Addressing Web Application Security

Gil Muñoz and Jeremy Tarpley
Major Attack Vectors
The Open Web Application Security Project (OWASP) Top 10

1. Injection
2. Broken Authentication and Session Management
3. Cross-Site Scripting (XSS)
4. Insecure Direct Object References
5. Security Misconfiguration
6. Sensitive Data Exposure
7. Missing Function Level Access Control
8. Cross-Site Request Forgery (CSRF)
9. Using Known Vulnerable Components
10. Unvalidated Redirects and Forwards
Injection

Attacker executes their code through your web application, running commands or access data in unintended ways.
Injection example

**Code for a login form:**
$\textit{db} = \text{new mysqli('localhost', 'root', 'passwd', 'base');}$
$\textit{result} = \textit{db}->\text{query('SELECT * FROM users WHERE user="".$\text{_GET['user']}$."" AND pass="".$\text{_GET['password']}$.""');}$

Attacker submission:

![Username](image)

New query:
`SELECT * FROM users WHERE user="" OR 1 -- AND pass="whatever"`
Broken Authentication, and Session Management

Attackers compromise passwords, session tokens or otherwise exploit an application to assume other’s identities or permissions.
Example:

Session Hijacking
Cross-Site Scripting (XSS)

Attacker adds their code to your web page. The code is then executed in a user’s browser when the page is visited.
XSS Example

Page allows you to select a language when you visit
http://yoursite.edu/page.html?default=French

But attacker sends a deceptive email to your users with this link:

```html
<a href="http://yoursite.edu/page.html?default=<script>alert(document.cookie)</script>">
  Click here to update your settings!
</a>
```
Direct object reference

Accessing files, info from your db, or other resources that shouldn’t be exposed
Direct object reference examples

http://example.com/application?file=1

http://example.com/users/profiles/../../../etc/passwd

http://mywordpresssite.edu/wp-config.php.bak
Security Misconfiguration

Not configuring or misconfiguring your application, server, etc., resulting in exposing a vulnerability.
Defense Against the Dark Arts
Basics for web app security

- Require SSL everywhere
- Sanitize input AND output, constrain variables
- Protect sensitive information
- Exercise least privilege & defense in depth
- Have a security prone mindset and master the basics – ex: patch / update everything regularly
Basic Defense

1. Require SSL

Set your Web Server to Require SSL
- IIS Manager
  - Features View
    - SSL Settings
      - Require SSL
Basic Defense

1. Require SSL

Web.config transforms – add permanent redirect

```xml
<system.webServer xdt:Transform="Insert">
  <rewrite>
    <rules>
      <rule name="HTTP to HTTPS redirect" stopProcessing="true">
        <match url="(.*)" />
        <conditions>
          <add input="{HTTPS}" pattern="off" />
        </conditions>
        <action type="Redirect"
          url="https://{HTTP_HOST}/{R:1}"
          redirectType="Permanent" />
      </rule>
    </rules>
  </rewrite>
</system.webServer>
```
Basic Defense

2. Sanitize input & output

According to OWASP

“The most common web application security weakness is the failure to properly validate input from the client or environment.”
Basic Defense

3. Protect Sensitive Information
   a) Exercise Least Privilege
   b) Encrypt all sensitive information.
   c) Remove content rather than hiding it.
   d) Use generic error page – no details.
Basic Defense

4. Build in-depth security

Across Every Module & Down Every Layer

Defense Against the Dark Arts
Basic Defense

5. Have a security prone mindset
   a) Be a bit paranoid.
   b) Be open to input, scrutiny.
   c) Be a learner.
Vulnerability Mitigation

Examples in ASP.NET and PHP
Script and SQL Injection (ASP.NET)

a) Use ASP.NET Request Validation

```csharp
void Application_Error(object sender, EventArgs e)
{
    Exception exc = Server.GetLastError();
    if (exc is HttpRequestValidationException) {
    } else if (exc is HttpUnhandledException) {
    }
}
```
Script and SQL Injection (ASP.NET)

a) Use ASP.NET Request Validation
b) Constrain Input
c) Encode Output
d) Separate DB access to service or API with type-safe parameters
e) Parameterize your queries
Script and SQL Injection (ASP.NET)

An Example of what not to do.

SELECT id, name
FROM city

BldgList = GetBuildings (CityList.SelectedValue)

SelCmd = "SELECT id, name FROM building WHERE city_id = " + CityId;

Vulnerability Mitigation
Script and SQL Injection (ASP.NET)

UI
HTML, JS...

Business
C#, Java, PHP...

Data
Web Service, DB

Vulnerability Mitigation
What to do in your data access layer

1. Service or API with type-safe parameters.
   ```csharp
   [OperationContract]
   List<Building> GetBuildings(Int64 cityId);
   ```

2. Parameterize your query
   ```csharp
   SelCmdTxt = "SELECT id, name FROM building
                  WHERE city_id = @CityId"
   SelCmd = new MySqlCommand(SelCmdTxt, conn);
   SelCmd.AddParameter("@CityId", cityId)
   ```
1) Sanitize all forms of input -
   a) Know and use input / output sanitization functions provided by php: `filter_input`, `filter_var` and other filter functions
   b) Know and use input / output controls provided by your framework or CMS. Ex `wp_kses`
Script and SQL Injection (PHP)

Validate, and Sanitize all forms of input

Know and use input / output sanitization functions provided by PHP. For example:

