Web Services API Reference

Cross-Site Request Forgery protection

5/7/2025

Cross-Site Request Forgery protection

Contents

Web Services provides protection against Cross-Site Request Forgery (CSRF) attacks by requiring a token in a custom header for all requests that modify data: PUT, POST, DELETE. Web Services generates and stores this token along with the HTTP session. The token shares the life cycle of the HTTP session.

See CSRF protection for details about how to enable this security feature.

To get the CSRF token and the expected header name from Web Services, just send a GET request — for example, **/api/v2/me**. The expected header name and token value are returned in two custom headers on the HTTP response: X-CSRF-HEADER and X-CSRF-TOKEN.

```
X-CSRF-HEADER: X-CSRF-TOKEN
X-CSRF-TOKEN: 4a92be65-ec55-4aa2-b9df-9518fd870f2f
```

You must cache the values of these headers because you'll need to use them on subsequent API requests that use PUT, POST, and DELETE so that Web Services doesn't think the request is coming from a third party. For instance, when you attempt to perform the StartContactCenterSession operation, you need include an HTTP header of X-CSRF-TOKEN with the corresponding value:

```
POST https://GWS-demo.genGWS.com/api/v2/me HTTP/1.1
Authorization: Basic <credentials>
X-CSRF-TOKEN: 4a92be65-ec55-4aa2-b9df-9518fd870f2f
Accept: application/json, application/xml, text/json, text/x-json, text/javascript,
text/xml
User-Agent: RestSharp/105.2.3.0
Content-Type: application/json
Host: GWS-demo.genGWS.com
Cookie: JSESSIONID=sngukrzemiyxchpu5isbufmm;
AWSELB=854B09E30CD5CEDDEDA518240935B76DEAC5D82EC5038C4B8F22CD5165FF21C65BC292BAD05CEEB
17D7500F4A489957FB3A5C23BD09BC31CAF09526FCBEFD7CE491CD7E5B3
Content-Length: 88
Accept-Encoding: gzip, deflate
{
   "operationName": "StartContactCenterSession",
   "channels": [
      "voice"
   ]
}
```

If you don't have that header in place, Web Services returns an HTTP 403 error with a response in the Content of "Missing or invalid Csrf token".

Cookie support

In addition to the CSRF feature, Web Services also requires your application to support cookies, specifically for the JSESSIONID cookie value that it returns. Without a cookie store, Web Services returns the same HTTP 403 error with a message of "Missing or invalid Csrf token", even if the X-CSRF-TOKEN is specified in the HTTP Header. This is because it can't confirm that the X-CSRF-TOKEN you specify lines up with the JSESSIONID that the token is supposed to be tied to.

Read on for some sample requests and examples of how to implement CSRF protection:

Authorized request returning token headers

Request

```
GET /api/v2/me

User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_9_3) AppleWebKit/537.36 (KHTML, like
Gecko) Chrome/35.0.1916.153 Safari/537.36
Authorization: Basic cGF2ZWxkQHJlZHdpbmdzLmNvbTpwYXNzd29yZA==
Accept: */*
Accept-Encoding: gzip,deflate,sdch
Accept-Language: en-US,en;q=0.8
Cookie: JSESSIONID=hac082exio454jcqk6ieqm4j
```

Response

```
200 - OK
Date: Mon, 23 Jun 2014 02:00:15 GMT
X-CSRF-HEADER: X-CSRF-TOKEN
Set-Cookie: JSESSIONID=1h49t997p4mgc1e108bz0cjntr;Path=/
Expires: Thu, 01 Jan 1970 00:00:00 GMT
X-CSRF-TOKEN: e2fcfafd-c600-4156-88ae-ca56babd24e1
Pragma: no-cache
Cache-Control: no-cache
Cache-Control: no-store
Content-Type: application/json
Transfer-Encoding: chunked
```

