

White Paper: Application is vulnerable to Cross Site Request Forgery (CSRF).

Author: Jon Coon, GSEC, Founder and CEO, Link Mountain, LLC

This whitepaper is intended to further expand on information already provided in a Link Mountain penetration test report.

If your application was found to be vulnerable, you will find more information here to assist you in understanding and remediating this vulnerability.

Full Description: The application is vulnerable to Cross Site Request Forgery attack. Using CSRF, an attacker makes the victim perform actions that they didn't intend to perform, such as purchasing an item, changing or creating account information, or any other function provided by the vulnerable website. CSRF attacks work by causing a rogue HTTP request to be sent from an authenticated user's browser to the application, which then commits a transaction without authorization given by the target user. As long as the user is authenticated and a meaningful HTTP request is sent by the user's browser to a target application, the application does not know if the origin of the request is a valid transaction or an action that the user was tricked into performing while authenticated to the application.

If our penetration test report lists this finding, then the application allows a security sensitive action to be taken without requiring a unique transaction identifier, or prompting the user for additional confirmation. If an attacker can gain a list of potential users of the application, emails could be sent to the users. The emails could either directly contain hostile script, or contain a link to a website (perhaps some breaking news about something that would interest the victim) that contained the hostile script. Upon visiting the site or opening the email, the script would execute in the users browser context and the browser would send the users cookie with the request. If the user is logged in at the time and has permissions to perform the operation, the server would execute it without further confirmation from the user.

Severity: This will depend on the context of the finding, but it is often given a **High severity** rating by our security engineers because it often involves very sensitive operations and adversely affects audit trails.

Link Mountain Recommendation: There are three basic approaches to countering CSRF attacks:

1. The first approach is to require additional confirmation from the user before performing sensitive operations. This is accomplished by storing the original requested operation, and returning a page that asks for confirmation. The confirmation page should require information that an attacker would not have access to, such as log on credentials or a CAPTCHA device. With this approach, the user must confirm the operation and an attacker would not have advance knowledge of the information required to forge the confirmation. Only after confirmation is received is the original request performed. This approach counters CSRF attacks by denying the attacker all of the information required to forge a request, and is dependent on the strength of the confirmation action required. If the confirmation action itself is predictable enough to be forged, the attack can still work. This is a workable approach for GET requests, although data changing or security sensitive operations via GET are inherently unsafe to start with and should be strongly discouraged. This approach can adversely affect the user experience by introducing additional steps.
2. The second approach requires the use of a unique identifier (nonce). In this approach, a list is compiled prior to delivering the page to the user. The list is stored with the user session and contains all valid nonce tokens that were generated for all links and forms on a given page, and the list is deleted and rebuilt for each page request. The nonce can be derived from a *secure*

random number generator. A nonce from the list is then appended to each link or form on the requested page prior to being displayed to the user. The application checks to insure that the nonce passed with the HTTP request is valid for a given request (maps to a given response from the application). If it is valid, the application performs the requested action. If the nonce is invalid or not present, the application terminates the user session and displays an error to the user. This approach counters CSRF attacks by denying the attacker all of the information required to forge a request, and is dependent on the strength of the nonce. If the nonce is predictable or discoverable, the attack can still work. This is a workable approach for GET requests *provided that nonces are regenerated for each request and previous nonces expired so they cannot be used in subsequent requests if discovered in log files*. This is the most secure approach, but also the most complicated to implement and can adversely affect the user experience by breaking back-button and tabbed browsing functionality if not carefully done.

3. The third approach uses a second token in addition to the session identifier. When a user visits a site, the site should generate a cryptographically strong token and associate it with the user's session. The token is included in the response as a hidden form field with each form. The site should require every form submission to include this token. If the token is invalid or not present, the application terminates the user session and displays an error to the user. The token is invalidated when the user's session is invalidated. This approach counters CSRF attacks by denying the attacker all of the information required to forge a request, and is dependent on the strength and secrecy of the second token. If the token is predictable or discoverable, the attack can still work. It should only be used with relatively short session timeouts, as the longer the token is valid the greater the chance of discovery. This approach is often the simplest to implement but cannot be used with GET requests, since the token may be logged in web access logs (examples are: on your own access logs, your client's proxy server logs and also foreign site logs for sites that are linked to by your site - where the query strings often appear as referrer strings). The potential token disclosure plus the fact that the tokens are valid as long as the original session is valid makes this approach unsafe for GET requests.

Common misconceptions:

Browser enforcement of the Same Origin Policy prevents CSRF:

This is false. Browsers that implement the Same Origin Policy (SOP) will prevent scripts from **accessing the DOM** of a page originating with another domain **or accessing cookies** that originate from another domain, but they do not prevent scripts from sending requests to other domains. Furthermore, when a script sends a request to another domain (or when an img tag src attribute is set to another domain, etc), the browser will execute the request AND will send any cookies it has that are valid for the domain along with the request.

Session Ids protect against CSRF:

This is false. Session identifiers will not prevent CSRF. CSRF attacks require an authenticated user, but do not require the attacker to have access to session tokens or authentication credentials.

Using only the POST method protects against CSRF:

This is false. CSRF attacks can be easily crafted for POST requests.

ASP.net Event Validation automatically protects against CSRF:

This is false. It only protects for POST requests **and then only** when ViewStateUserKey is also enabled. The EnableEventValidation page directive is enabled by default since .Net 2.0, and applies a cryptographically secure nonce value for form validation. This nonce value does provide CSRF protection similar to method 3 above - **with one caveat**: ViewStateUserKey must also be enabled, and **is not enabled** by default. Without ViewStateUserKey, the __EVENTVALIDATION nonce will be the same for any two users and will NOT provide CSRF protection. When ViewStateUserKey is enabled the



Link Mountain, LLC PO Box 182 Port Sanilac Michigan, 48469 – www.linkmountain.com -

`__EVENTVALIDATION` nonce will be unique for each user session and will accomplish CSRF protection for POST requests.

Inactive sessions should be expired in a reasonably short time period. As in method 3 above, this is NOT a secure method of handling GET requests since the token will appear in log files (examples are your own logs, your client's proxy server logs and any foreign sites that you link to). If the application must handle data changing GET requests, method 1 or 2 above should be used instead of .net Event Validation.