1.1.1 SQL Enjeksiyonu (SQL Injection) (CWE-89)

Açıklık Önem Derecesi: Kritik

Açıklığın Etkisi: Hassas Bilgilere Yetkisiz Erişim, Veri tabanı Üzerinde Keyfi Sorgu Çalıştırma, Sistem Kabuğunda Keyfi Komut Çalıştırma

Açıklığın Barındıran Dosyalar/Satırlar:

Proje Dosyası/Dosya Adı	Satır Numarası

Açıklığın Açıklaması:

Uygulamalar verdikleri hizmetin şekline ve kapsamına göre kullanıcılardan değerler alabilirler, bu bilgiler doğrultusunda bir takım bilgiler sunabilirler ve bu bilgiler üzerinde kullanıcıların işlem yapmasına olanak sağlayabilirler. Uygulamaların kullanıcılardan değer aldıkları durumlarda bu değerlerin kontrollü bir şekilde uygulama içinde işleme sokulması gerekir. Güvenlik literatüründe yer alan "Kullanıcıdan gelen hiçbir veriye güvenme" prensibi gereği kullanıcılardan gelen veriler kesinlikle girdi denetimine tabi tutularak uygulamada işleme sokulmalıdır. Bu denetimin yapılmaması kötü niyetli kullanıcıların uygulamanın çalışma şeklini bozacak şekilde değerler girebilmesine olanak sağlayabilir. Böyle bir durumda uygulama kötü niyetli kullanıcıya gereğinden fazla bilgi dönebilir.

"SQL" (Structured Query Language) Enjeksiyonu veri tabanı üzerinde işlem yapan uygulamalarda yeterli girdi denetimi yapılmaması sonucu kötü niyetli kullanıcıların veri tabanında çalışacak sorgulara gereğinden fazla bilgi açığa çıkaracak değerler aktarmasına denir. Teknik olarak saldırı şu şekilde gerçekleşir: Uygulama sql sorgusunda basit bir string concatenation (string birleşimi) işlemi ile kullanıcı girdisini sql sorgusuna ekler. Bu şekilde sorgu oluşturulunca kod (code) ve veri (data) arasında bir ayrım olmaz. Bu durumda güvensiz veri (data) normal veri yerine SQL komutları içerdiğinde mevcut sql sorgusunu modifiye edebilir ve veri tabanı bu değiştirilmiş sql sorgusunu çalıştırabilir. Saldırganlar bu açıklığı uygulamada URL'i modifiye ederek, bir formu submit'leyerek (onaylarak) veya http talebindeki diğer girdi alanlarını modifiye ederek sömürebilir.

SQL Enjeksiyonunu daha iyi somutlaştırmak adına Java, C#, ASP, Javascript ve PHP örneklerine yer verilmiştir:

Java - Güvensiz Hal:

```
// Create SQL query using string concatenation

public int getUserId(HttpServletRequest request)
        throws ServletException, IOException {
    int userId = 0;

    String userName = request.getParameter("UserName");
    String sql = "SELECT [UserID] FROM [AppUsers] WHERE [UserName] = '" +
userName + "' " ;

    try {
        Connection conn = getConnection();
        Statement stmt = conn.createStatement();
        ResultSet data = stmt.executeQuery(sql);

        userId = data.getInt(1);
    } catch (SQLException ex) {
        handleExceptions(ex);
    }
    finally {
        closeQuietly(data);
        closeQuietly(stmt);
        closeQuietly(conn);
    }

    return userId;
}
```

Java - Güvenli Hal (I):

```
// Create SQL query using Sanitized Username

public int getUserId(HttpServletRequest request)
        throws ServletException, IOException {
    int userId = 0;

    String userName = request.getParameter("UserName");

    // Sanitize input using OWASP's ESAPI Encoder library
    // Still not complete solution!
    Encoder esapiEncoder = new DefaultEncoder();
    String sanitizedUserName = esapiEncoder.encodeForSQL(new OracleCodec(),
userName);

    String sql = "SELECT [UserID] FROM [AppUsers] WHERE [UserName] = '" +
sanitizedUserName + "' " ;

    try {
        Connection conn = getConnection();
        Statement stmt = conn.createStatement();
        ResultSet data = stmt.executeQuery(sql);

        userId = data.getInt(1);
    } catch (SQLException ex) {
        handleExceptions(ex);
    }
    finally {
        closeQuietly(data);
        closeQuietly(stmt);
        closeQuietly(conn);
    }

    return userId;
}
```