POST request including CSRF token

Request

```
POST /api/v2/me

Origin: chrome-extension://hgmloofddffdnphfgcellkdfbfbjeloo
X-CSRF-TOKEN: e2fcfafd-c600-4156-88ae-ca56babd24e1
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_9_3) AppleWebKit/537.36 (KHTML, like
Gecko) Chrome/35.0.1916.153 Safari/537.36
Content-Type: application/json
Accept: */*
Accept-Encoding: gzip,deflate,sdch
Accept-Language: en-US,en;q=0.8
Cookie: JSESSIONID=1h49t997p4mgc1e108bz0cjntr

{ "operationName": "Ready" }
```

Response

```
200 - OK
Date: Mon, 23 Jun 2014 02:02:51 GMT
Pragma: no-cache
Cache-Control: no-cache
Cache-Control: no-store
Content-Type: application/json
Transfer-Encoding: chunked
Server: Jetty(8.1.14.v20131031)
```

```
{ "statusCode": 0 }
```

JavaScript example

```
<html>
        <head>

                <script type="text/javascript" src="./org/cometd.js"></script>
            <script type="text/javascript" src="./org/cometd/ReloadExtension.js"></script>
                <script src="//ajax.googleapis.com/ajax/libs/jquery/1.11.1/
jquery.min.js"></script>
                <script src="./jquery.cometd.js"></script>

                <script>

                ////////////////////////////////////////////////////////////////////////
                // Initialization
                ////////////////////////////////////////////////////////////////////////
                var baseUri = 'http://127.0.0.1:8080';
                var username = 'paveld@redwings.com';
                var password = 'password';

                var csrfHeaderName;
                var csrfToken;
                var cometd;

                $.ajaxSetup({
             beforeSend: function(xhr) {
                    if (csrfHeaderName && csrfToken) {
                       xhr.setRequestHeader(csrfHeaderName, csrfToken);
                    }
             }
          });

                $(document).ready(function() {
                        $('#getMeButton').click(getMe);
                        $('#startCometdButton').click(connectCometD);
                        $('#startSessionButton').click(startContactCenterSession);
                        $('#readyButton').click(ready);
                        $('#stopCometdButton').click(disconnectCometD);
                        $('#endSessionButton').click(endContactCenterSession);

                        cometd = $.cometd;

                        cometd.addListener('/meta/handshake', onHandshake);
                    cometd.addListener('/meta/connect', onConnect);
                        cometd.addListener('/meta/disconnect', onDisconnect)

                        $(window).unload(function() {
                                cometd.disconnect();
                        });
                });

                ////////////////////////////////////////////////////////////////////////
                // HTTP Helpers
                ////////////////////////////////////////////////////////////////////////
                var get = function(params)
                {
                        var request = {
```

```
                    url: baseUri + params.uri,
                    type: 'GET',
                                crossDomain: true,
                                xhrFields: {
                                    withCredentials: true
                                },
                    success: function (data, textStatus, response) {
                            console.log(response.getAllResponseHeaders());

                            if (response.getResponseHeader('X-CSRF-HEADER') &&
response.getResponseHeader('X-CSRF-TOKEN')) {
                                csrfHeaderName = response.getResponseHeader('X-CSRF-HEADER');
                                csrfToken = response.getResponseHeader('X-CSRF-TOKEN');

                                console.log('csrfHeaderName: ' + csrfHeaderName);
                                console.log('csrfToken: ' + csrfToken);
                        }

                        if (params.callback) {
                                    params.callback(data);
                                }
                    },
                    error: function (result) {
                        console.log(result);

                        if (params.error) {
                                                params.error(result);
                                        }
                    }
                };

                if (params.includeCredentials) {
                        request.beforeSend = function (xhr) {
                                            xhr.setRequestHeader('Authorization', 'Basic ' +
window.btoa(username + ':' + password));
                                };
                        }

                $.ajax(request);
                };

                var post = function(params)
                {
                        var data = JSON.stringify(params.json, undefined, 2);

                var request = {
                    url: baseUri + params.uri,
                    type: 'POST',
                                data: data,
                    headers: {
                        'Content-Type' : 'application/json'
                    },
                                crossDomain: true,
                                xhrFields: {
