A browser typically sends requests to web applications in one of two ways. It either sends data via URL parameters where a HTTP GET request is used or sends data via forms where HTTP POST is used. The application typically performs some action as a result, for example, inserting of a new user into a table, deleting a forum post, etc. A problem occurs if the web application does not check if the requests are generated by the application itself (a user clicks a link or submits a filled in form). An attacker can create a link for a certain action and send it to the user. The user then clicks the link and the action is performed without the user even noticing. This is called **Cross Site Request Forgery**.

So a user has to click an attacker's link or fill in an attacker's form. Another condition is that the user must be logged in to a vulnerable website. These days, almost every application provides the "keep me logged in" functionality, so this condition is easily met.

ASP.NET complicates such attacks because of ***ViewState.*** If ViewState is enabled, you cannot send tampered POST requests to an ASP.NET application because validation of ViewState fails. For this reason, many developers think that ASP.NET applications are bulletproof against CSRF. However, this is not exactly true:

- GET requests can still cause CSRF.
- ASP.NET does not take form values from **Request.Form** but from **Request.Params**. This is the reason why it is possible to perform so-called **One click attacks**, a special type of CSRF. An attacker simply sends ViewState and the values of form fields via GET. The trick is that the attacker can use ViewState generated by ASP.NET after POST and change the values of fields and the validation will still succeed.

CSRF tokens

Kentico also provides a ***security token*** mechanism for additional protection against forged POST requests:

1. Whenever a user initiates a new session, a session cookie (HTTP only) is saved to the browser. The cookie stores a randomly generated token.
2. When the user loads a page containing an HTML form, Kentico automatically adds a hidden field to the form data, which contains an encrypted value matching the user's token.
3. After the form data is submitted via POST, the system validates the hidden field's value against the user's token.

If an attacker attempts to forge a form, they cannot generate a valid value for the hidden token field. If the tokens do not match when the form is submitted, the system raises an error and the attack is blocked.

Using custom security tokens against CSRF

If you have your own security token implementation, you can disable the default tokens by adding the following key to the *appSettings* section of your web.config file:

```
<add key="CMSEnableCsrfProtection" value="false" />
```

CSRF protection on MVC pages

ViewState and the default security token mechanism only protect web form pages (the Kentico administration interface and standard portal engine or ASPX template pages). If your site has pages handled by MVC controllers and views, you need to add the **ValidateAntiForgeryToken** attribute to your action methods, and generate security tokens by calling the *@Html.AntiForgeryToken()* method in your MVC views that post to the action methods.

Examples of CSRF

Let's have a simple page without any content and the following code in the code behind:

```
if (!string.IsNullOrEmpty(Request.Form["UserID"]))
{
        DoSomeAction(Request.Form["UserID"]) ;
}
```

It does not matter what the DoSomeAction() method does. The important thing is that in this case, the action is performed with the value specified in the UserID field. If ViewState validation or security tokens are not used, anyone who is authorized and sends a form with this field can perform the action. And there is no way to check if the actual user really wants to do this action.

Let's have another page without content and the following code behind:

```
using CMS.Membership;
using CMS.Helpers;

...

if (MembershipContext.AuthenticatedUser.CheckPrivilegeLevel(UserPrivilegeLevelEnum.
GlobalAdmin))
{
        int userID = QueryHelper.GetInteger("UserID", 0);
        if (userID != 0)
        {
                Response.Write("I just deleted a user with id: " + userID);
        }
}
else
{
        Response.Write("You don't have sufficient permissions to delete the user");
}
```

This code is similar to the previous example. The only difference is that now, the UserID is taken from a query string (by the GET method).

The third example shows a one-click attack. Let's have a simple page with a textbox and a button. The following code handles the Onclick action of the button:

```
protected void btnSend_Click(object sender, EventArgs e)
{
        Response.Write(txtUserID.Text);
}
```

Users typically insert a value into the txtUserID textbox and click the button. But the attacker can forge a link and send it to an authorized user:

http://site/Page.aspx?*_VIEWSTATE=/wEPDwULLTE0MDM4MzYxMjNkZIb5PxpCoDI4Dt3C2LKzz8CnHkbd&txtUserID=<anything>&btnSend=Send&*_EVENTVALIDATION=/wEWAwLdr4fPBgLT8dy8BQKFzrr8AbhBL27NfMMamif/pHIFUlo41HNI

The attacker can change the **<anything>** part to any other value. ViewState is taken from the page that can be generated after postback on that page and the validation is successful.

What can CSRF attacks do?

Vulnerability to CSRF attacks depends on individual applications and on the security of the web server. For example, if the application is poorly implemented, then attackers can do anything that the victims of the attack could normally do.

Finding CSRF vulnerabilities in Kentico

If you find any page/control/etc., that performs an action on GET requests, there is a possibility of a CSRF vulnerability. For example, try to find the following and similar strings in your code:

- QueryHelper.GetString("action")
- EnableViewState="false"

If you find the second string in the **<%@ page** directive, it means that a developer turned off ViewState validation.

ViewState validation helps a lot to avoid POST CSRF, so globally, we strongly recommend keeping it enabled.

If a page does not have these features and does not perform any actions, it also does not need to be protected against CSRF.

Avoiding CSRF

Even if your application uses ViewState validation and the Kentico security tokens, a special case of CSRF is still possible: **one click attacks**. The problem is simple – ViewState is the same for all users. However, you can specify a key corresponding to the current user through the **ViewStateUserKey** page property. The recommended value is a user's session ID. All Kentico pages must inherit from **AbstractCMSPage** (the base page for all Kentico pages). AbstractCMSPage page sets ViewStateUserKey to a unique value for every user and thus prevents one click attacks.

Because ASP.NET ViewState validation and the Kentico security tokens protect the application against POST CSRF by default, ***only use POST requests for actions***.

By default (on the level of the global web.config file), ASP.NET is set to:

```
<machineKey validationKey="AutoGenerate,IsolateApps" decryptionKey="AutoGenerate,
IsolateApps" validation="SHA1" decryption="Auto" compatibilityMode="Framework20SP1" />
<pages buffer="true" enableSessionState="true" enableViewState="true"
enableViewStateMac="true" viewStateEncryptionMode="Auto">
```

This means that machineKey is generated automatically and ViewState is validated, including encoding of ViewState based on machineKeys. If a control requests it, ViewState is encrypted using SHA1.

Summary

- Do not use GET requests to perform actions, always use POST.
- Never globally disable validation of ViewState on pages (**EnableViewState** key).
- Leave the default security tokens enabled, unless you have your own mechanism that protects against CSRF.
- If you create a custom page, always make it inherit from one of the Kentico base pages.
- If you create a new Kentico page class, check that it directly or indirectly inherits from **AbstractCMSPage**.