- `filter_input`, `filter_var`,
- `htmlspecialchars`()
- `htmlentities`()
- `strip_tags`()
- `urlencode`()
- `json_encode`()
- `mysqli_real_escape_string`()
- `addslashes`()
Script and SQL Injection (PHP)

filter_input example

```php
<?php
$sanitized_email = filter_input(INPUT_GET, 'email', FILTER_SANITIZE_EMAIL);
echo "The email address you entered is $sanitized_email.\n";

$sanitized_search_html = filter_input(INPUT_GET, 'search', FILTER_SANITIZE_SPECIAL_CHARS);
$sanitized_search_url = filter_input(INPUT_GET, 'search', FILTER_SANITIZE_ENCODED);
echo "You have searched for $sanitized_search_html.\n";
echo "<a href=?search=$sanitized_search_url>Search again.</a>";
?>
```
Script and SQL Injection (PHP)

CMS, framework specific sanitization

Know and use input / output controls provided by your framework or CMS. Ex wp_kses for WordPress.

```php
<?php
$allowed_tags = array(
    'strong' => array(),
    'a' => array(
        'href' => array()
    )
);

// pretend we have a blog comment we want to display
$result_from_query_from_wp = '<a href="#" class="external">link</a>.
This is <b>bold</b> and <strong>strong</strong>';

$sanitized_result = wp_kses( $result_from_query_from_wp, $allowed_tags );

echo $sanitized_result;
?>

Output:
<a href="#">link</a>. This is bold and <strong>strong</strong>
```
Script and SQL Injection (PHP)

- Validate input
- Escape output (data written to the database)
- Use prepared statements
- Use least privileged accounts
<?php
$link = mysqli_connect('db.srv', 'unique_user', 'super_long_password', 'app_specific_db');
mysqli_query($link, "CREATE TABLE cityInfo (cityName VARCHAR(50))");

// notice the single quote before the S.
$cityName = "St. John's, Newfoundland";

/* this query will fail, bc we didn't escape $cityName */
if (!mysqli_query($link, "INSERT INTO cityInfo (cityName) VALUES ('$cityName')") ) {
    // write to you log file
}

$safeForDbCityName = mysqli_real_escape_string($link, $cityName);

/* this query with escaped city name will work */
if (mysqli_query($link, "INSERT INTO cityInfo (cityName) VALUES ('$safeForDbCityName')") ) {
    // write to you log file ("%d Row inserted.\n", mysqli_affected_rows($link));
}

$stmt = $pdo->prepare('SELECT cityName FROM cityInfo ');
$stmt->execute();
$row = $stmt->fetch(PDO::FETCH_ASSOC);

$safeForOutputCityName = htmlspecialchars($row['cityName']);

<p>City selected:, <?php echo $safeForOutputCityName ?></p>

Vulnerability Mitigation
Script and SQL Injection prevention with prepared statements

```php
<?php
$dBConnection = new PDO('mysql:dbname=app_specific_db;host=db.srv.edu;charset=utf8', 'user', 'pass');
$dBConnection->setAttribute(PDO::ATTR_EMULATE_PREPARES, false);

// old way: mysqli_query($link, "INSERT INTO cityInfo (cityName) VALUES ('$cityName')"); */
// new PDO way
 generado = $dBConnection->prepare("INSERT INTO cityInfo (cityName) VALUES (:cityName)" );
 generado->bindParam(':cityName', $cityName);

// the parameter values are combined with the compiled statement,
// not an SQL string so we don't need to escape */
$cityName = $_POST['cityName'];

// insert row
 generado->execute();
?>
```
Script and SQL Injection

Separate database access to service or api.
Build In-Depth Security
Check authentication and authorization on every layer on every module
Build In-Depth Security

Base Page Class

```csharp
public class MyBasePage : System.Web.UI.Page
{
    public void PageIsAdminOnly()
    {
        if (!CurrentUser.role.isAdmin)
        {
            Server.Transfer("~/NoPageAccess");
        }
    }
}
```

On Each Page’s Code – Assert security

```csharp
public partial class Users : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
        PageIsAdminOnly();
    }
}
```
Authentication and Authorization

Protect User Credentials

a) “The best password is one you don’t have to store”.
b) Enforce strong password requirements
c) Store password securely
d) Use generic “invalid login” message
e) Limit login attempts – use timeout
Authentication and Authorization

Understand and Use Identity Protection in the Page Life Cycle

```csharp
public class MyBasePage : System.Web.UI.Page
{
    ...

    protected override void OnInit(EventArgs e)
    {
        base.OnInit(e);
        ViewStateUserKey = Session.SessionID;
    }
}
```
Authentication and Authorization

Understand and Use Identity Protection in the Page Life Cycle

a) Set ViewStateUserKey to SessionId
b) Use additional user token.
c) Use Base Page methods to check credentials – Authentication and Authorization.
Deployment
1. Separate TEST and PRODUCTION servers
2. Do not deploy test accounts to production servers.
3. Have a security audit of the Web application before deployment
   a) Not done by anyone involved in development
   b) Scan with vulnerability scanner application (ex: Nexpose)
   c) Manual penetration testing
   d) Code review.
General code best practices

Separate code files into public and private directories
• Public directory for files that can be displayed by the browser. Set document root for this folder.
• Private directory for files that should never be seen by a user. Configure webserver to disallow browser access to this folder.

Label variables
Label sanitized variables.
ex:
$safe_url_string = urlencode($url_string);
$safe_name = addslashes($name);
Deployment - WAF

- HTTP requests
- Filtered HTTP requests
- Filtered HTTP responses
- HTTP responses

Web Application Firewall

Web Server
Deployment

Web Server hardening
Configuration hardening
Monitoring and auditing
Regular patching
If we’re out of time, let’s meet again during cocktail hour.