ITSecTeam
IT Security Research & Penetration Testing Team

راهنمای سریع برای حملات SQL Injection و راه های مقابله

نوشته ی r3dm0v3

www.ITSecTeam.com

forum.ITSecTeam.com

ITSecTeam
IT Security Research & Penetration Testing Team

فهرست مطالب

1-توضیح

تزریق کور (blind injection)

2-کد اسیب پذیر

3-اکسپلویت

اسیب پذیری فرم ورود/خروج رایج

تزریق مبتنی بر خطا (SQL Server)

تزریق مبتنی بر Union

تزریق دستورات SQL

اجرای دستورات CMD

حملات تزریق کور (Blind)

4-چگونگی جلوگیری

استفاده از جستجو های پارامتر بندی شده Parameterized Queries (Prepared Statements)

استفاده از فرایند های ذخیره شده (Stored Procedures)

Escape کردن تمام ورودی های اعمال شده توسط کاربر

اعتبار سنجی توسط لیست سیاه و یا لیست سفید

سایر پدافند ها (تنظیمات)

کمترین اختیارات

جدا سازی سرویس دهنده وب

غیر فعال کردن گزارش خطاها

تنظمیات PHP

www.ITSecTeam.com

ITSecTeam
IT Security Research & Penetration Testing Team

5-کد اطلاح شده

استفاده از جستجو های پارامتر بندی شده (Parameterized Queries (Prepared Statements

استفاده از فرایند های ذخیره شده (Stored Procedures)

Escape کردن تمام ورودی های اعمال شده توسط کاربر

6-مراجع

forum.ITSecTeam.com

ITSecTeam
IT Security Research & Penetration Testing Team

1-توضیح

یک حمله ی تزریق SQL (SQL Injection) شامل وارد کردن و یا تزریق (injection) یک جستجوی SQL از طریق اطلاعات ورودی از سوی کاربر به برنامه می باشد. یک بهره برداری (exploit) موفق تزریق SQL می تواند اطلاعات حساس را از بانک اطلاعات بخواند، اطلاعات را تغییر دهد (Insert/Update/Delete)، عملیات مدیریتی بر روی بانک اطلاعاتی اجرا کند (مانند خاموش کردن DBMS)، محتویات یک فایل بر روی سیستم فایل DBMS را بازیابی کند و در برخی حالات فرمان هایی را برای سیستم عامل صادر کند. حملات تزریق SQL گونه ای از حملات تزریقی هستند که در ان دستورات SQL داخل ورودی، قطار- اطلاعات تزریق می شوند تا بر روی اجرای دستورات از پیش تعیین شده ی SQL تاثیر بگذارند.

تزریق SQL یک تکنیک تزریق کد است که یک اسیب پذیری امنیتی که در لایه بانک اطلاعاتی یک برنامه رخ می دهد را اکسپلویت می کند.

تزریق SQL یکی از قدیمی ترین حملات علیه برنامه های تحت وب می باشد.

این اسیب پذیری هنگامی وجود دارد که کاراکتر های escape رشته های لیترال جاسازی شده در جملات SQL به شکل نادرستی از ورودی کاربر فیلتر شده باشد و یا ورودی کاربر طبقه بندی نشده باشد و در نتیجه به صورت غیر منتظره اجرا می شود. این یک نمونه از کلاس جامعی از اسیب پذیری هاست که هنگامی رخ می دهند که برنامه نویسی و یا زبان اسکریپت نویسی در داخل دیگری جاسازی شده اند.

تزریق کور (Blind SQL Injection)

زمانی که یک نفوذگر حملات تزریق SQL را اجرا می کند بعضی اوقات سرور با پیغام های خطایی از جانب سرور بانک اطلاعاتی پاسخ می دهد حاکی از اینکه گرامر جستجوی SQL غلط است. تزریق کور همان تزریق معمولی است تنها تفاوت در این است که هنگامی که نفوذگر برای اکسپلویت کردن یک برنامه تلاش می کند به جای انکه یک پیغام خطای قابل استفاده دریافت کند یک صفحه ی عمومی مشخص شده توسط برنامه نویس را دریافت کند. این اکسپلویت کردن یک حمله ی تزریق SQL را سخت تر می کند اما غیر ممکن نمی سازد. یک نفوذگر همچنان می تواند اطلاعات را با پرسیدن یک سری سوالات بله و خیر (True/False) از طریق جملات SQL برباید.

www.ITSecTeam.com

ITSecTeam
IT Security Research & Penetration Testing Team

2-کد آسیب پذیر

تزریق SQL زمانی اتفاق می افتد که یک برنامه نویس ورودی که مستقیما در داخل جمله ی SQL قرار داده شده است از کاربر قبول می کند و به درستی کاراکتر های خطرناک ان فیلتر نشده است. این کار به نفوذگر اجازه می دهد تا نه فقط اطلاعات را بدزد بلکه حتی انها را تغییر و یا حذف هم بکند. برخی از سرویس دهنده های بانک اطلاعاتی مانند Microsoft SQL Server دارای فرایند های ذخیره شده و اضافی (توابع سرور بانک اطلاعاتی) هستند. اگر یک نفوذگر بتواند به این فراند ها دسترسی پیدا کند امکان دارد کل سیستم را در اختیار بگیرد. نفوذگر ها اغلب برای تست تزریق SQL کاراکتر نقل قول تکی (single quote) را در کلمه ی جستجوی یک URL و یا در داخل یک فرم وارد می کنند. اگر یک نفوذگر پیغام خطایی شبیه به انچه در زیر امده دریافت کند به احتمال زیاد برنامه به تزریق SQL اسیب پذیر است.

```
Microsoft OLE DB Provider for ODBC Drivers error '80040e14'

[Microsoft][ODBC SQL Server Driver][SQL Server]Incorrect syntax near the

keyword 'or'.

/wasc.asp, line 69
```

هر کدی که از ورودی های کاربر برای تولید جستجوی های SQL بدون بررسی استفاده می کند نسبت به تزریق SQL اسیب پذیر است.

