@@SERVERNAME,1,1) = CHAR(65) THEN 1 ELSE 0 END
```

Võrdustehe tehakse viimasena.

Kräkker kasutab siin meetodit, kus võrreldakse parameetri muutmise teel muutuvat väljundit, CHAR(65), CHAR(66), CHAR(67) jne ehk kui serveri nime esimene täht langeb kokku CHAR funktsiooni poolt tagastatava sümboliga, peab väljund muutuma. Selle meetodiga otsitakse kindlaksmääratud stringi serveri vastuses.

- Veateadetel põhinevad

```
SELECT TOP 1 name FROM dbo.users WHERE ID = 3 / CASEWHEN
SUBSTRING(@@SERVERNAME,1,1) = CHAR(65) THEN 0 ELSE 1 END
```

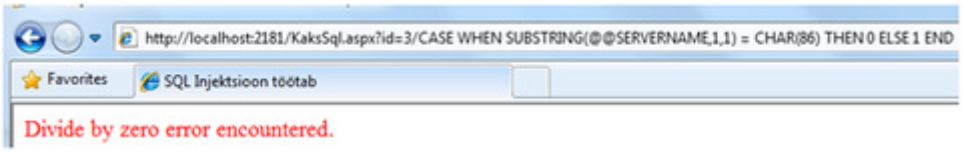
Kui serveri nime esitäht langeb kokku CHAR funkstoooni poolt tagastatava sümboliga, on tulemuseks nulliga jagamine ehk SQLSERVER annab veateate Divide by zero encoutered. Kui see tekst leitakse tagastatavas väljundis, on serveri nime esitäht teada.

Näiteks, kui ASP.NET kood on kirjutatud järgmise konstruktsioonina

```
try

{

  using(...)

  {

   //andmebaasi poole pöördumine

   ...

  }

}

catch(SystemException ex)

{

   this.CustomValidator1.ErrorMessage = ex.Message;

   this.CustomValidator1.IsValid = false;
```

}

Siis sirviku pilt paistab niimoodi, ehk serveri nime esitäht on CHAR(86)



- Otsene veateate tekitamine

```
SELECT TOP 1 name FROM dbo.users WHERE ID = 3/@@SERVERNAME
```

Antud juhul tekib tüübierinevus jagataval ja jagajal, veateade aga, mis serverist seepeale tuleb: Conversion failed when converting the nvarchar value 'SINUSERVERINIMI' to data type int,annab kräkkerile vajaliku info.

- Kahendotsingut võimaldavad

```
SELECT TOP 1 name FROM dbo.users WHERE ID = 3 AND
ASCII(SUBSTRING(@@SERVERNAME,1,1)) < 76
```

ASCII('L')on number 76, 'A'-'Z' on numbrid 65 – 90, binaarotsingut kasutades ei pea järjestikku ükshaaval väärtusi kontrollima, vaid saab need palju vähema katsete arvuga kätte.

- Stringi lõpetamine ja väljakommenteerimine

Ülakoma ’ on SQL-is stringi lõpetaja ja -– rea kommentaar, kõik mis on peale -– loetakse kommentaariks ja andmebaasi mootori poolt täitmisele ei võeta. See meetod on eriti ablas string tüüpi muutujatele.

Näiteks on andmebaasis vaja sooritada päring:

```
SELECT ID FROM dbo.users WHERE NAME = 'NIMI'
```

ASP.NET koodis konstrueeritakse vajalik avaldis nii:

```
stringnimi=this.TextBoxUserName.Text.Trim();  SqlCommandkomm = new
SqlCommand(@"SELECT ID FROM dbo.users  WHERE NAME ='"+ nimi +"'");
```

Sobivalt valitud sisendiga NIMI' OR 1=1 -- saab aga muuta SQL lause loogikat:

```
SELECT ID FROM dbo.users WHERE NAME ='NIMI'OR 1=1 --'
```

Kuna 1=1 on alati tõene ja kasutatakse OR tehet, mis on tõene kui üks operand on tõene, siis päringu tulemusena saadakse baasist tunduvalt rohkem andmeid kui algselt ette nähtud.

- Pakettpäringud, semikooloniga ründeavaldis

```
SELECT TOP 1 name FROM dbo.users WHERE ID = 3;SELECTCASE WHEN SUBSTRING(@@SERVERNAME,1,1) = CHAR(65) THEN 1/0 ELSE 0 END
```

Semikoolon ; on MSSQL lause lõpu märgend. Kõik mis peale ; tuleb, loetakse eraldi SQL lauseks ehk tegu on kahe erineva SQL lausega:

```
SELECT TOP 1 name FROM dbo.users WHERE ID = 3
```

ja

```
SELECTCASE WHEN SUBSTRING(@@SERVERNAME,1,1) = CHAR(65) THEN 1/0 ELSE 0 END
```

Uus sisendparameetri SQL lausekonstruktsioon lisab algsele SQL lause loogikale veateate tekitamise osa analoogselt ülalkirjeldatuga.

- Ajalise viivituse kasutamine pakettpäringuga (Blind Injection).

Kui serveri vastusest veateadet välja ei loe ja ka tagastatav väljund ei muutu, saab kräkker kasutada ajalist viivitust ehk võrrelda serverile esitatud päringu ja saabuva vastuse ajavahemikku.

