Cross-Site Request Forgery (CSRF): CSRF

How To Fix Cross-Site Request Forgery (CSRF) using Microsoft .Net ViewStateUserKey and Double Submit Cookie

Overview

Cross-Site Request Forgery is an attack where a user is forced to execute an action in a web site without knowing the action ever took place. If a web site is vulnerable, an attacker can capture a well-known action and craft a malicious link duplicating the action. By luring a victim via email or another public web site to a web page that contains the malicious link, the action may be executed on the victim's behalf if they have an authenticated session in the vulnerable site.

To mitigate this vulnerability, a random nonce (one time token) should be added to the parameters required to execute each action and validated prior to execution. If this solution were implemented in our example above, the malicious link crafted by the attacker would contain a token that is no longer valid. As a result, the action requested on the victimâs behalf would be rejected without a valid token.

ViewStateUserKey & Double Submit Cookie

Starting with Visual Studio 2012, Microsoft added built-in CSRF protection to new web forms application projects. To utilize this code, add a new ASP .NET Web Forms Application to your solution and view the Site.Master code behind page. This solution will apply CSRF protection to all content pages that inherit from the Site.Master page.

The following requirements must be met for this solution to work:

- All web forms making data modifications must use the Site.Master page.
- All requests making data modifications must use the ViewState.
- The web site must be free from all Cross-Site Scripting (XSS) vulnerabilities. See how to fix Cross-Site Scripting (XSS) using Microsoft .Net Web Protection Library for details.

```
1  public partial class SiteMaster : MasterPage
2  {
3      private const string AntiXsrfTokenKey = "__AntiXsrfToken";
4      private const string AntiXsrfUserNameKey = "__AntiXsrfUserName";
5      private string _antiXsrfTokenValue;
6
7      protected void Page_Init(object sender, EventArgs e)
8      {
9          //First, check for the existence of the Anti-XSS cookie
10         var requestCookie = Request.Cookies[AntiXsrfTokenKey];
11         Guid requestCookieGuidValue;
12
13         //If the CSRF cookie is found, parse the token from the cookie.
14         //Then, set the global page variable and view state user
15         //key. The global variable will be used to validate that it matches in the v
16         //method.
17         if (requestCookie != null
```

```
        && Guid.TryParse(requestCookie.Value, out requestCookieGuidValue))
        {
            //Set the global token variable so the cookie value can be
            //validated against the value in the view state form field in
            //the Page.PreLoad method.
            _antiXsrfTokenValue = requestCookie.Value;

            //Set the view state user key, which will be validated by the
            //framework during each request
            Page.ViewStateUserKey = _antiXsrfTokenValue;
        }
        //If the CSRF cookie is not found, then this is a new session.
        else
        {
            //Generate a new Anti-XSRF token
            _antiXsrfTokenValue = Guid.NewGuid().ToString("N");

            //Set the view state user key, which will be validated by the
            //framework during each request
            Page.ViewStateUserKey = _antiXsrfTokenValue;

            //Create the non-persistent CSRF cookie
            var responseCookie = new HttpCookie(AntiXsrfTokenKey)
            {
                //Set the HttpOnly property to prevent the cookie from
                //being accessed by client side script
                HttpOnly = true,

                //Add the Anti-XSRF token to the cookie value
                Value = _antiXsrfTokenValue
            };

            //If we are using SSL, the cookie should be set to secure to
            //prevent it from being sent over HTTP connections
            if (FormsAuthentication.RequireSSL &&
            Request.IsSecureConnection)
            responseCookie.Secure = true;

            //Add the CSRF cookie to the response
            Response.Cookies.Set(responseCookie);
        }

            Page.PreLoad += master_Page_PreLoad;
        }

        protected void master_Page_PreLoad(object sender, EventArgs e)
        {
            //During the initial page load, add the Anti-XSRF token and user
            //name to the ViewState
            if (!IsPostBack)
            {
                //Set Anti-XSRF token
                ViewState[AntiXsrfTokenKey] = Page.ViewStateUserKey;

                //If a user name is assigned, set the user name
                ViewState[AntiXsrfUserNameKey] =
                Context.User.Identity.Name ?? String.Empty;
            }
            //During all subsequent post backs to the page, the token value from
            //the cookie should be validated against the token in the view state
            //form field. Additionally user name should be compared to the
            //authenticated users name
            else
            {
                //Validate the Anti-XSRF token
                if ((string)ViewState[AntiXsrfTokenKey] != _antiXsrfTokenValue
                || (string)ViewState[AntiXsrfUserNameKey] !=
                (Context.User.Identity.Name ?? String.Empty))
            {
            throw new InvalidOperationException("Validation of
            Anti-XSRF token failed.");
            }
        }
    }
}
```

Note that the above code example is auto-generated by Visual Studio 2012. But, it can be implemented as outlined above in prior versions of Visual Studio to provide the same protection against CSRF attacks.

Contributors

James Jardine

Authors

Eric Johnson

Latest Blog Posts

The SANS Blog is an active, ever-updating wealth of information including Cloud Security, DevOps, AppSec, and more.

Learn More »

Latest Tweets

@SANSCloudSec

View All Tweets »

Latest Papers

Top 5 Considerations for Multicloud Security
By Brandon Evans

How to Secure App Pipelines in AWS
By Dave Shackleford

JumpStart Guide to Application Security in Amazon Web Services
By Nathan Getty

View More »

"The course used real code with simple scenarios illustrating a full attack scenario and how to fix the vulnerabilities."
- Jeffrey Bell, Lockheed Martin

"Teaches important security concepts that allow developers to build solid applications."
- Marcus Gunn, Lockheed Martin

"Very practical. Something every developer should attend."
- Chris Yokley, Baldwin County Commission