Tapestry5CSRF

Tapestry 5 and Cross-site request forgeries

Cross-site request forgeries (CSRF) is a web application vulnerability which is often overlooked by web developers. If your application is vulnerable to CSRF and an attacker can entice you to request a specific URL (like open a page, view an image in webmail etc.) the attacker can execute random Tapestry actions perform post's etc. without the users consent. For example an attacker can change the users password or change the email address used to sent a new password to.

For more info on CSRF see for example: http://en.wikipedia.org/wiki/Cross-site_request_forgery.

One way to protect against CSRF is to add a non-guessable code, from now on called sid, to the URLs that need to be protected against CSRF. The sid is added to the user session to make sure that every user has his/her own sid. When Tapestry receives a request (for a page or action) and that page/action need to be protected a check is done to see if the sid from the URL matches the sid stored in the user session. If the sid from the URL is missing or the sid from the URL is different from the sid in session you know that the request was not generated by Tapestry.

Howard suggested two possible solutions

http://www.nabble.com/Re:-T5:-How-to-protect-against-'Cross-site-request-forgery'--p18697729.html

My implementation uses the LinkFactory approach because it does not require changing any code other than adding a few lines to you application module (there are some exceptions but I come to that later).

Implementation

ASO

The user sid need to be stored in the user session so we will need a ASO. The CSRF ASO uses currentTimeMillis as the source of the sid. This is just for demo purposes and you should change it to make it use some secure random source.

```
public class CSRF
{
        private final String securityCode;

        public CSRF()
        {
                // Note: Change this so the security code (sid) created by a
                // secure random source (like SecureRandom)
                securityCode = Long.toString(System.currentTimeMillis());
        }

        public String getSecurityCode() {
                return securityCode;
        }
}
```

Filter and LinkFactoryListener

To add the sid to newly created links we need an implementation of the LinkFactoryListener interface. For checking the sid we need an implementation of ComponentEventRequestFilter. The CSRFFilterImpl implements both although they can be split-up into two different classes.

First the CSRFFilter interface (strictly not required but I like to use interfaces)

Interface (CSRFFilter)

```
public interface CSRFFilter extends ComponentEventRequestFilter {
        /*
         * For now we do not need any methods.
         */
}
```

Implementation (CSRFFilterImpl)

```
/**
```

```
 * ComponentEventRequestFilter that checks if the request contains the correct secureID. Without
 * a secureID the application is vulnerable to 'Cross Site Request Forgeries' (CSRF).
 *
 * Note: This filter only protects actions (like component actions and posts). It does not protect
 * a page render request but that should not be problematic because a page render should not
 * have any side effects just render the page.
 *
 * @author Martijn Brinkers
 *
 */
public class CSRFFilterImpl implements CSRFFilter, LinkFactoryListener
{
        public static String CSRF_SECURITY_PARAMETER = "sid";

        private final ApplicationStateManager asm;
        private final Request request;
        private final Response response;
        private final LinkFactory linkFactory;
        private final RequestPageCache requestPageCache;

        /*
         * The page to redirect to when secureID is incorrect
         */
        private final String redirectTo;

        public CSRFFilterImpl(ApplicationStateManager asm, Request request, Response response,
                        LinkFactory linkFactory, RequestPageCache requestPageCache, String redirectTo)
        {
                Check.notNull(asm, "asm");
                Check.notNull(request, "request");
                Check.notNull(response, "response");
                Check.notNull(linkFactory, "linkFactory");
                Check.notNull(requestPageCache, "requestPageCache");
                Check.notNull(redirectTo, "redirectTo");

                this.asm = asm;
                this.request = request;
                this.response = response;
                this.linkFactory = linkFactory;
                this.requestPageCache = requestPageCache;
                this.redirectTo = redirectTo;
        }

        private String getSecurityCode() {
                return asm.get(CSRF.class).getSecurityCode();
        }

        public void createdActionLink(Link link)
        {
                link.addParameter(CSRF_SECURITY_PARAMETER, getSecurityCode());
        }

        public void createdPageLink(Link link)
        {
                /*
                 * Page links do not need the SecureID because pages should only show the page and
                 * not have any side effects.
                 */
        }

        public void handle(ComponentEventRequestParameters parameters,
                        ComponentEventRequestHandler handler)
        throws IOException
        {
        Page page = requestPageCache.get(parameters.getActivePageName());

        if (isCSRFProtected(page))
        {
                /*
                 * Check if the secureID is equal to the secureID in the session
                 */
                String secureID = request.getParameter(CSRF_SECURITY_PARAMETER);

                CSRF csrf = asm.get(CSRF.class);

                if (secureID == null || csrf == null || !csrf.getSecurityCode().equals(secureID))
                {
                        /*
                         * Illegal secure ID so redirect
                         */
                Link link = linkFactory.createPageLink(redirectTo, false);

                response.sendRedirect(link);

                return;
                }
        }

        /*
         * Not protected or secureID is correct so continue
         */
        handler.handle(parameters);
        }

        private boolean isCSRFProtected(Page page)
        {
                /*
                 * For now all actions are protected. We can always create a special Annotation when
                 * we need to specify which pages/actions are protected.
                 */
                return true;
        }
}
```