PHP)

```
<?

$query="SELECT * from tbl_TableName where id=".$_GET['ID']; //user input is directly used for generating query!

$q=mysql_query($query);

?>
```

Asp)

```
v_cat = request("category")

sqlstr="SELECT * FROM product WHERE PCategory='" & v_cat & "'"

set rs=conn.execute(sqlstr)
```

Asp)

```
Dim SQL As String = "SELECT Count(*) FROM Users WHERE UserName = '" & _

username.text & "' AND Password = '" & password.text & "'"
```

www.ITSecTeam.com

ITSecTeam
IT Security Research & Penetration Testing Team

```
Dim thisCommand As SQLCommand = New SQLCommand(SQL, Connection)
Dim thisCount As Integer = thisCommand.ExecuteScalar()
```

Asp)

```
dim conn, cmd, recordset

'Create Connection

Set conn=server.createObject("ADODB.Connection")

conn.open "DNS=LOCAL"

'Create Command

set cmd = server.createobject("ADODB.Command")

With cmd

 .activeconnection=conn

 .commandtext="Select * from DataTable where Id = " &
Request.QueryString("Parameter") 'Vulnerable Line!

End With

'Get the information in a RecordSet

set recordset = server.createobject("ADODB.Recordset")

recordset.Open cmd, conn

'....

'Do whatever is needed with the information

'....

'Do clean up

recordset.Close

conn.Close

set recordset = nothing

set cmd = nothing

set conn = nothing
```

Aspx)

```
protected void Button1_Click(object sender, EventArgs e)
```

www.ITSecTeam.com

ITSecTeam
IT Security Research & Penetration Testing Team

```
{
  string connect = "MyConnString";

  string query = "Select Count(*) From Users Where Username = '" +
UserName.Text + "' And Password = '" + Password.Text + "'";

  int result = 0;

  using (var conn = new SqlConnection(connect))

  {

    using (var cmd = new SqlCommand(query, conn))

    {

      conn.Open();

      result = (int)cmd.ExecuteScalar();

    }

  }

  if (result > 0)

  {

    Response.Redirect("LoggedIn.aspx");

  }

  else

  {

    Literal1.Text = "Invalid credentials";

}
```

Aspx)

```
string connect = "MyConnString";

string query = "Select * From Products Where ProductID = " + Request["ID"];

using (var conn = new SqlConnection(connect))

{

  using (var cmd = new SqlCommand(query, conn))

  {

    conn.Open();
```

www.ITSecTeam.com

ITSecTeam
IT Security Research & Penetration Testing Team

```
    //Process results
  }

}
```

PHP)

```
<?

$query="SELECT * from tbl_TableName where name='".
str_replace("'","''",$_GET['name']) ."'"; //vulnerable to input \';drop
tbl_tablename--

$q=mysql_query($query);

?>
```

Java)

```
String DRIVER = "com.ora.jdbc.Driver";

String DataURL = "jdbc:db://localhost:5112/users";

String LOGIN = "admin";

String PASSWORD = "admin123";

Class.forName(DRIVER);

//Make connection to DB

Connection connection = DriverManager.getConnection(DataURL, LOGIN,
PASSWORD);

String Username = request.getParameter("USER"); // From HTTP request

String Password = request.getParameter("PASSWORD"); // From HTTP request

int iUserID = -1;

String sLoggedUser = "";

String sel = "SELECT User_id, Username FROM USERS WHERE Username = '"
+Username + "' AND Password = '" + Password + "'";

Statement selectStatement = connection.createStatement ();

ResultSet resultSet = selectStatement.executeQuery(sel);

if (resultSet.next()) {

       iUserID = resultSet.getInt(1);
```

www.ITSecTeam.com

ITSecTeam
IT Security Research & Penetration Testing Team

```
        sLoggedUser = resultSet.getString(2);
}

PrintWriter writer = response.getWriter ();

if (iUserID >= 0) {

        writer.println ("User logged in: " + sLoggedUser);

} else {

        writer.println ("Access Denied!")

}
```

Java)

```
 String query = "SELECT account_balance FROM user_data WHERE user_name = "

   + request.getParameter("customerName");

 try {

      Statement statement = connection.createStatement( … );

      ResultSet results = statement.executeQuery( query );

 }
```

Hibernate Query Language (HQL))

```
Query unsafeHQLQuery = session.createQuery("from Inventory where
productID='"+userSuppliedParameter+"'");
```

www.ITSecTeam.com

ITSecTeam
IT Security Research & Penetration Testing Team

3-اکسپلویت

حملات تزریق SQL به نفوذگر اجازه کارهایی از قبیل کارهای زیر را می دهد. جعل هویت، مداخله کردن در اطلاعات موجود، انکار فرامین مانند بی اثر کردن تراکنش ها و یا تغییر تراز ها، اجازه برای کامل کردن فاشسازی تمام اطلاعات بر روی سیستم، نابود کردن اطلاعات و یا غیرقابل دسترسی کردن ان و در دست گرفتن مدیریت سرور بانک اطلاعاتی.

تزریق SQL در بین برنامه های PHP و ASP به خاطر شیوع رابط کارکردی قدیمی خیلی رایج است. به علت طبیعت رابط برنامه نویسی موجود، برنامه های Java EE و ASP.NET به نظر کمتر به اسانی تزریق کد های اکسپلویت شده دارند.

سختی حملات تزریق SQL محدود به مهارت نفوذگر و تخیلش است. و کمی هم اقدامات متقابل دفاعی مانند ارتباطات با دسترسی کم به سرور بانک اطلاعاتی و سایر. به طور کلی تزریق کد را یک برخورد سخت در نظر بگیرید.

توجه: این مقاله اکسپلویت کردن کامل تزریق SQL را توضیح نمی دهد زیرا اکسپلویت کردن باگ های تزریق SQL خیلی گوناگون است و این مقاله بر مبنای شناسایی و بر طرف نمودن این اسیب پذیری ها است. در زیر مثال هایی برای متوجه شدن حملات تزریق SQL وجود امده است.

یک نفوذگر تلاش می کند تا موقعیت های استثنایی (exception conditions) و رفتار های غیر عادی از برنامه تحت وب را استنباط کند. این کار را بوسیله ی دست کاری ورودی های تعیین شده – با استفاده از کاراکتر های خاص، فاصله، کلمات کلیدی SQL، درخواست های با حجم زیاد و سایر – انجام می دهد. هرگونه عکس العمل غیرمنتظره از سوی برنامه تحت وب مورد توجه و بررسی قرار می گیرد. ممکن است این رفتار به شکل پیغام های خطای اسکریپت نویسی (شاید همراه با تکه هایی از کد)، خطاهای سرور (HTTP 500) و یا صفحات نیمه بارگذاری شده باشد.

