Cross-Site Request Forgery (CSRF) Development Mitigation SOP

# Development Mitigation SOP

A CSRF vulnerability occurs when:

1. A Web application uses session cookies.
2. The application acts on an HTTP request without verifying that the request was made with the user's consent.

## Defense Against Cross-Site Request Forgery

The solution for Cross-Site Request Forgery is the use of a cryptographic nonce. A nonce is a cryptographic random value that is sent with a message to prevent replay attacks. If the request does not contain a nonce that proves its validity, the code that handles the request is vulnerable to a CSRF attack (unless it does not change the state of the application). This means a Web application that uses session cookies has to take special precautions in order to ensure that an attacker can't trick users into submitting bogus requests.

# Examples

## Example

<c:if test="${generateView eq 'true'}">

<form>

<p>

\* Did the claim get submitted via a 21-526EZ form, where the form version is dated &nbsp;&nbsp;&nbsp;<b>August 2011</b> or more recently?

…

</form>

## Explanation

The form does not utilize a nonce to identify itself to the server as a valid request upon submission. The corresponding server-side code apparently did not require that the CSRF token be submitted with the request either.

The solution:

<c:if test="${generateView eq 'true'}">  
 <form>  
 <input type="hidden" id="OWASP\_CSRFTOKEN" name="OWASP\_CSRFTOKEN" value="${sessionScope.OWASP\_CSRFTOKEN}"/>  
 <p>  
 \* Did the claim get submitted via a 21-526EZ form, where the form version is dated &nbsp;&nbsp;&nbsp;<b>August 2011</b> or more recently?

…

</form>

The hidden input field was added to the form. It retrieves the CSRF token value from a session scoped variable named OWASP\_CSRFTOKEN.

# Recommendation

Applications that use session cookies must include some piece of information in every form post that the back-end code can use to validate the provenance of the request. One way to do that is to include a random request identifier or nonce. The back-end logic then can validate the request identifier before processing the rest of the form data. When possible, the request identifier should be unique to each server request rather than shared across every request for a particular session. As with session identifiers, the harder it is for an attacker to guess the request identifier, the harder it is to conduct a successful CSRF attack. The token should not be easily guessed and it should be protected in the same way that session tokens are protected.

Framework protection**:** Most modern web application frameworks embed CSRF protection and they will automatically include and verify CSRF tokens.

Use a Challenge-Response control**:** Forcing the customer to respond to a challenge sent by the server is a strong defense against CSRF. Some of the challenges that can be used for this purpose are: CAPTCHAs, password re-authentication and one-time tokens.

Check HTTP Referer/Origin headers**:** An attacker will not be able to spoof these headers while performing a CSRF attack. This makes these headers a useful method to prevent CSRF attacks.

Double-submit Session Cookie**:** Sending the session ID Cookie as a hidden form value in addition to the actual session ID Cookie is a good protection against CSRF attacks. The server will check both values and make sure they are identical before processing the rest of the form data. If an attacker submits a form on behalf of a user, he won't be able to modify the session ID cookie value as per the same-origin-policy.

Limit Session Lifetime**:** When accessing protected resources using a CSRF attack, the attack will only be valid as long as the session ID sent as part of the attack is still valid on the server. Limiting the Session lifetime will reduce the probability of a successful attack.

The techniques described here can be defeated with XSS attacks. Effective CSRF mitigation includes XSS mitigation techniques.

# Tips:

* Fortify Static Code Analyzer flags all HTML forms and XMLHttpRequest objects that might perform a POST operation. The auditor must determine if each form could be valuable to an attacker as a CSRF target and whether or not an appropriate mitigation technique is in place.

# References

1. [OWASP, 2007 OWASP Top 10](http://www.owasp.org/index.php/Top_10_2007)
2. [Standards Mapping - Common Weakness Enumeration - (CWE), CWE ID 352](http://cwe.mitre.org/data/definitions/352.html)
3. [Standards Mapping - NIST Special Publication 800-53 Revision 4 - (NIST SP 800-53 Rev.4), SC-23 Session Authenticity (P1)](VBMS%20IA%20Documentation/Developer%20Wiki%20SOP/Drafts/csrc.nist.gov/publications/drafts/800-53-rev4/sp800-53-rev4-ipd.pdf)