# CWE Detail – CWE-352

## Description

The web application does not, or cannot, sufficiently verify whether a request was intentionally provided by the user who sent the request, which could have originated from an unauthorized actor.

## Extended Description

N/A

## Threat-Mapped Scoring

Score: 0.0

Priority: Unclassified

## Observed Examples (CVEs)

**•** CVE-2004-1703: Add user accounts via a URL in an img tag

**•** CVE-2004-1995: Add user accounts via a URL in an img tag

**•** CVE-2004-1967: Arbitrary code execution by specifying the code in a crafted img tag or URL

**•** CVE-2004-1842: Gain administrative privileges via a URL in an img tag

**•** CVE-2005-1947: Delete a victim's information via a URL or an img tag

**•** CVE-2005-2059: Change another user's settings via a URL or an img tag

**•** CVE-2005-1674: Perform actions as administrator via a URL or an img tag

**•** CVE-2009-3520: modify password for the administrator

**•** CVE-2009-3022: CMS allows modification of configuration via CSRF attack against the administrator

**•** CVE-2009-3759: web interface allows password changes or stopping a virtual machine via CSRF

## Related Attack Patterns (CAPEC)

* CAPEC-111
* CAPEC-462
* CAPEC-467
* CAPEC-62

## Modes of Introduction

**•** Architecture and Design: REALIZATION: This weakness is caused during implementation of an architectural security tactic.

## Common Consequences

**•** Impact: Gain Privileges or Assume Identity, Bypass Protection Mechanism, Read Application Data, Modify Application Data, DoS: Crash, Exit, or Restart — Notes: The consequences will vary depending on the nature of the functionality that is vulnerable to CSRF. An attacker could trick a client into making an unintentional request to the web server via a URL, image load, XMLHttpRequest, etc., which would then be treated as an authentic request from the client - effectively performing any operations as the victim, leading to an exposure of data, unintended code execution, etc. If the victim is an administrator or privileged user, the consequences may include obtaining complete control over the web application - deleting or stealing data, uninstalling the product, or using it to launch other attacks against all of the product's users. Because the attacker has the identity of the victim, the scope of CSRF is limited only by the victim's privileges.

## Potential Mitigations

**•** Architecture and Design: Use a vetted library or framework that does not allow this weakness to occur or provides constructs that make this weakness easier to avoid. For example, use anti-CSRF packages such as the OWASP CSRFGuard. [REF-330] Another example is the ESAPI Session Management control, which includes a component for CSRF. [REF-45] (Effectiveness: N/A)

**•** Implementation: Ensure that the application is free of cross-site scripting issues (CWE-79), because most CSRF defenses can be bypassed using attacker-controlled script. (Effectiveness: N/A)

**•** Architecture and Design: Generate a unique nonce for each form, place the nonce into the form, and verify the nonce upon receipt of the form. Be sure that the nonce is not predictable (CWE-330). [REF-332] (Effectiveness: N/A)

**•** Architecture and Design: Identify especially dangerous operations. When the user performs a dangerous operation, send a separate confirmation request to ensure that the user intended to perform that operation. (Effectiveness: N/A)

**•** Architecture and Design: Use the "double-submitted cookie" method as described by Felten and Zeller: When a user visits a site, the site should generate a pseudorandom value and set it as a cookie on the user's machine. The site should require every form submission to include this value as a form value and also as a cookie value. When a POST request is sent to the site, the request should only be considered valid if the form value and the cookie value are the same. Because of the same-origin policy, an attacker cannot read or modify the value stored in the cookie. To successfully submit a form on behalf of the user, the attacker would have to correctly guess the pseudorandom value. If the pseudorandom value is cryptographically strong, this will be prohibitively difficult. This technique requires Javascript, so it may not work for browsers that have Javascript disabled. [REF-331] (Effectiveness: N/A)

**•** Architecture and Design: Do not use the GET method for any request that triggers a state change. (Effectiveness: N/A)

**•** Implementation: Check the HTTP Referer header to see if the request originated from an expected page. This could break legitimate functionality, because users or proxies may have disabled sending the Referer for privacy reasons. (Effectiveness: N/A)

## Applicable Platforms

**•** None (Class: Not Language-Specific, Prevalence: Undetermined)

## Demonstrative Examples

**•** The following HTML is intended to allow a user to update a profile.

## Notes

**•** Relationship: There can be a close relationship between XSS and CSRF (CWE-352). An attacker might use CSRF in order to trick the victim into submitting requests to the server in which the requests contain an XSS payload. A well-known example of this was the Samy worm on MySpace [REF-956]. The worm used XSS to insert malicious HTML sequences into a user's profile and add the attacker as a MySpace friend. MySpace friends of that victim would then execute the payload to modify their own profiles, causing the worm to propagate exponentially. Since the victims did not intentionally insert the malicious script themselves, CSRF was a root cause.

**•** Theoretical: The CSRF topology is multi-channel: Attacker (as outsider) to intermediary (as user). The interaction point is either an external or internal channel. Intermediary (as user) to server (as victim). The activation point is an internal channel.