```
SELECT TOP 1 name FROM dbo.users WHERE ID = 3;
IFSUBSTRING(@@SERVERNAME,1,1) = CHAR(65)              WAITFOR DELAY '00:00:04'
```

Mis masinkeelest tõlgituna tähendab: kui serveri nime esitäht on CHAR(65) ehk A, oota 4 sekundit. Ründaja poolt mõõdetakse, kas tekib ajalist erinevust serverist saabuvates vastustes.

- UNION ründeavaldis

Tõmbab andmebaasist välja rohkem kui algselt mõeldud. Näiteks algne avaldis:

```
SELECTdropdowntekst FROM aspdrop WHERE ID=2
```

Kräkkeri poolt modifitseerituna

```
SELECTdropdowntekst FROM aspdrop WHERE ID=2  UNIONALL SELECT
@@SERVERNAME
```

Lisaks algselt mõeldud andmetele saadakse selle päringuga veel ka serveri nimi. See meetod võimaldab korraga ja kiiresti palju andmeid kätte saada. Sellega suudab kräkker ka andmebaasi väljundi enda kasuks keerata, näiteks seda järjestades:

```
SELECTdropdowntekst FROM aspdrop WHERE ID=2  UNIONALL SELECT
@@SERVERNAME ORDER BY 1 DESC
```

SQL süstimise omapära võrreldes teise meetoditega on see, et pole vaja täiendavaid komponente („troojalast“, „käomuna“ jne.) Kogu vajalik tehniline arsenal on SQL Server-is endas olemas. SQL Server aga ise on suure jõudlusega arvuti.

Vältimine

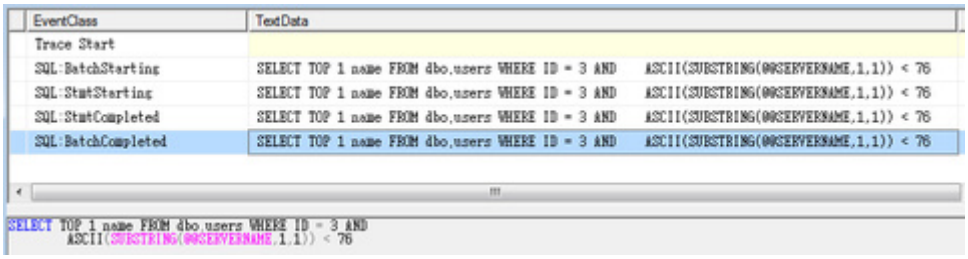
SQL süstimise vältimiseks on hea, kui saadakse aru, kuidas MS SQL Server töötleb temale esitatud päringuid. Seda saab vaadata SQL Profileri nimelise utiliidiga.

SQL süstimise jälgimiseks tuleb *Event*-idest valida SQL:BatchStarting, SQL:StmtStarting, SQL:StmtCompleted, SQL:BatchCompleted

SQL lause

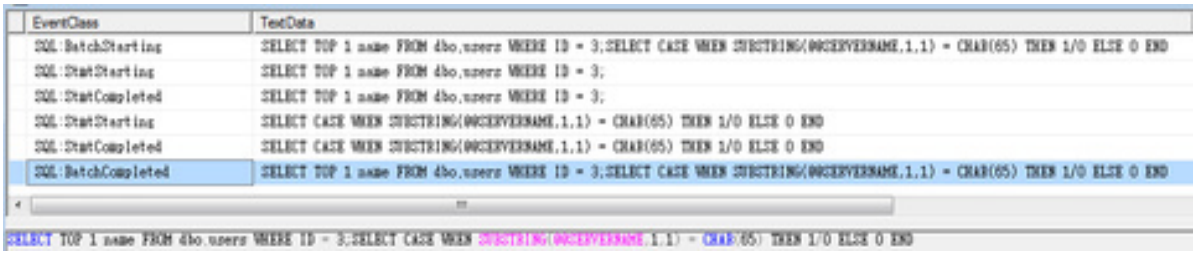
```
SELECTTOP 1 name FROM dbo.users WHERE ID = 3 AND
ASCII(SUBSTRING(@@SERVERNAME,1,1)) < 76
```

töötamine näeb Profileris välja nii



Pakettpäringu täitmine

SELECT TOP 1 name FROM dbo.users WHERE ID = 3;SELECT CASE WHEN SUBSTRING(@@SERVERNAME,1,1) = CHAR(65) THEN 1/0 ELSE 0 END



Semikoolon on SQL Serveris lause lõpetaja ja seetõttu täidetakse SQL Serveris kahte eraldi lauset.

Mida saab programmeerija teha?

SQL süstimist saab parametriseeritud päringute korral vältida tüübiteisendustega.

Kirjutame ASP.NET-is andmebaasi poole pöördumise ümber parametriseeritud päringuna.

> SqlCommandkomm = new SqlCommand(@"SELECT TOP 1 name FROM dbo.users
>
> WHERE [ID] = @id");
>
> komm.Parameters.AddWithValue("@id", Convert.ToInt32(this.TextBoxUserId.Text.Trim()));
>
> Objecto = komm.ExecuteScalar();

Kui nüüd seda lauset täita erinevate SQL süstimise sisenditega

> 3 - CASEWHEN SUBSTRING(@@SERVERNAME,1,1) = CHAR(65) THEN 1 ELSE 0 END
>
> 3 ANDASCII(SUBSTRING(@@SERVERNAME,1,1)) < 76
>
> 3;SELECTCASE WHEN SUBSTRING(@@SERVERNAME,1,1) = CHAR(65) THEN 1/0 ELSE 0 END
>
> 3/@@SERVERNAME

jõuab programm koodi täitmisel kohani

> komm.Parameters.AddWithValue("@id", Convert.ToInt32(this.TextBoxUserId.Text.Trim()));

kus sisendit hakatakse pöörama Convert.ToInt32(this.TextBoxUserId.Text.Trim()) int andmetüübiks, tulemuseks ASP.NET veateade Input string was not in a correct formatehk SQL Serverini lause täitmine ei jõuagi.

Parametriseeritud päringu parameetri saab kirjeldada ka nii:

```
komm.Parameters.Add("@id", System.Data.SqlDbType.Int);

komm.Parameters["@id"].Value = this.TextBoxUserId.Text.Trim();
```

Veateade, mis tuleb Failed to convert parameter value from a String to a Int32. on erinev, aga see saadakse seekord juba ADO.NET-ist.

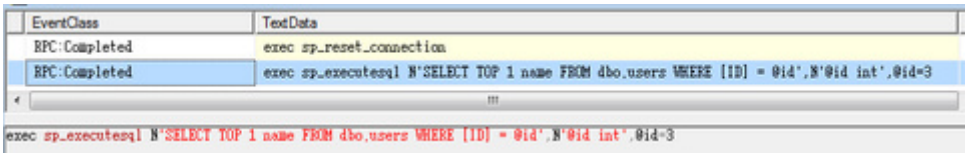
Kuna oodatakse INT tüüpi parameetrit, siis tüübiteisendus kaitseb SQL süstimise vastu.

Sama tüübipööramise kaitset saab ju tegelikult kasutada ka ilma parametriseeritud päringute kasutamisega, kirjutades SQL lause alljärgnevalt:

```
SqlCommandkomm = new SqlCommand(@"SELECT TOP 1 name FROM dbo.users

WHERE [ID] ="+
Convert.ToInt32(this.TextBoxUserId.Text.Trim()).ToString());
```

Põhimõtteline vahe on selles, kuidas SQL Server parametriseeritud päringut käivitab. Seda saab jälle täpselt järele vaadata SQL Profileriga.

Parametriseeritud päringu puhul näeb SQL lause täitmine SQL Profileris välja nii:



SQL lause käivitamiseks kasutatakse käivitusplaani tegevat protseduuri sp_executesql, mis jätab täidetud lause ka SQL Serveri Execution Plan Cache-sse taaskasutamiseks. SQL lause ise jääb muutmatuks, ainult parameeteri väärtused muutuvad, kui sama lauset uuesti kasutada.

Sellele, mida parameerisse @id sisse antakse, ülejäänud SQL lausel „ligipääsu“ ei ole, ehk see, mis parameetrisse kirjutatakse, sinna ka sisse jääb.

Kui ülaltoodud näites parameeter määrata niimoodi

```
komm.Parameters.Add("@id", System.Data.SqlDbType.Int);
```

```
komm.Parameters["@id"].Value = this.TextBoxUserId.Text.Trim();
```

siis SQL Serverisse päringu saatmiseks üritab ADO.NET kräkkeri poolt antud sisendaparameetrit 3-1 pöörata tüüpi Int32, mis aga ei õnnestu ja tulemuseks saab ta tüübipööramisel veateate Failed to convert parameter value from a String to a Int32.

SQL süstimise vältimine parametriseeritud päringutega.

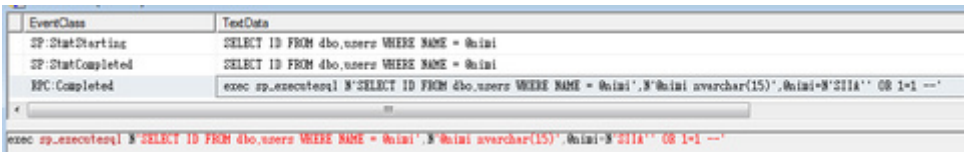
Kui parameetriks on string tüüpi muutuja, siis tüübiteisendusest abi pole.