Java - Güvenli Hal (II):

```
// Build PreparedStatement to call Stored Procedure and set input to
parameters

public int getUserId(HttpServletRequest request)
        throws ServletException, IOException {
    int userId = 0;

    String userName = request.getParameter("UserName");
    String sqlStoredProc = "{call getUserId (?, ?)}";

    try {
        Connection conn = getConnection();
        CallableStatement stmt = conn.prepareCall(sqlStoredProc);

        stmt.setString(1, userName);
        stmt.registerOutParameter(2, java.sql.Types.INTEGER);

        stmt.execute();
        userId = stmt.getInt(2);
    } catch (SQLException ex) {
        handleExceptions(ex);
    }
    finally {
        closeQuietly(stmt);
        closeQuietly(conn);
    }

    return userId;
}
```

Kod bloklarını yorumlayacak olursak “Java - Güvensiz Hal” kod bloğunda GET userName parametresi, yani kullanıcı girdisi sql sorgusuna hiçbir denetim uygulanmadan eklenmektedir. Bu durumda sql enjeksiyonu açıklığı meydana gelecektir. “Java - Güvenli Hal (I)” kod bloğunda ise GET userName parametresi, yani kullanıcı girdisi sql sorgusuna eklenmeden önce OWASP’ın ESAPI encoder (kodlayıcı) düzenleyici aşamasından geçmektedir. Bu düzenleme aşamasında GET userName parametresindeki, yani kullanıcı girdisindeki sql sorgularında özel anlam ifade edecek anahtar karakterler düzenlenmektedir ve etkisiz hale getirilmektedir. Bu durumda sql enjeksiyonu açıklığı büyük ölçüde giderilmiş olacaktır. Ancak en güvenli yöntem “Java - Güvenli hal (II)” kod bloğunda gösterildiği gibi “**parametrik sorgu**” kullanmaktır.

C# - Güvensiz Hal:

```
// Create SQL query using string concatenation

private int GetUserId_Unsafe(HttpRequest request)
{
    int userId = 0;

    string userName = request.Form["UserName"];
    string sql = "SELECT [UserID] FROM [AppUsers] WHERE [UserName] = '" +
userName + "' " ;

    using (SqlConnection conn = GetConnection())
    {
        using (SqlCommand command = new SqlCommand(sql, conn))
        {
            using (SqlDataReader reader = command.ExecuteReader())
            {
                reader.Read();
                userId = reader.GetInt32(0);
            }
        }
    }

    return userId;
}
```

C# - Güvenli Hal:

```
// Set DbCommand to call Stored Procedure and set input to DbParameters

private int GetUserId_SafeStoredProcWithParameters(HttpRequest request)
{
    int userId = 0;

    string userName = request.Form["UserName"];
    string spName = "GetUserId";

    using (SqlConnection conn = GetConnection())
    {
        using (SqlCommand command = new SqlCommand(spName, conn))
        {
            command.CommandType = CommandType.StoredProcedure;
            command.Parameters.AddWithValue("@userName", userName);

            using (SqlDataReader reader = command.ExecuteReader())
            {
                reader.Read();
                userId = reader.GetInt32(0);
            }
        }
    }

    return userId;
}
```

Kod bloklarını yorumlayacak olursak “C# - Güvensiz Hal” kod bloğunda userName parametresi, yani kullanıcı girdisi sql sorgusuna hiçbir denetim uygulanmadan eklenmektedir. Bu durumda sql enjeksiyonu açıklığı meydana gelecektir. “C# - Güvenli Hal” kod bloğunda ise “**parametrik sorgu**” kullanıldığından kullanıcısı girdisi güvenle sorguya eklenecektir ve açıklık meydana gelmeyecektir.

