ThingWorx: Cross-Site Request Forgery Defense

Cross-site request forgery (CSRF) is a web-based attack that submits forged requests on behalf of an unwitting victim. If a user is currently authenticated to a site and can be tricked into submitting a malicious command, the attacker can request a state-change request using the victim's credentials.

CSRF attacks require social engineering, a user who is in an active session, predictable URLs, and a website that does not verify source origin on incoming requests. These attacks are only useful for requesting state changes on a target server.

For more information, see the following links:

- https://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF)
- https://www.owasp.org/index.php/CSRF_Prevention_Cheat_Sheet

A common defense against CSRF attacks is to use synchronized tokens. However, ThingWorx uses an alternate approach and instead routes all requests through a strict content type filter. This approach is equally secure while being more appropriate for a stateless implementation.

Content Type Filtering

All requests to ThingWorx (<url-pattern>/*</url-pattern>) pass through the ContentTypeServlet filter.

Incoming requests are evaluated on the following criteria:

- Form data requests must be directed at a safe URL (/ThingWorx/Home). Otherwise, the request is denied.
- If a request does not include the POST, PUT, or DELETE method, then it is allowed through. (Only GET, POST, PUT or DELETE methods are mapped to a valid internal API request context.)
- If a request specifies POST, PUT, or DELETE, then it must meet one of the following criteria:
  - Includes a Content-Type header with a value of application/json, text/xml, or application/xml.
  - If the request has multi-step content, then it must include an X-XSRF-TOKEN header with a valid token (TWX-XSRF-TOKEN-VALUE).
- All method switch requests are ignored.

After a request clears the content type filter, the web server routes the request to serve static content or to execute an HttpService implementation, which is mapped to either the REST API or Service API.

Rest API

The ThingWorx REST API is used to get state, update state, and invoke services on the ThingWorx server for a given entity type. Each entity type implementation inherits from BaseService(HttpServlet).

All requests to modify state require POST, PUT, or DELETE and a content header with a value of application/json, text/xml, or application/xml.

GET requests return read-only data.

For more information, see Updating, Deleting, and Executing Through the API in the ThingWorx help center.

Service API

ThingWorx also provides custom services. Like the REST API, any service that modifies state requires POST, PUT, or DELETE and a valid content header.

Custom services outside of the REST API include the following:

- AvatarViewer—Read Only
- FileRepositories—Read Only
- FileRepositoryDownloader—Read Only
- OrganizationLogoViewer—Read Only
- DataImporter—POST service; requires a CSRF token
- ExtensionPackageUploader—POST service; requires a CSRF token
- FileRepositoryUploader—POST service; requires a CSRF token
- ImageEncoder—POST service; requires a CSRF token
- Importer—POST service; requires a Content-Type of application/json

Best Practices for CSRF Defense

All out-of-the-box ThingWorx implementations are secure against CSRF attacks using the methods described above.

To ensure CSRF protection, do not modify the following Platform Subsystem settings from their default values:

- **Allow Method Request Switch**—Default is false

- **Filter Content Type**—Default is true

  Note

  The options above are deprecated and will be removed from future ThingWorx releases.

Additional guidelines:

- When developing new applications, all mashups built in Composer are secure. If you are using a custom UI that accesses ThingWorx through the REST API, ensure that you are not including any request that requires a method switch.
- When developing new Java extensions, ensure that all new services are exposed through the ThingWorx service annotation framework. This framework is responsible for identifying, mapping, and routing all incoming requests to an implementation of a service. All services that are created and exposed using ThingWorx service annotations will be CSRF secured.
- In the rare case that a new service must bypass the ThingWorx annotation framework, ensure the service only reacts to the GET, POST, PUT, or DELETE methods. For example:

```
package com.thingworx.webservices;

import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public final class CustomServlet extends HttpServlet {
    private static final long serialVersionUID = -3741015242119110249L;

    @Override
    protected void doGet(HttpServletRequest request,
HttpServletResponse response) {
        // Read only operation
    }

    @Override
    protected void doPost(HttpServletRequest request,
HttpServletResponse response) {
        // Not read only operation
    }

    @Override
    protected void doPut(HttpServletRequest request,
HttpServletResponse response) {
        // Not read only operation
    }
    @Override
    protected void doDelete(HttpServletRequest request,
HttpServletResponse response) {
        // Not read only operation
    }
}
```