```
SqlCommandkomm = new SqlCommand(@"SELECT ID FROM dbo.users

WHERE NAME = @nimi");
```

Aga nagu öeldud, parametriseeritud päringu korral SQL lausele parameetrile ligipääsu pole

```
komm.Parameters.AddWithValue("@nimi",
this.TextBoxUserName.Text.Trim());
```

ehk kui üritada eespoolt toodud kräkkimist sisendiga SIIA' OR 1=1 -- siis SQL Profiler näitab lause täitmist niimoodi.



Ehk sisendit SIIA' OR 1=1 -– käsitletaksegi nii nagu ta on ehk sellist stringi andmebaasist otsitaksegi ja SQL süstimise rünnak ei õnnestu.

Stringi tüübipööramine ja parameetri arvamine

Mida parametriseeritud päringute ja tüübipööramiste puhul tuleb tähele panna on see, kuidas ASP.NET ja SQL Server stringi käsitlevad.

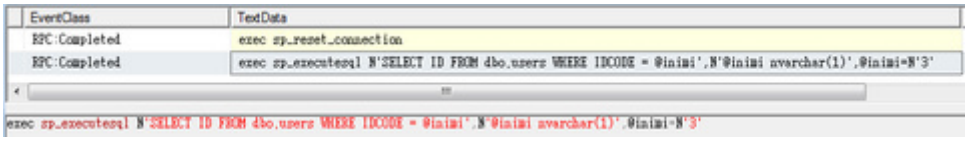
ASP.NET-is on stringid Unicode formaadis, millele SQL Serveris vastab NVARCHAR andmetüüp. Kui nüüd näiteks andmebaasi väli IDCODE on VARCHAR tüüpi ja saata sooritamiseks järgmine päring

```
SqlCommandkomm = new SqlCommand(@"SELECT ID FROM dbo.users
```

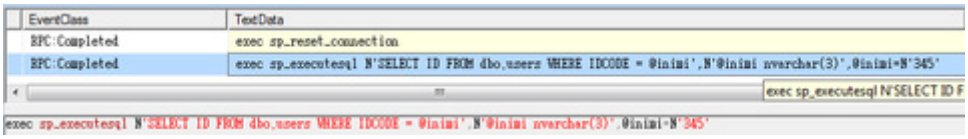
WHERE IDCODE = @inimi");

komm.Parameters.AddWithValue("@inimi",
this.TextBoxUserIdCode.Text.Trim());

ja TextBoxUserIdCode sisestada 3, näitab SQL profiler järgmist pilti



Kui aga TextBoxUserIdCode sisestada 345, näitab SQL profiler järgmist pilti



Pahasti on siin see, et andmebaasi tabeli väli on tüüpi VARCHAR, aga parameetri tüüp on NVARCHAR ehk SQL Server hakkab tegema tüübiteisendust, mis võtab indeksi kasutamise ja koos sellega ka päringu kiiruse maha. Samas SQL Server ei tea, kui pikk string sisse antakse. Kui antakse 3, siis muutuja tüübiks tuleb NVARCHAR(1), kui aga sisendiks 345, siis muutuja tüübiks NVARCHAR(3) ehk tegu on SQL Serveri jaoks kahe erineva andmetüübiga. Sellest tulenevalt rakenduvad ka erinevad käivitusplaanid.

Käivitusplaanide puhvri sisu saab vaadata päringuga

SELECTusecounts, cacheobjtype, objtype, text

FROMsys.dm_exec_cached_plans CROSS APPLY
sys.dm_exec_sql_text(plan_handle)

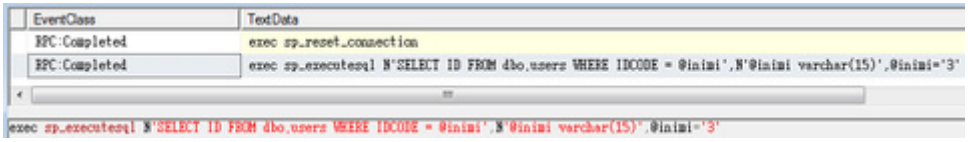
WHEREusecounts > 1

Probleemi saab vältida parameetri tüübi määramisega, ehk kasutame konstruktsiooni

komm.Parameters.Add("@inimi",SqlDbType.VarChar,15);

komm.Parameters["@inimi"].Value = this.TextBoxUserIdCode.Text.Trim();

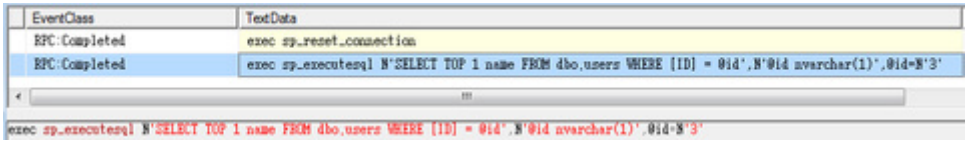
Nüüd pole enam vahet, kas sisestada 3 või 345, SQL Profileris paistab pilt mõlemal puhul ühtemoodi.



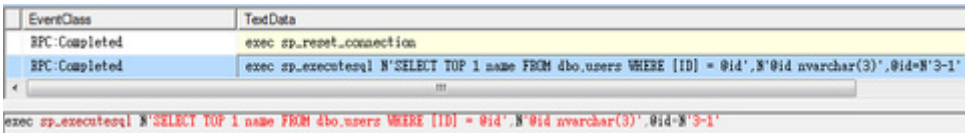
ADO.NET eeldab muidu ise, et kõik parameetrid, millel pole tüüpi otseselt määratud, on Unicode formaadis ehk SQL Server-ini jõuab andmetüüp NVARCHAR. Ehk parameetrit ilma tüübita kirjeldades