نفوذگر ها اغلب ورودی های زیر را امتحان می کنند تا تشخیص دهند ایا برنامه تحت وب باگ تزریق SQL دارد یا نه.

```
'
 or 1=1
 or 1=1--
 " or 1=1--
 or 1=1--
' or 'a'='a
" or "a"="a
') or ('a'='a
```

www.ITSecTeam.com

ITSecTeam
IT Security Research & Penetration Testing Team

اسیب پذیری فرم ورود/خروج رایج

ما یک کد ساده ی اسیب پذیر را که شبیه به این می باشد استفاده می کنیم:

```
<%@LANGUAGE = JScript %>
<%
var cn = Server.createobject( "ADODB.Connection" );
cn.connectiontimeout = 20;
cn.open( "localserver", "sa", "password" );
var username;
var password;
username = Request.form("username");
password = Request.form("password");
var sql = "select * from users where username = '" + username + "' and
password = '" + password + "'";
rso.open( sql, cn );
if (rso.EOF)
{
rso.close();
%>
Access Dennied!
%>
Response.end
return;
}
else
{
Session("username") = rso("username");
%>
Welcome
<% Response.write(rso("Username"));
Response.end
```

forum.ITSecTeam.com
www.ITSecTeam.com

ITSecTeam
IT Security Research & Penetration Testing Team

```
}
%>
```

نقطه ی خطرناک در اینجا قسمتی از کد است که 'رشته ی جستجو' را می سازد.

```
var sql = "select * from users where username = '" + username + "' and
password = '" + password + "'";
```

در یک ورود معمولی زمانی که کاربر ورودی های زیر را وارد می کند:

Username: John

Password: 1234

رشته ی جستحوی هست:

```
select * from users where username = 'John' and password = '1234'
```

اما اگر کاربر ورودی را مانند زیر دستکاری کند:

Username: John

Password: i_dont_know' or 'x'='x

جستجو (query) می شود:

```
select * from users where username = 'John' and password = 'i_dont_know' or 'x'='x'
```

بنابراین عبارت where برای تمامی سطر های جدول صحیح می باشد و کاربر می تواند بدون دانستن پسورد وارد شود.

اگر کاربر ورودی های یزر را تعیین کند:

Username: '; drop table users--

Password:

جدول 'users' حذف خواهد شد و دسترسی به برنامه برای تمام کاربر ها قطع خواهد شد. توالی کاراکتر '—' برای توضیح تک خطی است و کاراکتر ';' پایان یک جستجو (query) و شروع دیگری را مشخص می کند. '—' د ر انتهای نام کاربری لازم است تا این جستجو بخصوص بدون خطا پایان بپذیرد.

نفوذگر می توانست به عنوان هر کاربری که نام کاربری ان را می دانست وارد شود، با استفاده از ورودی زیر:

Username: admin'--

نفوذگر می توانست به عنوان اولین کاربر در جدول 'users' وارد شود، با ورودی زیر:

Username: ' or 1=1--

www.ITSecTeam.com

ITSecTeam
IT Security Research & Penetration Testing Team

... و در کمال تعجب، نفوذگر با ورودی زیر می تواند به عنوان یک کاربر کاملا ساختگی وارد شود:

```
Username: ' union select 1, 'fictional_user', 'some_password', 1--
```

دلیل انکه همچین چیزی کار میکند ان است که برنامه باور دارد که سطر 'ثابت' که با استفاده از دستور union توسط نفوذگر مشخص شده قسمتی از recordset دریافتی از بانک اطلاعاتی می باشد.

تزریق مبتنی بر خطا (SQL Server)

این نوع از حمله مبتنی بر 'پیغام های خطایی' است که از سرور دریافت می شود. خوشبختانه (برای نفوذگر) اگر پیغام های خطا از برنامه برگردانده شوند (رفتار پیش فرض ASP) نفوذگر می تواند تمام ساختار بانک اطلاعاتی را تشخیص دهد و هر مقدار قابل خواندن – توسط کاربری که برنامه ی ASP برای اتصال به سرور SQL استفاده می کند – را بخواند.

برای مثال نفوذگر می تواند ورژن سرور SQL را با تزریق عبارت زیر بدست اورد:

```
Username:' union select @@version,1,1,1--
```

که خطای زیر را تولید می کند که ورژن سرور SQL را نمایش می دهد:

```
Microsoft OLE DB Provider for ODBC Drivers error '80040e07'

[Microsoft][ODBC SQL Server Driver][SQL Server]Syntax error converting the nvarchar value
'Microsoft SQL Server 2000 - 8.00.194 (Intel X86) Aug 6 2000 00:57:48 Copyright (c) 1988-2000
Microsoft Corporation Enterprise Edition on Windows NT 5.0 (Build 2195: Service Pack 2) ' to a
column of data type int.

/process_login.asp, line 11
```

پیدا کردن نام جدول ها و ستون ها:

```
Username: ' having 1=1--
```

که خطای زیر را ایجاد می کند:

```
Microsoft OLE DB Provider for ODBC Drivers error '80040e14'

[Microsoft][ODBC SQL Server Driver][SQL Server]Column 'users.id' is invalid in the select list because
it is not contained in an aggregate function and there is no GROUP BY clause.

/login.asp, line 11
```

www.ITSecTeam.com

ITSecTeam
IT Security Research & Penetration Testing Team

بنابراین نفوذگر الان اسم جدول و اولین ستون در جستجو را می داند. سپس می تواند با معرفی هر فیلد در داخل یک عبارت 'group by' این کار را ادامه دهد، مانند زیر:

Username: ' group by users.id having 1=1--

Microsoft OLE DB Provider for ODBC Drivers error '80040e14'

[Microsoft][ODBC SQL Server Driver][SQL Server]Column 'users.username' is invalid in the select list because it is not contained in either an aggregate function or the GROUP BY clause.

/process_login.asp, line 11

با ادامه دادن این کار نفوذگر می تواند تمام ستون ها را پیدا کند.

توجه: نمی توانید این کار را برای سایر بانک های اطلاعاتی انجام دهید. برای اطلاعات بیشتر می توانید به منابع مراجعه کنید.

تزریق مبتنی بر Union

عملگر Union برای ملحق کردن نتیجه ی دو یا چند جمله ی Select استفاده می شود. در این نوع از تزریق نفوذگر تلاش می کند، برای تغییر نتیجه جستجو و خواندن اطلاعات عملگر union را به جستجو تزریق کند.