The Application module

You will need to add the following lines to your application module (or better create a special separate security module)

```
    public static void contributeFactoryDefaults(
            MappedConfiguration<String, String> configuration)
    {
          // sets the page to redirect to when the sid is incorrect
        configuration.add("csrf.redirectTo", "accessdenied");
    }


    /*
     * Builds the 'Cross Site Request Forgeries' (CSRF) filter.
     *
     * Note: must be EagerLoad'ed because we want the LinkFactory listener to be active from the
     * start.
     */
    @EagerLoad
    public static CSRFFilter buildCSRFLinkFactoryListener(ApplicationStateManager asm,
                    Request request, Response response, LinkFactory linkFactory,
                    RequestPageCache requestPageCache, @Inject @Value("${csrf.redirectTo}") String redirectTo)
    {
            CSRFFilterImpl filter = new CSRFFilterImpl(asm, request, response, linkFactory,
                            requestPageCache, redirectTo);

            linkFactory.addListener(filter);

            return filter;
    }

    public void contributeComponentEventRequestHandler(OrderedConfiguration<ComponentEventRequestFilter>
configuration,
                    CSRFFilter csrfFilter)
    {
            /*
             * Make sure the CSRF filter is injected before anything else
             */
            configuration.add("CSRFFilter", csrfFilter, "after:UploadException", "before:Ajax");
    }
```

The gotcha's

1.Only action links are protected. I think that page rendering should not contain any side-effects (ie. they should only render a page) so not having a sid on a page link should not be a problem security wise. A sid on a page link is also not something you want because it creates 'ugly' links that cannot be bookmarked (because the sid wil be invalid after the session is renewed). 2.Because every link now contains an extra attribute (?sid=xxxx) this can interfere with an existing application. A problem I encountered was that some AJAX links (like used by some Tapestry 3'rd party components) created in a Javascript library did not cope with a URL attributes. Fixing it however was pretty straightforward (see below) 3.All links (and form submits) are now protected so you somehow need to have a page (login page for example) that is not protected otherwise it is impossible to add the sid to the user session. Because I use (native) Spring security this was solved for automatically because the login page is handled by Spring security so you need white-filter the login page. You can for example add a white-filter check to CSRFFilterImpl.isCSRFProtected to filter out the login page. Another option would be to create a special Annotation (for example @NoCSRF) and check if that page contains the annotation.

Using a LinkFactory protects all action links (which I like because you cannot forget a link) but if you want to protect just a few links you can create a Mixin that just adds the 'sid' attribute. In the filter you now need to know which actions are protected and which not so you will need to do some extra filtering (see gotcha 3 above for some tips).

If you have any question feel free to ask

Martijn Brinkers

m.brinkers@pobox.com