```
komm.Parameters.AddWithValue("@id", this.TextBoxUserId.Text.Trim());
```

näeb käivitusplaan välja järgmine:



Kui nüüd parameetriks sisestada 3-1, üritab sp_executesql täita lauset



SQL süstimise hoiab ära tüübipööramine SQL Server-is ja tulemuseks saame veateate

Conversion failed when converting the nvarchar value '3-1' to data type int.

Paha on see, et see veateade tuleb juba SQL Serverist, ehk kräkker on läbistanud mitu andmevahetuskihti.

Numbrite puhul eeldab SQL Server vaikimisi tüübiks INT, kuid ADO.NET on ise õnneks vägagi andekas:

```
komm.Parameters.AddWithValue("@id",
Convert.ToInt16(Request.QueryString["id"].ToString()));
```

täidetaksegi kui SMALLINT, ehk Convert.ToInt16 vastab SQL Serveris SMALLINT andmetüüp ning andmebaasimootori pooolt läheb täitmisele järgnev parametriseeritud päring:

```
exec sp_executesql N'SELECT TOP 1 [name] FROM [dbo].[users] WHERE [ID]
= @id',N'@id smallint',@id=2
```

User Defined Functions väljakutsumine

MS SQL ServerI-sse tehtud kasutajafunktsioonid on süstitavad sisendi kräkkimisel pakettpäringuks. Kirjutame SQL lause üle funktsiooniks

```
CREATEFUNCTION [dbo].[ID_NIMI](@nimi NVARCHAR(50))

RETURNSINT

AS

BEGIN

DECLARE@ret INT

SET@ret=( SELECT ID FROM dbo.users WHERE NAME = @nimi)

RETURN@ret

END
```

Koostame ASP.NET funktsiooni poole pöördumise

```
komm = new SqlCommand(@"SELECT [dbo].[ID_NIMI] ('" +
this.TextBoxSisu.Text.Trim() + "')");
```

Antud pöördumine andmebaasi on süstitav pakettpäringuna koostatud sisendiga:

```
NIMI'); DELETE FROM [dbo].[users]--
```

Andmebaasis täidetakse kahte SQL lauset, semikooloniga eraldatakse esimene

```
SELECT [dbo].[ID_NIMI] ('NIMI');
```

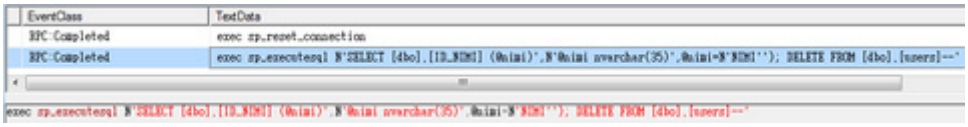
ja teise lause jaoks kommenteeritakse välja algse lause lõpp

```
DELETE FROM [dbo].[users] --')
```

Vältida saab seda parameetritega funktsiooni väljakutsumisel