حمله ی مبتنی بر Union شبیه زیر می باشد:

Username: junk' union select 1,2,3,4,... --

توجه داشته باشید که هر جمله SELECT همراه با union باید به تعداد یکسانی ستون داشته باشد. ستون ها باید نوع (type) مشابه ای داشته باشند. همچنین ستون ها در هر جمله ی select باید در ترتیب مشابهی باشند. نفوذگر می تواند تعداد ستون ها را با سعی و خطا بدست اورد. (با استفاده از union و یا order by)

توجه: عملگر union به طور پیش فرض تنها مقادیر مجزا (distinct) را انتخاب می کند. برای انتخاب مقادیر تکراری از union all استفاده کنید.

برای مثال نفوذگر می تواند ورژن سرور SQL را استفاده از تزریق عبارت زیر بدست اورد:

Username:' union select 1,@@version,1,1--

که ورژن سرور SQL را در ستون username بر می گرداند و کد صفحه ورود ان را نمایش می دهد. اگر کد هیچ ستونی از نتیجه ی جستجو را در صفحه نمایش ندهد، نفوذگر نمی تواند اطلاعات را بدین طریق بدست اورد و باید از روش های تزریق کور (blind injection) استفاده کند.

www.ITSecTeam.com

تزریق دستورات SQL

اگر بانک اطلاعاتی جستجو های پشت سر هم (Stacked) را پشتیبانی کند نفوذگر می تواند دستورات SQL را تزریق کند. در اکثر بانک های اطلاعاتی این امکان وجود دارد تا با استفاده از (;) چندین جستجو را در یک تراکنش انجام دهیم.

مثال زیر نحوه ی ایجاد یک جدول با نام foo که یک ستون به اسم line دارد را با استفاده از تزریق دستورات پشت سر هم نشان می دهد.

```
Username: ' create table foo (line varchar(1000))--
```

اجرای دستورات CMD

نفوذگر می تواند برای انجام کارهایی از قبیل اجرای دستورات از فرایند های ذخیره شده استفاده کند.

xp_cmdshell یک فرایند ذخیره شده ی اضافی است که اجازه ی اجرای دستورات دلخواه را می دهد. برای مثال:

```
Username: '; exec master..xp_cmdshell 'dir'--
```

برخی از فرایند های ذخیره شده ی اضافی MS-SQL در زیر لیست شده اند:

* xp_cmdshell - اجرای دستورات شل
* xp_enumgroups - برشمردن گروه های کاربری
* xp_logininfo - مشخصات لاگین کنونی
* xp_grantlogin - واگذاری حقوق ورود
* xp_getnetname - نام سرور WINS را بر می گرداند
* xp_regdeletekey - تغییر رجیستری
* xp_regenumvalues
* xp_regread
* xp_regwrite
* xp_msver - اطلاعات ورژن سرو

ITSecTeam
IT Security Research & Penetration Testing Team

حملات تزریق کور (Blind)

یک نفوذگر ممکن است به دو روش بررسی کند ایا یک درخواست فرستاده شده True و یا False بازگردانده:

محتویات اشکار/پنهان

اگر یک سفحه ی ساده داشته باشیم که مقالات را با ID داده شده به عنوان پارامتر نمایش می دهد، نفوذگر می تواند یک سری تست هایی برای تشخیص اسیب پذیری انجام دهد.

صفحه ی نمونه:

```
http://newspaper.com/items.php?id=2
```

جستجوی زیر را به بانک اطلاعاتی می فرستد:

```
SELECT title, description, body FROM items WHERE ID = 2
```

نفوذگر ممکن است هر جستجویی (query) را تزریق کند(حتی غیر معتبر) که باید منجر به این شود که هیچ نتیجه ای برگردانده نشود:

```
http://newspaper.com/items.php?id=2 and 1=2
```

حالا جستجوی SQL باید شبیه این به نظر برسد:

```
SELECT title, description, body FROM items WHERE ID = 2 and 1=2
```

که بدان معنی است که جستجو نتیجه ای را در بر نخواهد داشت.

اگر برنامه ی تحت وب نسبت به تزریق SQL اسیب پذیر باشد احتمالا چیزی نمایش نخواهد داد. برای اطمینان خاطر نفوذگر یک جستجوی معتبر را تزریق می کند:

```
http://newspaper.com/items.php?id=2 and 1=1
```

اگر محتویات صفحه یکسان باشد انوقت نفوذگر قادر خواهد بود تشخیص دهد چه زمان جستجو True و یا False است.

گام بعد چیست؟ تنها محدودیت ها دسترسی های تنظیم شده توسط مدیر بانک اطلاعاتی ونسخه های مختلف SQL و بلاخره تخیل نفوذگر هستند.

www.ITSecTeam.com

ITSecTeam
IT Security Research & Penetration Testing Team

حمله ی زمانی

یک حمله ی زمانی بر روی تزریق جستجوی MySQL زیر استوار است.

```
SELECT IF(expression, true, false)
```

استفاده از برخی عملیات زمانبر مانند BENCHMARK() پاسخ سرور را به تاخیر می اندازد در صورتی که عباری (expression) صحیح (True) باشد.

```
BENCHMARK(5000000,ENCODE('MSG','by 5 seconds'))
```

- 50000000 بار تابع encode را اجرا می کند

بسته به عملکرد سرور بانک اطلاعاتی و بار بر روی ان، فقط باید یک لحظه تا انجام این عملیات طول بکشد. نکته مهم از منظر نفوذگر تعیین کردن تعداد زیادی باز انجام تابع BENCHMARK() به نحوی است که زمان پاسخ گویی سرور را به مقدار قابل توجهی تغییر دهد.

نمونه ترکیب هردو جستجو:

```
1 UNION SELECT IF(SUBSTRING(user_password,1,1) =
CHAR(50),BENCHMARK(5000000,ENCODE('MSG','by 5 seconds')),null) FROM users WHERE user_id =
1;
```

اگر پاسخ سرور بکلی طولانی بود احتمالا اولین حرف پسورد با id=2 حرف 2 است.

با استفاده از همین روش برای باقی حروف می توان کل پسورد ذخیره شده در بانک اطلاعاتی را بدست اورد. این روش حتی زمانی که نفوذگر جستوی SQL را تزریق می کند ولی محتوای صفحه ی اسیب پذیر تغییر نمی کند نیز کار می کند.