ASP - Güvensiz Hal:

```
' Create SQL query using string concatenation

<%

    Dim userId = 0
    Dim oConn, oRec
    Dim strUserName, sql

    strUserName = Request.Form("txtUsername")
    sql = "SELECT [UserID] FROM [AppUsers] WHERE [UserName] = '" & strUserName
& "' "

    On Error Resume Next

    Set oConn = Server.CreateObject("ADODB.Connection")
    oConn.Open CONN_STRING

    Set oRec = oConn.Execute(sql)

    userId = oRec("UserId")

    If Not IsNothing(oRec) Then oRec.Close
    Set oRec = Nothing
    If Not IsNothing(oConn) Then oConn.Close
    Set oConn = Nothing

    Response.Write userId
%>
```

ASP - Güvenli Hal (I):

```
' Create SQL query using Sanitized UserName

<%
    Dim userId = 0
    Dim oConn, oRec
    Dim strUserName, sql

    strUserName = Request.Form("txtUsername")

    '' Sanitize input using custom sanitizer
    '' Still not complete solution!

    Dim strSanitizedUserName
    strSanitizedUserName = SanitizeForSql(strUserName)

    sql = "SELECT [UserID] FROM [AppUsers] WHERE [UserName] = '" &
strSanitizedUserName & "' "

    On Error Resume Next

    Set oConn = Server.CreateObject("ADODB.Connection")
    oConn.Open CONN_STRING

    Set oRec = oConn.Execute(sql)

    userId = oRec("UserId")

    If Not IsNothing(oRec) Then oRec.Close
    Set oRec = Nothing
    If Not IsNothing(oConn) Then oConn.Close
    Set oConn = Nothing

    Response.Write userId
%>
```

ASP - Güvenli Hal (II):

```
' Use Command / Parameters to call Stored Procedure

<%
    Dim userId = 0
    Dim oConn, oRec
    Dim oCmd
    Dim spName = "GetUserId"

    Dim strUserName
    strUserName = Request.Form("txtUsername")

    On Error Resume Next

    Set oConn = Server.CreateObject("ADODB.Connection")
    oConn.Open CONN_STRING

    Set oCmd = Server.CreateObject("ADODB.Command")
    With oCmd
        Set .ActiveConnection = oConn
        .CommandType = adCmdStoredProc
        .CommandText = spName

        .Parameters.Append .CreateParameter("UserName", adVarChar,
adParamInput, 50, strUserName)
    End With

    Set oRec = oCmd.Execute
    userId = oRec("UserId")

    If Not IsNothing(oRec) Then oRec.Close
    Set oRec = Nothing
    If Not IsNothing(oConn) Then oConn.Close
    Set oConn = Nothing
    Set oCmd = Nothing

    Response.Write userId
%>
```

Kod bloklarını yorumlayacak olursak “ASP - Güvensiz Hal” kod bloğunda txtUsername parametresi, yani kullanıcı girdisi sql sorgusuna hiçbir denetim uygulanmadan eklenmektedir. Bu durumda sql enjeksiyonu açıklığı meydana gelecektir. “ASP - Güvenli Hal (I)” kod bloğunda ise txtUsername parametresi, yani kullanıcı girdisi sql sorgusuna eklenmeden önce SanitizeForSQL() metodu ile bir filtreleme işlemine tabi tutulmaktadır. Bu filtreleme aşamasında txtUsername parametresindeki, yani kullanıcı girdisindeki sql sorgularında özel anlam ifade edecek anahtar karakterler ayıklanmaktadır. Bu durumda sql enjeksiyonu

açıklığı büyük ölçüde giderilmiş olacaktır. Ancak en güvenli yöntem “ASP- Güvenli hal (II)” kod bloğunda gösterildiği gibi “**parametrik sorgu**” kullanmaktır.

Javascript - Güvensiz Hal:

```
// Concatenated SQL Query Leads to SQL Injection in Client Code

db.transaction(
    function(tx) {
        var url = new URL(window.location.href);
        var name = url.searchParams.get("name");
        var result = tx.executeSql('SELECT * FROM user WHERE name=' + name);
        // Handle result
    }
);
```

Javascript - Güvenli Hal:

```
// Use of a Parameterized Query to Avoid a SQL Injection

db.transaction(
    function(tx) {
        var url = new URL(window.location.href);
        var name = url.searchParams.get("name");
        var result = tx.executeSql('SELECT * FROM user WHERE name=?', [name]);
        // Handle result
    }
);
```

Kod bloklarını yorumlayacak olursak “Javascript - Güvensiz Hal” kod bloğunda name parametresi, yani kullanıcı girdisi sql sorgusuna hiçbir denetim uygulanmadan eklenmektedir. Bu durumda sql enjeksiyonu açıklığı meydana gelecektir. “Javascript - Güvenli Hal” kod bloğunda ise “**parametrik sorgu**” kullanıldığından kullanıcısı girdisi güvenle sorguya eklenecektir ve açıklık meydana gelmeyecektir.