```
SqlCommandkomm = new SqlCommand(@"SELECT
[dbo].[ID_NIMI](@nimi)");

komm.Parameters.AddWithValue("@nimi", this.TextBoxSisu.Text.Trim());
```

SQL Profileris pilt selline:



Remote Procedure Call käsitleb kõike parameetrisse sisseantud nii nagu see on ja kräkkeri sisend SQL lauset ennast muutma ei ulata.

Salvestatud protseduuride kasutamine

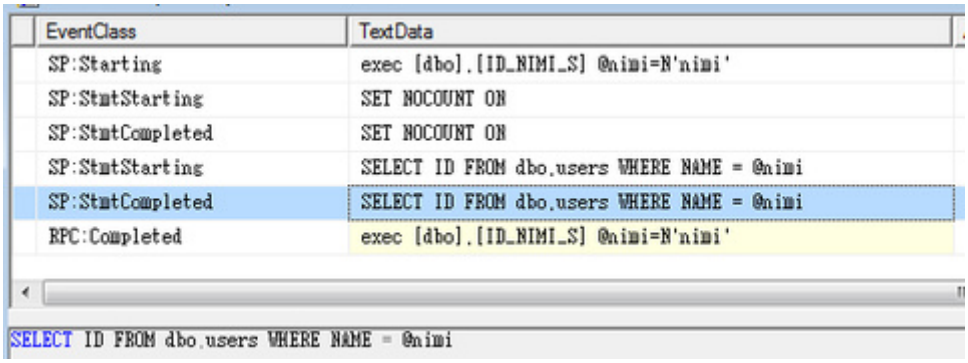
Alates SQL2005 Service Pack 2-st oskab MS SQL Server string tüüpi parameetreid automaatselt lühemaks lõigata. Näiteks sisendaparemeetri @nimi pikkuseks on NVARCHAR(50) ja kui peaks sisestatama üle 50 märgi, siis lõpp lõigatakse salvestatud protseduuri väljakutsumisel maha. Siit ka hea põhjus salvestatud protseduuride kasutamiseks - kräkkeril muutub sobiva ründeavaldise koostamine raskemaks. Ülaltoodud näidetes toodud SQL väljakutsumise saab salvestatud protseduuris teha kolmel erineval viisil, mis kõik annavad sama tulemuse.

```
ALTERPROCEDURE [dbo].[ID_NIMI_S] @nimi NVARCHAR(50)

AS

BEGIN

SETNOCOUNT ON

DECLARE@sql NVARCHAR(MAX)


SELECTID FROM dbo.users WHERE NAME = @nimi


SET@sql='SELECT ID FROM dbo.users WHERE NAME = @nimi'

EXECUTEsp_executesql@stmt=@sql, @params=N'@nimi NVARCHAR(50)',
@nimi=@nimi


SET@sql='SELECT ID FROM dbo.users WHERE NAME = '''+@nimi+''''