بدیهی است که در این مثال نام جدول و تعداد ستون ها مشخص شده بود. اگرچه می توان انها را حدس زد و یا با روش ها سعی و خطا و یا روش مشابهی بدست اورد.

سایر بانک های اطلاعاتی به غیر از MySQL توابع اجرایی دارند که می توان از انها برای حملات زمانی استفاده کرد.

* MS SQL - WAITFOR DELAY '0:0:5' -- pause for 5 seconds

* PostgreSQL - pg_sleep()

www.ITSecTeam.com

ITSecTeam
IT Security Research & Penetration Testing

4-چگونگی جلوگیری

استفاده از جستجو های پارامتر بندی شده Parameterized Queries (Prepared Statements)

استفاده از جملات اماده شده (پارامتر بندی شده) اولین روشی است که برنامه نویس ها باید برای نوشتن جستجو های بانک اطلاعاتی یاد بگیرند. در مقایسه با جستجو های پویا (dynamic) نوشتن انها ساده و فهمیدن انها اسان است. جستجو های پارامتر بندی شده برنامه نویس را مجبور می کند تا ابتدا کد SQL را مشخص کند و سپس هر پارامتر را به جتسجو بفرستد. این شیوه کد نویسی به بانک اطلاعاتی اجازه می دهد تا صرف نظر از ورودی کاربر کد و اطلاعات را از یکدیگر تمیز دهد.

جملات اماده اطمینان ایجاد می کنند که نفوذگر قادر به تغییر قصد و هدف جتسجو نیست حتی اگر دستورات SQL توسط نفوذگر وارد شده باشند.

توصیه ها در زبان های بخصوص:

* Java EE – use PreparedStatement() with bind variables
* .NET – use parameterized queries like SqlCommand() or OleDbCommand() with bind variables
* PHP – use PDO with strongly typed parameterized queries (using bindParam())
* Hibernate - use createQuery() with bind variables (called named parameters in Hibernate)

در شرایط نادر جملات اماده می توانند به عملکرد (performance) زیان وارد کنند. هنگام مواجه با این وضعیت بهترین کار به جای استفاده از جملات اماده، escape کردن تمام ورودی های کاربر با استفاده از روش مشخص شده برای بانک اطلاعاتی مورد استفاده تان می باشد. گزینه ی دیگر که قادر به برطرف کردن مسئله کارایی (performance) است استفاده از یک فرایند ذخیره شده است.

استفاده از فرایند های ذخیره شده (Stored Procedures)

فرایند های ذخیره شده زمانی که به طرز ایمن مورد استفاده قرار گیرند* نتیجه ی مشابهی با استفاده از جملات اماده دارند. لازم است برنامه نویس ابتدا کد SQL را تعریق کند و سپس پارامتر ها را وارد کند. تفاوت میان جملات اماده و فرایند های ذخیره شده این است که کد SQL برای یک فرایند ذخیره شده تعریف شده و در داخل بانک اطلاعاتی قرار دارد و سپس از طریق برنامه فراخوانی می شود. هر دوی این روش ها تاثیر یکسانی در جلوگیری از تزریق SQL دارند بنابراین سازمان شما باید تشخیص دهد کدام راه حل بهترین برای شما خواهد بود.

*توجه: به طرز ایمن مورد استفاده قرار گیرند یعنی فرایند های ذخیره شده هیچ گونه تولید SQL پویا را شامل نشوند. برنامه نویس ها معمولا SQL پویا در داخل فرایند های ذخیره شده تولید نمی کنند. اگرچه این کار را می توان انجام داد ولی باید از ان اجتناب کرد. فرایند ذخیره

www.ITSecTeam.com

ITSecTeam
IT Security Research & Penetration Testing Team

شده برای اطمینان حاصل کردن از اینکه تمام ورودی های اعمال شده توسط کاربر نمی تواند برای تزریق کد SQL داخل جتسجوی ایجاد شده ی دینامیکی، مورد استفاده قرار بگیرد باید از اعتبار سنجی ورودی و یا escape کردن صحیح استفاده کند.

Escape کردن (خنثی کردن) تمام ورودی های اعمال شده توسط کاربر

روش سوم escape کردن ورودی کاربر قبل از استفاده در جستجو می باشد. این روش می تواند یک روش موثر برای اصلاح باگ تزریق SQL بر روی یک برنامه موجود باشد. چرا که می توان ان را با تقریبا هیچ تغییری بر روی ساختار کد پیاده کرد. اگر شما نگران این هستید که نوشتن جستجوهای دینامیک خود به صورت جملات اماده و یا فرایند های ذخیره شده ممکن است باعث از کار افتادن ان و یا تاثیر بد بر روی عملکرد برنامه شود، این روش بهترین راه حل برای شما خواهد بود.

این روش بدین صورت کار میکند. هر DBMS یک رویه escape کاراکتر را پشتیبانی می کند که شما می توانید با استفاده از ان کاراکتر های ویژه را escape کنید تا برای DBMS مشخص کنید که قصد شما از کاراکترهایی که در جستجو قرار داده اید اطلاعات بوده و نه کد. اگر شما تمام ورودی های اعمال شده توسط کاربر را به طریق صحیحی برای بانک اطلاعاتی escape کنید، انوقت DBMS اطلاعات ورودی را با کد SQL نوشته شده توسط کاربر اشتباه نمی گیرد از این رو از اسیب پذیری های محتمل تزریق SQL جلوگیری می شود.

برای انجام escaping باید رویه escaping مورد پشتیبانی توسط DBMSی که از ان استفاده می کنید را بدانید و یک روتین برای اینکار بنویسید. اطمینان حاصل کنید که از توابع ضعیف برای escaping استفاده نمی کنید مانند addslashes() در PHP و یا توابع جایگزینی کاراکتر مانند str_replace("'", "") (فقط نقل قول تکی را به دو نقل تکی تبدیل می کند). این ها ضعیف هستند و به خوبی توسط نفوذگر ها اکسپلویت شده اند. اطمینان حاصل کنید که روتینی که برای escaping استفاده می کنید **تمام** کاراکتر های خطرناک برای مفسری که اطلاعات را برای ان می فرستید escape می کند.