PHP - Güvensiz Hal:

```php
<?php

    // ...

    $id = $_COOKIE["mid"];
    mysql_query("SELECT MessageID, Subject FROM messages WHERE MessageID =
'$id'");

    // ...

?>
```

PHP - Güvenli Hal:

```php
<?php

    // ...

    $id = intval($_COOKIE["mid"]);
    mysql_query("SELECT MessageID, Subject FROM messages WHERE MessageID =
'$id'");

    // ...

?>
```

Kod bloklarını yorumlayacak olursak “PHP - Güvensiz Hal” kod bloğunda mid çerez parametresi, yani kullanıcı girdisi sql sorgusuna hiçbir denetim uygulanmadan eklenmektedir. Bu durumda sql enjeksiyonu açıklığı meydana gelecektir. “PHP - Güvenli Hal” kod bloğunda ise intval() fonksiyonu ile girdi doğrulama yapılmaktadır. Girdi doğrulaması sonrası mid çerez parameretresi sql sorgusuna yerleştirilmektedir. Bu sayede sql enjeksiyonu açıklığı büyük ölçüde giderilmiş olacaktır. Ancak en doğru sql enjeksiyonu açıklığını kapama yöntemi “**parametrik sorgu**” kullanmaktır.

Kurum uygulamada “SQL Enjeksiyonu (CWE-89)” açıklığı tespit edilmiştir. Bu durum Şekil XXX. ABCDEF’de gösterilmiştir.

::::::::BULGU::::::::::

Açıklığın Önlemi:

SQL Enjeksiyonu açıklığı şu esas önlemler neticesinde önlenebilir:

- Kaynağına bakmaksızın tüm istemci girdileri girdi denetimine tabi tutulmalıdır. Bu denetimde
    - Girdinin veri tipi
    - Girdinin karakter uzunluğu
    - Girdide beklenen değer kıyaslaması (yani “whitelist önlemi”)

  kriterleri kontrol edilmelidir.

- En az ayrıcalık prensibi (principle of least privilege) gereği veri tabanı nesnelerine ve fonksiyonlarına erişimler kısıtlı olmalıdır. Örn; uygulamada mümkün olduğunca düşük yetkili veri tabanı kullanıcısı ile sorguların çalıştırılması sağlanmalıdır.
- SQL sorguları oluşturmak için dinamik olarak string’leri birleştirme (dynamically string concatenation) kullanılmamalıdır.
- Güvensiz dinamik olarak string birleştirme (dynamically string concatenation) kullanma yerine parameterize sorgular kullanılmalıdır.

İlaveten sql enjeksiyonu açıklığı şu ekstra önlemlerle de önlenebilir:

- WAF kullanılması
- IDS/IPS kullanılması

Daima esas önlemler uygulanmalıdır, fakat mevcut güvenliğin seviyesini arttırmak için bahsedilen ekstra önlemler de güvenlik mimarisine eklenebilir.

Uyarı:

Sadece WAF/IDS/IPS kullanmak ve arkadaki uygulamada yer alan açıklığı kaynak kodda önlememek gelecekte WAF/IDS/IPS ürününde çıkabilecek bir açıklık neticesinde arkadaki uygulamada yer alan sql enjeksiyonu açıklığının sömürülebilmesine imkan tanıyabilir. Güvenlik literatüründe yer alan derinlikli defans (defense-in-depth) prensibi gereği her katmanda güvenlik önlemleri alınması gerekir. Sadece bir katmana bel bağlamak sonradan güvenlik ihlallerine yol açabilir.

Referanslar:

1. https://cwe.mitre.org/data/definitions/89.html
2. https://stackoverflow.com/questions/16993225/datagrid-get-cell-contents
3.