EXECUTE(@sql)
```

END

SQL Profiler näitab lausete täitmist niimoodi:

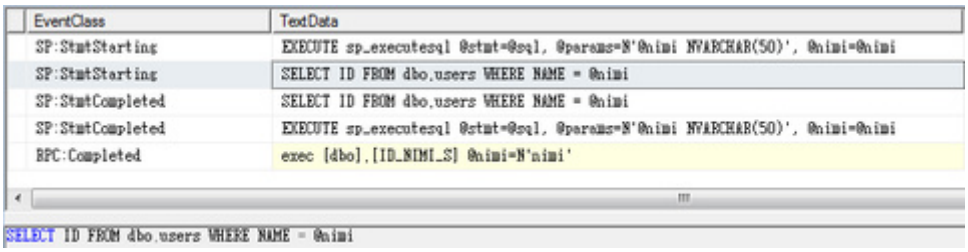
SELECTID FROM dbo.users WHERE NAME = @nimi täidetakse niimoodi



SET@sql='SELECT ID FROM dbo.users WHERE NAME = @nimi'

EXECUTEsp_executesql@stmt=@sql, @params=N'@nimi NVARCHAR(50)', @nimi=@nimi

täidetakse niimoodi

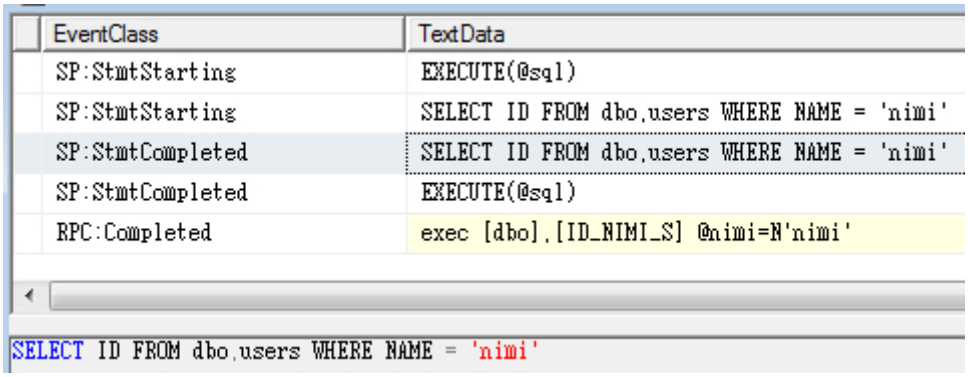


ja

SET@sql='SELECT ID FROM dbo.users WHERE NAME = '''+@nimi+''''

EXECUTE(@sql)

täidetakse niimoodi

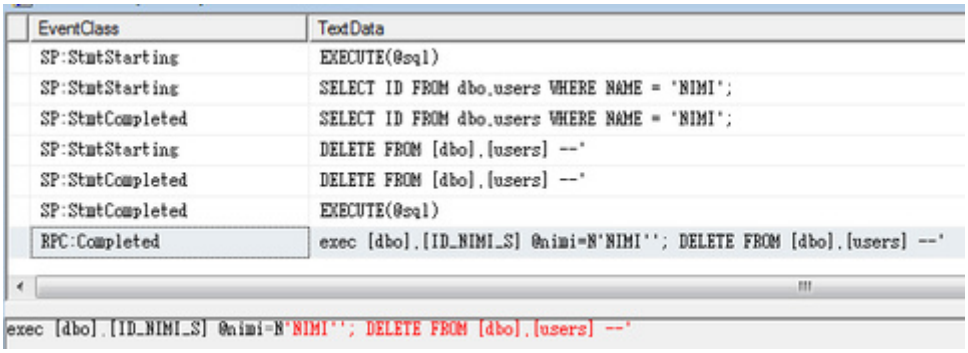


Salvestatud protseduuride kasutamine iseenesest ei taga kaitset SQL süstimise vastu. Oluline on ka teada, kuidas SQL lause salvestatud protseduuris koostatakse ja käivitatakse

Käivitades EXECUTE(@sql) täidetakse seda kui SQL AdHoc päringut, mis on vastuvõtlik SQL süstmisele. Sobivalt valitud sisendiga

NIMI'; DELETE FROM [dbo].[users] --

saab baasipäringu rekonstruktreerida pakettpäringuks



Ehk täidetakse juba kahte eraldi SQL lauset nii nagu kräkker seda tahab.

Dünaamilise SQL-i kasutamine

Mõnel juhul ei saa kasuta parametriseeritud päringud (Prepared statement), näiteks kui on vaja kooostada muutuvate veergude arvuga päring. Sel juhul tuleb SQL lause omavahel osadest kokku liita ja see on olemuselt vastuvõtlik SQL süstimisele.

Tõrjeks annab alati kasutada tüübipööramist, kas .NET koodis või SQL Serveris endas.

Number pööratakse integeriks ja tagasi stringiks, juhul kui sisend on modifitseeritud, saab veateate.

.NET-is

```
Convert.ToInt32(this.TextBoxUserId.Text.Trim()).ToString())
```

SQL-is

```
DECLARE@sql NVARCHAR(MAX)

SET@sql='SELECT TOP 1 name FROM dbo.users WHERE ID  =
'+CAST(CAST(@nimi AS INT) AS NVARCHAR(50))
```

Stringi polsterdamine

Dünaamilise SQL lause saab SQL Serveris ohutult kokku panna polsterdamisega

```
SELECT'' -- saame tühja stringi

SELECT'''' --saame ülakoma ehk lõpetame või alustame stringi

SELECT'''''' --saame kaks ülakoma mis annavad ülakoma sümboli mis aga ei
lõpeta ega alusta stringi
```

Koostame SQL lause, polsterdades stringi tüüpi muutujat @nimi

```
SET@sql='SELECT ID FROM dbo.users WHERE NAME =
'''+REPLACE(@nimi,'''','''''')+''''

EXECUTE(@sql)
```

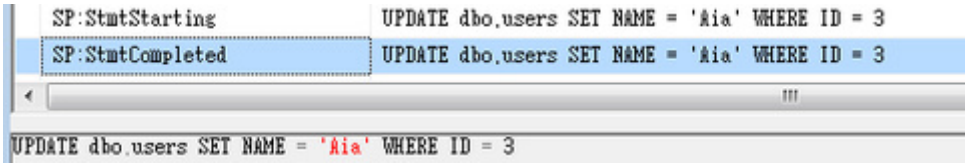
Protseduuri sisseantavas muutujas @nimi polsterdame ühekordse ülakoma kahekordseks, ehk ei luba kräkkeri poolt stringi lõpetamist.

SQL truncation

Polsterdamise kõrvalnäht on see, et REPLACE(@nimi,'''','''''') teeb ühest ülakomast kaks, ehk ruumi SQL lause enda stringi mahust võetakse topelt. Kui @sql muutuja võtta pikkusega DECLARE @sql NVARCHAR(60)

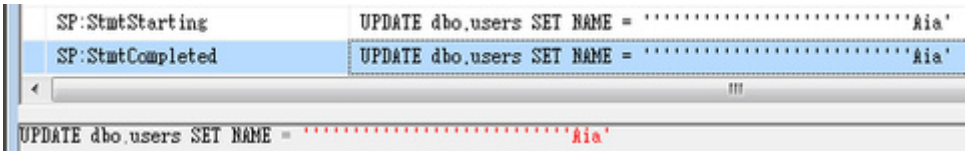
ja kui SQL lauset koostatada nii, et parameetrid on @nimi NVARCHAR(20) ja@id INT, väärtusteks vastavalt 'Aia' ja 3