اعتبار سنجی توسط لیست سیاه و یا لیست سفید

همیشه توصیه می شود تا هرچه سریعتر در هنگام پردازش درخواست کاربر (نفوذگر) از حمله جلوگیری شود. می توان از اعتبار سنجی ورودی (input validation) برای شناسایی ورودی هایی ناخواسته قبل از ارسال ان به جستجوی SQL استفاده نمود. برنامه نویسان بیشتر اوقات اعتبار سنجی لیست سیاه (black list validation) را برای پیدا کردن کاراکترهای حمله و الگوهایی مانند کاراکتر ' یا رشته ی 1=1 انجام می دهند، اما این یک راه حل ناقص است چرا که نفوذگر خیلی ساده می تواند از کاراکتر هایی که توسط همچین فیلتر هایی گرفته شده است استفاده نکند. به علاوه همچین فیلتر هایی خیلی وقت ها مانع ورودی های مجاز هم می شوند مانند O'Brian زمانی که کاراکتر ' فیلتر می شود.

www.ITSecTeam.com

ITSecTeam
IT Security Research & Penetration Testing Team

اعتبار سنجی لیست سفید مناسب تمام فیلد های ورودی کاربر است. اعتبار سنجی لیست سفید دقیقا مشخص می کند که چه چیز مجاز است، و بنا بر تعریف هرچیز دیگر غیر مجاز است. اگر ورودی دارای ساختار مشخصی باشد مانند تاریخ، شماره های امنیت اجتماعی، کد پستی، ادرس ایمیل و غیره، برنامه نویس باید قادر به تعریف الگوی اعتبار سنجی دقیقی باشد که معمولا بر اساس عبارت عادی (regular expressions) برای اعتبار سنجی اینگونه ورودی ها می باشد. اگر ورودی از مجموعه ی ثابتی از گزینه ها می اید مانند یک لیست drop down و یا radio buttons انوقت ورودی باید با یکی از مقادیر پیشنهاد شده به کاربر مطابقت داشته باشد. سخت ترین فیلد ها برای اعتبار سنجی "متن ازاد" (free text) نامیده می شوند مانند مطالب وبلاگ. اگرچه حتی این ورودی هارا هم می توان تا حدودی اعتبار سنجی کرد، می توانید حداقل تمام کاراکتر های غیر قابل چاپ را حذف کنید و یک حداکثر سایزی را برای ورودی در نظر بگیرید.

مراقب باشید که اعتبار سنجی ورودی صرفا به معنی حذف کاراکتر نقل قول نیست چرا که حتی کاراکترهای معمولی هم می توانند مشکل ایجاد کنند.

سایر پدافند ها (تنظیمات)

کمترین اختیارات

برنامه تحت وب نباید برای تمامی تراکنش ها با بانک اطلاعاتی از یک ارتباط با بیشترین دسترسی استفده کند. چرا که در این صورت اگر باگ تزریق کدی مورد اکسپلویت قرار گیرد بیشترین دسترسی را در اختیار نفوذگر قرار می دهد. بنابراین بهتر است برنامه تحت وب از ارتباطی متناسب با کاربر استفاده کند برای مثال برای کاربر وارد نشده باید از یک ارتباط با اجازه خواندن استفاده کند و تنها به جداول خاصی دسترسی داشته باشد و دسترسی به سایر جداول را قطع کند.

تاثیر این کار ان است که حتی یک حمله موفق تزریق SQL موفقیت خیلی کمتری خواهد داشت چرا که نقوذگر قادر به انجام درخواست UPDATE که دسترسی را می تواند برای او فراهم کند، نیست.

جدا سازی سرویس دهنده وب

حتی با اعمال تمام این مراحل محدود سازی، بازهم این امکان وجود دارد که چیزی فراموش شده باشد و باعث به خطر افتادن سرور شود. کسی که باید ساختار شبکه را طراحی کند باید فرض کند که نفوذگرها دسترسی کامل به سیستم دارند و سپس برای محدودکردن راه های نفوذ برای دسترسی به سایر چیز ها تلاش کند.

برای نمونه قرار دادن سیستم در یک DMZ با راه های ارتباطی به شدت محدود شده به داخل شبکه بدان معنی است که حتی به دست گرفتن کامل کنترل وب سرور به صورت خودکار دسترسی کامل به چیز دیگری را فراهم نمی دهد. البته این باعث توقف همه چیز نمی شود ولی این کار را بسیار سخت تر می کند.

www.ITSecTeam.com

ITSecTeam
IT Security Research & Penetration Testing Team

غیر فعال کردن گزارش خطاها

گزارش خطای پیش فرض برای برخی framework ها شامل اطلاعات دیباگ کردن برای برنامه نویس ها می باشد و برای کاربران بیرونی نمایش داده نمی شود. تصور کنید که چقدر برای یک نفوذگر ساده تر می شود زمانی که کل جستجو (query) مربوط به خطای پیش امده نمایش داده شود.

این اطلاعات برای برنامه نویسان مفید هستند ولی باید فقط محدود به کاربران داخلی باشند (در صورت امکان).

در ASP.NET می توانید با تنظیم عنصر CustomErrors در فایل web.config نمایش خطا را غیر فعال کنید. در زیر یک مثال امده:

```
<customErrors mode="RemoteOnly">
```

در PHP می توانید این کار را با استفاده از ;error_reporting(0) در هر صفحه و یا با تنظیمات php.ini انجام دهید.

تنظمیات PHP

تنظیمات PHP تاثیر مستقیم در شدت حمله ها دارد. اگرچه هیچ CVE بخصوص در رابطه با تنظیمات وجود ندارد، ولی تنظمیات ضعیف امتیازات و خسارات نفوذگر به سیستم با پیکر بندی ضعیف را حداکثر می کند. چیزی که غلط است ان است که بسیاری از تنظیمات امنیتی در PHP به طور پیش فرض اشتباه ست شده اند و مفهوم غلطی از امنیت را ایجاد می کنند.

