Article
David Crawford · Jul 31, 2019 2m read

Open Exchange

Anti CSRF Methods

IRIS provides us with anti **login** CSRF attack mitigation, however this is not the same as a CSRF attack, as login attacks only occur on the login form. There are currently no built-in tools to mitigate CSRF attacks on api calls and other forms, so this is a step in mitigating these attacks.

See the following link from OWASP for the definition of a CSRF attack:

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

The method shown in this article for mitigating these attacks is currently not proactive, but a minimum that needs to be combined with other attack vector prevention methods, like disabling CORS, mitigating http pollution, etc. By proactive, I mean that it is possible in the future for attackers to coerce a browser into creating arbitrary custom headers and values. Because this is not currently possible, the mere presence of a custom header is enough to mitigate this risk. See the following for further explanation:

https://security.stackexchange.com/questions/23371/csrf-protection-with-custom-headers-and-without-validating-token

Example

A rest call made to a CSP dispatch class extends the %CSP.REST class which gives us access to handling the headers before any intrusion. In the client code, we add the custom header to our request, along with junk data that can be populated in the future with specific information as a proactive approach:

```
$.ajax({
  beforeSend: function(request) {
  // Where we set the custom header. The value doesn't matter. In the future we want
to set this and verify the values matches the servers'
    request.setRequestHeader("Grandmas-Cookies", Math.random());
  },
  url:serverURL+url+data,
  type: "GET",
  async: true,
  contentType: "application/json",
  dataType: "text",
  error: function(response) {
    console.log(response);
  },
  success: successCallback
  });
```

Then on the server's Page function of the %CSP.REST extended class, we check for the presence of this header.

```
set header = %request.GetCgiEnv("HTTP_GRANDMAS_COOKIES")
// If blank, toss it out. Client browsers cannot be coerced into setting values of cu
```

```
stom headers
if header = "" {
  Set tSC=..Http403()
 Quit
}
```

If the header isn't there or it has no value, the request is dropped. We've tested this with several penetration testing tools including OWASP ZAP and Burp-Suite, and it works well. I'm curious if any of you out there have worked on anti CSRF methods for your Intersystems based applications, and how you helped mitigate them. A lot of security prevention is not "out of the box" with Caché/IRIS, so many vulnerabilities require custom solutions.

Thank you!

#CSP #JavaScript #REST API #Security #Frontend #Caché #InterSystems IRIS
Check the related application on InterSystems Open Exchange