```
SET@sql='UPDATE dbo.users SET NAME = '''+REPLACE(@nimi,'''','''''')+'''
WHERE ID = '+CAST(@id AS NVARCHAR(10))

EXECUTE(@sql)
```



Kräkker aga duubeldab osavalt muutuja @nimi sisu, ehk REPLACE lause teeb ühest ülakomast kaks, kui kräkkeri sisend on näiteks

''''''''''''''''''''''''Aia'



siis tulemus pole päris see, mis mõeldi, WHERE osa on lausest kadunud.

Odav variant SQL TRUNCATION rünnaku tõrjumiseks on SQL lause muutuja defineerida 2GB mahuga, ehk DECLARE @sql NVARCHAR(MAX). Ära ei maksaks põlata ka LEFT() funktsiooni, mis võtab parameetrist ainult kindlaksmääratud pikkusega osa. Koodi tasemel SQL süstimise tõrjeks võib kasutada ka näiteks SQL kommentaari väljalõikamist REPLACE(@nimi,'--',''), aga riske, mida taoline väljalõikamine endaga kõrvalnähuna kaasa võib tuua, tuleb eelnevalt hinnata. Alati tuleb kasutada igasuguse sisendi kontrollimist ja parameetrite tüübiteisendust. Alati tuleb rakendada igasuguse sisendi ja parameetrite tüübi kontrollimist.

SqlDataSource

SqlDataSource jälgib parametriseeritud päringute mõtet, eelnevalt tehtud User Defined Functioni kasutamine SqlDataSource-na ja

```
<asp:SqlDataSourceID="SqlDataSource1" runat="server"

    DataSourceMode="DataReader"

    SelectCommand="SELECT dbo.ID_NIMI(@nimi) AS Expr1">

    <SelectParameters>
```

```
            <asp:Parameter Name="nimi" />
        </SelectParameters>
    </asp:SqlDataSource>
```

käivitamine näeb SQL Profileris välja parametriseeritud päringu kasutamisena

kuid SQL Server üritab taas parameetri tüüpi ära arvata. Määrates otseselt parameetri andmetüübi

```
    <asp:ParameterName="nimi" DbType="String" Size="50" />
```

saame aga SQL Server-i sõbralikuma lähenemise.

LINQ to SQL

Olemuselt tugev tüübiteisendus

```
    [Column(Storage="_name", DbType="NVarChar(50) NOT NULL",
    CanBeNull=false)]
    publicstring name
    {
        get
          {
            return this._name;
        }
        set
        {
            if ((this._name != value))
            {
                this.OnnameChanging(value);
                this.SendPropertyChanging();
```

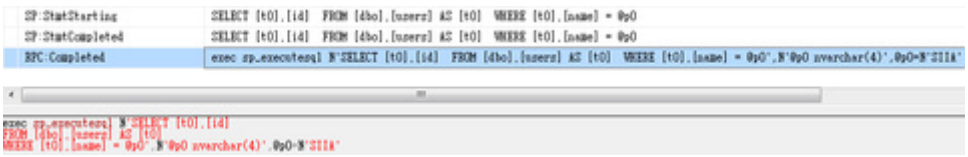
```
                this._name = value;
                this.SendPropertyChanged("name");
                this.OnnameChanged();
            }
        }
    }
```

Koostame Ling to SQL baasipäringu ja käivitame selle.

```
    varquery = from kasutaja in tabelid.users
      where kasutaja.name == this.TextBoxSisu.Text.Trim()
      select kasutaja.id;
    List<int> nimed = query.ToList();
    if(nimed.Count > 0)
    {
      this.TextBoxSisu.Text = nimed[0].ToString();
    }
```

Linq to SQL kasutab parametriseeritud päringut, kuid tegeleb ka parameetri arvamisega.

Erinevate string tüüpi parameetri väärtuste korral saame erinevad käivitusplaanid, kuid SQL süstimine on välditud.



Kokkuvõte

Täiuslikke süsteeme pole olemas, kuid igale ohule saab leida vasturohu. Kübersõdade võitmiseks tuleb kaitsmine odavaks ja ründamine kalliks teha.

Arendagem tarkvara teadlikult.

Kuido Külm
tarkvaraarendaja

- Lahendused

- Tarkvara
- Turvalisus