باعث شگفتی است که هیچ تنظیمات PHP مورد توافقی وجود ندارد.و حتی چگونگی تنظیمات پیش فرض جای تعجب بیشتری دارد. اینها ننظیمات امنیتی رایج هستند:

* register_globals (off by default in modern PHP, should be off)

* allow_url_fopen (enabled by default, should be off)

* magic_quotes_gpc (on by default in modern PHP, should be off)

* magic_quotes_runtime (off by default in modern PHP, should be off)

* safe_mode and open_basedir (disabled by default, should be enabled and correctly configured. Be aware that safe_mode really isn't safe and can be worse than useless)

www.ITSecTeam.com

ITSecTeam
IT Security Research & Penetration Testing Team

5-کد اطلاح شده

استفاده از جستجو های پارامتر بندی شده Parameterized Queries (Prepared Statements)

Java)

```
String custname = request.getParameter("customerName"); // This should
REALLY be validated too

// perform input validation to detect attacks

String query = "SELECT account_balance FROM user_data WHERE user_name = ?
";

PreparedStatement pstmt = connection.prepareStatement( query );

pstmt.setString( 1, custname);

ResultSet results = pstmt.executeQuery( );
```

Java)

```
String firstname = req.getParameter("firstname");

String lastname = req.getParameter("lastname");

// FIXME: do your own validation to detect attacks

String query = "SELECT id, firstname, lastname FROM authors WHERE forename
= ? and surname = ?";

PreparedStatement pstmt = connection.prepareStatement( query );

pstmt.setString( 1, firstname );

pstmt.setString( 2, lastname );

try

{

      ResultSet results = pstmt.execute( );

}
```

C#.net)

```
String query =

      "SELECT account_balance FROM user_data WHERE user_name = ?";
```

www.ITSecTeam.com

ITSecTeam
IT Security Research & Penetration Testing Team

```
try {
      OleDbCommand command = new OleDbCommand(query, connection);

      command.Parameters.Add(new OleDbParameter("customerName",
CustomerName Name.Text));

      OleDbDataReader reader = command.ExecuteReader();

      // …

} catch (OleDbException se) {

      // error handling

}
```

Perl)

```
$sth = $dbh->prepare("SELECT email, userid FROM members WHERE email = ?;");

$sth->execute($email);
```

Hibernate Query Language (HQL) Prepared Statement (Named Parameters) Examples

```
Query safeHQLQuery = session.createQuery("from Inventory where
productID=:productid");

safeHQLQuery.setParameter("productid", userSuppliedParameter);
```

PHP)

در PHP ورژن 5 و بالاتر، چندین انتخاب برای استفاده از جملات پارامتر بندی شده وجود دارد.لایه ی بانک اطلاعاتی PDO یکی از انهاست:

```
$db = new PDO('pgsql:dbname=database');

$stmt = $db->prepare("SELECT priv FROM testUsers WHERE username=:username
AND password=:password");

$stmt->bindParam(':username', $user);

$stmt->bindParam(':password', $pass);

$stmt->execute();
```

www.ITSecTeam.com

ITSecTeam
IT Security Research & Penetration Testing Team

Java)

این یک مثال JAVA با استفاده از JDBC API است:

```
PreparedStatement prep = conn.prepareStatement("SELECT * FROM USERS WHERE
USERNAME=? AND PASSWORD=?");

prep.setString(1, username);

prep.setString(2, password);

prep.executeQuery();
```

PHP)

روش های مشخص شده از سوی کمپانی ها هم وجود دارند. برای مثال mysqli extension در MySQL 4.1 و بالاتر:

```
$db = new mysqli("localhost", "user", "pass", "database");

$stmt = $db -> prepare("SELECT priv FROM testUsers WHERE username=? AND
password=?");

$stmt -> bind_param("ss", $user, $pass);

$stmt -> execute();
```

ColdFusion

در ColdFusion جمله ی CFQUERYPARAM در اتصال با جمله ی CFQUERY می تواند برای بی اثر کردن کد SQL فرستاده شده همراه با مقدار CFQUERYPARAM به عنوان قسمتی از عبارت SQL مورد استفاده قرار گیرد.

```
<cfquery name="Recordset1" datasource="cafetownsend">

SELECT *

FROM COMMENTS

WHERE COMMENT_ID =<cfqueryparam value="#URL.COMMENT_ID#"
cfsqltype="cf_sql_numeric">

</cfquery>
```

VB.NET)

```
Dim thisCommand As SQLCommand = New SQLCommand("SELECT Count(*) " & _

 "FROM Users WHERE UserName = @username AND Password = @password",
Connection)
```

www.ITSecTeam.com

ITSecTeam
Security Research & Penetration Testing Team

```
thisCommand.Parameters.Add ("@username", SqlDbType.VarChar).Value =
username

thisCommand.Parameters.Add ("@password", SqlDbType.VarChar).Value =
password

Dim thisCount As Integer = thisCommand.ExecuteScalar()
```

ASP)

```
<%

    option explicit

    dim conn, cmd, recordset, iTableIdValue

    'Create Connection

    set conn=server.createObject("ADODB.Connection")

    conn.open "DNS=LOCAL"

    'Create Command

    set cmd = server.createobject("ADODB.Command")

    With cmd

          .activeconnection=conn

          .commandtext="Select * from DataTable where Id = @Parameter"

          'Create the parameter and set its value to 1

          .Parameters.Append .CreateParameter("@Parameter", adInteger,
adParamInput, , 1)

    End With

    'Get the information in a RecordSet

    set recordset = server.createobject("ADODB.Recordset")

    recordset.Open cmd, conn

    '....

    'Do whatever is needed with the information

    '....

    'Do clean up

    recordset.Close

    conn.Close
```

www.ITSecTeam.com

ITSecTeam
IT Security Research & Penetration Testing Team

```
    set recordset = nothing
    set cmd = nothing

    set conn = nothing

%>
```

C#.net)

```
protected void Page_Load(object sender, EventArgs e)

{

  var connect =
ConfigurationManager.ConnectionStrings["NorthWind"].ToString();

  var query = "Select * From Products Where ProductID = @ProductID";

  using (var conn = new SqlConnection(connect))

  {

    using (var cmd = new SqlCommand(query, conn))

    {

      cmd.Parameters.Add("@ProductID", SqlDbType.Int);

      cmd.Parameters["@ProductID"].Value =
Convert.ToInt32(Request["ProductID"]);

      conn.Open();

      //Process results

    }

  }

}
```

C#.net)

```
protected void Page_Load(object sender, EventArgs e)

{

  var connect =
ConfigurationManager.ConnectionStrings["NorthWind"].ToString();

  var query = "Select * From Products Where ProductID = @ProductID";

  using (var conn = new SqlConnection(connect))
```

www.ITSecTeam.com

ITSecTeam
IT Security Research & Penetration Testing Team

```
  {
    using (var cmd = new SqlCommand(query, conn))

    {

      cmd.Parameters.AddWithValue("@ProductID",
Convert.ToInt32(Request["ProductID"]);

      conn.Open();

      //Process results

    }

  }

}
```

