Top 10 Defenses for Website Security

Jim Manico
VP of Security Architecture

Jim.Manico@whitehatsec.com

July 2012
Jim Manico  @manicode

- 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
- Running for the OWASP Board 2013
- OWASP Connections Committee Chair
  - OWASP Podcast Series Producer/Host
  - OWASP Cheat-Sheet Series Manager
$stmt = $dbh->prepare("INSERT INTO REGISTRY (name, value) VALUES (:name, :value)";

$stmt->bindParam(' :name', $name);
$stmt->bindParam(' :value', $value);
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 (Ruby)

# 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 *Fail* (Ruby)

# 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>
Query Parameterization (PERL)

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;
}
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(bytes);
        }
        String encoded = ESAPI.encoder().encodeForBase64(bytes, false);
        return encoded;
    } catch (Exception ex) {
        throw new EncryptionException("Internal error", "Error");
    }
}
Secure Password Storage

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(bytes);
        }
        String encoded = ESAPI.encoder().encodeForBase64(bytes,false);
        return encoded;
    } catch (Exception ex) { 
        throw new EncryptionException("Internal error", "Error");
    }
}
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");
    }
}
Secure Password Storage

• BCRYPT
  - *Really slow on purpose*
  - *Blowfish derived*
  - *Suppose you are supporting millions on concurrent logins…*
  - *Takes about 10 concurrent runs of BCRYPT to pin a high performance CPU*

• PBKDF2
  - *Takes up a lot of memory*
  - *Suppose you are supporting millions on concurrent logins…*
Data Sanitization (Stop XSS)

- 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, nosize, noshade, nowrap, ref, rel, rev, rows, colspan, scrolling, shape, span, summary, tabindex, title, usemap, valign, value, vlink, vspace, width
<span>UNTRUSTED DATA</span>
<input type="text" name="fname" value="UNTRUSTED DATA">
HTTP GET Parameter Context

<a href="/site/search?value=UNTRUSTED DATA">clickme</a>
URL Context

<a href="UNTRUSTED URL">clickme</a>

<iframe src="UNTRUSTED URL"/>
CSS Value Context

<div style="width: UNTRUSTED DATA;">Selection</div>
JavaScript Variable Context

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

<script>
someFunction('UNTRUSTED DATA');
</script>
```
<table>
<thead>
<tr>
<th>Dangerous jQuery 1.7.2 Data Types</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CSS</strong></td>
</tr>
<tr>
<td><strong>HTML</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>jQuery methods that directly update DOM or can execute JavaScript</th>
</tr>
</thead>
<tbody>
<tr>
<td>$() or jQuery()</td>
</tr>
<tr>
<td>.add()</td>
</tr>
<tr>
<td>.after()</td>
</tr>
<tr>
<td>.animate()</td>
</tr>
<tr>
<td>.append()</td>
</tr>
<tr>
<td>.appendTo()</td>
</tr>
</tbody>
</table>

Note: .text() updates DOM, but is safe.

<table>
<thead>
<tr>
<th>jQuery methods that accept URLs to potentially unsafe content</th>
</tr>
</thead>
<tbody>
<tr>
<td>jQuery.ajax()</td>
</tr>
<tr>
<td>jQuery.get()</td>
</tr>
<tr>
<td>jQuery.getScript()</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);`
Best Practice: 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’));`
Permission Based Access Control

• Code to the permission, not the role
• Centralize access control logic
• Design access control as a filter
• Fail securely (deny-by-default)
• Apply same core logic to presentation and server-side access control decisions
• Server-side trusted data should drive access control
• Provide privilege and user grouping for better management
• Isolate administrative features and access
Best Practice: Code to the Permission

```java
if (AC.hasAccess(ARTICLE_EDIT, NUM)) {
    //execute activity
}
```

- Code it once, and it never needs to change again
- Implies policy is persisted in some way
- Requires more design/work up front to get right
OWASP Access Control
Cheat Sheet
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*
OWASP Cross-Site Request Forgery Cheat Sheet
Multi Factor Authentication

• Passwords as a single AuthN 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

• Require identity and security questions
  - Last name, account number, email, DOB
  - Enforce lockout policy
  - Ask one or more good security questions
    - http://www.goodsecurityquestions.com/

• Send the user a randomly generated token via out-of-band method
  - email, SMS or token

• Verify code in same Web session
  - Enforce lockout policy

• Change password
  - Enforce password policy
Session Defenses

• Ensure secure session IDs
  - 20+ bytes, cryptographically random
  - Stored in HTTP Cookies
  - Cookies: Secure, HTTP Only, limited path

• Generate new session ID at login time
  - To avoid session fixation

• Session Timeout
  - Idle Timeout
  - Absolute Timeout
  - Logout Functionality
OWASP Session Management Cheat Sheet
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" );
Encryption in Transit (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
  - All other sensitive data should be protected via HTTPS!

- [https://www.ssllabs.com](https://www.ssllabs.com) free online assessment of public-facing server HTTPS configuration

OWASP Transport Layer Protection Cheat Sheet
Thank You

jim@owasp.org