استفاده از فرایند های ذخیره شده (Stored Procedures)

Java)

```
String custname = request.getParameter("customerName "); // This should
REALLY be validated

 try {

      CallableStatement cs = connection.prepareCall("{call
sp_getAccountBalance(?)}");

      cs.setString(1, custname);

      ResultSet results = cs.executeQuery();

      // … result set handling

 } catch (SQLException se) {

      // … logging and error handling

 }
```

vb.net)

```
Try

      Dim command As SqlCommand = new SqlCommand("sp_getAccountBalance",
connection)
```

www.ITSecTeam.com

ITSecTeam
IT Security Research & Penetration Testing Team

```
        command.CommandType = CommandType.StoredProcedure
        command.Parameters.Add(new SqlParameter("@CustomerName",
CustomerName.Text))

        Dim reader As SqlDataReader = command.ExecuteReader()

        ' …

Catch se As SqlException

        ' error handling

End Try
```

VB.net)

```
Dim thisCommand As SQLCommand = New SqlCommand ("proc_CheckLogon",
Connection)

thisCommand.CommandType = CommandType.StoredProcedure

thisCommand.Parameters.Add ("@username", SqlDbType.VarChar).Value =
username

thisCommand.Parameters.Add ("@password", SqlDbType.VarChar).Value =
password

thisCommand.Parameters.Add ("@return", SqlDbType.Int).Direction =
ParameterDirection.ReturnValue

Dim thisCount As Integer = thisCommand.ExecuteScalar()
```

C#.net)

```
var connect =
ConfigurationManager.ConnectionStrings["NorthWind"].ToString();

var query = "GetProductByID";

using (var conn = new SqlConnection(connect))

{

  using (var cmd = new SqlCommand(query, conn))

  {

    cmd.CommandType = CommandType.StoredProcedure;

    cmd.Parameters.Add("@ProductID", SqlDbType.Int).Value =
Convert.ToInt32(Request["ProductID"]);

    conn.Open();
```

www.ITSecTeam.com

ITSecTeam
IT Security Research & Penetration Testing Team

```
    //Process results
  }

}
```

Escape کردن تمام ورودی های اعمال شده توسط کاربر

PHP)

```
$Username=$_POST['uname'];

$Password=$_POST['pass'];

mysql_query("SELECT * FROM Users where UserName='".
mysql_real_escape_string($Username) ."' and Password='".
mysql_real_escape_string($Password)."'")
```

هشدار کد PHP زیر امن نیست:

```
$tablename=$_GET['tb'];

mysql_query("SELECT * FROM " . mysql_real_escape_string($tablename). "
where id=1") ;
```

به طور کلی:

* تا حد ممکن ازاستفاده از نام های جدول به صورت دینامیک خودداری کنید.

* اگر مجبورید از نام جدول به صورت دینامیک استفاده کنید، تا حد ممکن انها از کاربر قبول نکنید.

* اگر مجبورید به کاربر اجازه دهید نام جدول دینامیکی را انتخاب کند، فقط از کاراکتر های مورد قبول لیست سفید برای نام جدول استفاده کنید و حتما برای طول نام جدول محدودیت بگذارید. به طور خاص، سیستم های بانک اطلاعاتی رنج گسترده ای از کاراکتر های غیر قابل قبول دارند که در انتشارات مختلف تغییر می کنند.

PHP)

```
$Username=$_POST['uname'];

$Password=$_POST['pass'];
```

www.ITSecTeam.com

ITSecTeam

```
mysql_query("SELECT * FROM Users where UserName='". addslashes($Username)
."' and Password='". addslashes($Password)."'")
```

هشدار کد PHP زیر ایمن نیست:

```
mysql_query("SELECT * FROM Users where ID=". addslashes($UserId))
```

این کد را می توان اکسپلویت کرد زیرا ورودی به عنوان قسمتی از دستور SQL عمل می کند. کد زیر ایمن می باشد:

```
mysql_query("SELECT * FROM Users where ID='". addslashes($UserId)."'")
```

ASP)

```
v_cat = request("category")

v_cat=replace(v_cat, "'", "''") 'Escaping singe quotes

sqlstr="SELECT * FROM product WHERE PCategory='" & v_cat & "'"

set rs=conn.execute(sqlstr)
```

Aspx)

```
protected void Button1_Click(object sender, EventArgs e)

{

  string connect = "MyConnString";

 string strUname = UserName.Text
 strUname= strUname.Replace("'", "''")
 string strPass = Password.Text
 strPass= strPass.Replace("'", "''")

  string query = "Select Count(*) From Users Where Username = '" + strUname
+ "' And Password = '" + strPass + "'";

  int result = 0;

  using (var conn = new SqlConnection(connect))

  {

    using (var cmd = new SqlCommand(query, conn))

    {

      conn.Open();

      result = (int)cmd.ExecuteScalar();
```

www.ITSecTeam.com

ITSecTeam
IT Security Research & Penetration Testing Team

```
    }
  }

  if (result > 0)

  {

    Response.Redirect("LoggedIn.aspx");

  }

  else

  {

    Literal1.Text = "Invalid credentials";

}
```

forum.ITSecTeam.com

ITSecTeam
IT Security Research & Penetration Testing Team

6-مراجع

http://www.owasp.org/index.php/Testing_for_SQL_Injection_%28OWASP-DV-005%29

http://www.cgisecurity.com/questions/sql.shtml

http://www.webappsec.org/projects/glossary/#SQLInjection

http://unixwiz.net/techtips/sql-injection.html

http://www.owasp.org/index.php/Blind_SQL_Injection

http://www.owasp.org/index.php/Guide_to_SQL_Injection

http://www.mikesdotnetting.com/Article/113/Preventing-SQL-Injection-in-ASP.NET

http://www.owasp.org/index.php/Reviewing_Code_for_SQL_Injection

http://www.nextgenss.com/papers/advanced_sql_injection.pdf

http://www.securiteam.com/securityreviews/5DP0N1P76E.html

http://en.wikipedia.org/wiki/SQL_injection

http://www.owasp.org/index.php/SQL_Injection

http://www.w3schools.com/sql/

http://ferruh.mavituna.com/sql-injection-cheatsheet-oku/

forum.ITSecTeam.com