                                    withCredentials: true
                                },
                    handleAs: 'json',
                    success: function(data) {
                            if (params.callback) {
                                    params.callback(data);
                            }
                    },
                    error: function (req, err, exception) {
```

```
        console.log('Error! (' + req.status + ') : ' + err + ' ' + exception);
        if (params.error) {
                                params.error(result);
                        }
    }
       };

$.ajax(request);
}

////////////////////////////////////////////////////////////////////////////
// API Functions
////////////////////////////////////////////////////////////////////////////
var getMe = function() {
        get({
                uri: '/api/v2/me',
                includeCredentials: true
        });
};

var startContactCenterSession = function() {
        post({
                uri: '/api/v2/me',
                json: {
                        operationName: 'StartContactCenterSession',
                        channels: ['voice']
                }
        });
};

var ready = function() {
        post({
                uri: '/api/v2/me',
                json: {
                        operationName: 'Ready'
                }
        });
};

var endContactCenterSession = function() {
        post({
                uri: '/api/v2/me',
                json: {
                        operationName: 'EndContactCenterSession'
                },
                callback: onEndContactCenterSessionComplete
        });
};

////////////////////////////////////////////////////////////////////////////
// Callbacks
////////////////////////////////////////////////////////////////////////////
var onEndContactCenterSessionComplete = function() {
        csrfHeaderName = null;
        csrfToken = null;
}

////////////////////////////////////////////////////////////////////////////
// CometD
////////////////////////////////////////////////////////////////////////////

var connected = false;
var subscription;
```

```
                var onConnect = function(message) {
                        if (cometd.isDisconnected()) {
                                return;
                        }

                        var wasConnected = connected;
                        connected = message.successful;
                        if (!wasConnected && connected) {
                                console.log('Cometd connected.');
                        } else if (wasConnected && !connected) {
                                console.log('Cometd disconnected...');
                        }
                };

                var onDisconnect = function(message) {
                        if (message.successful) {
                                connected = false;
                                console.log('Cometd disconnected.');
                        }
                };

                var onMessage = function(message) {
                        console.log('Cmetd message received:\n' + JSON.stringify(message,
null, 2));
                };

                var onHandshake = function(handshake) {
                if (handshake.successful === true) {
                                if (subscription) {
                                        console.log('unsubscribing: ' + subscription);
                                        cometd.unsubscribe(subscription);
                                }

                                console.log('Subscribing to channels...');
                                subscription = cometd.subscribe('/v2/me/*', onMessage);
                }
                };

                var connectCometD = function() {

                        var reqHeaders = {};
                        reqHeaders[csrfHeaderName] = csrfToken;

                        cometd.unregisterTransport('websocket');
                        cometd.unregisterTransport('callback-polling');
                        cometd.configure({
                            url: baseUri + '/api/v2/notifications',
                            logLevel: "info",
                            requestHeaders: reqHeaders
                        });

                        cometd.handshake();
                };

                var disconnectCometD = function() {
                        cometd.disconnect();
                };

                </script>
        </head>

        <body>
```

```
                <button id='getMeButton'>Get Me</button>
                <br/>
                <button id='startCometdButton'>Start CometD</button>
                </br>
                <button id='startSessionButton'>Start Contact Center Session</button>
                <br/>
                <button id='readyButton'>Ready</button>
                <br/>
                <button id='stopCometdButton'>Stop CometD</button>
                </br>
                <button id='endSessionButton'>End Contact Center Session</button>
        </body>
</html>
```

Python example

```
import base64;
import httplib2;
import json;

GWS_BASE_URI = "http://127.0.0.1:8080/api/v2"
ADMIN_USERNAME = "mikeb@redwings.com"
ADMIN_PASSWORD = "password"

CONTACT_CENTER_USERS = [
    {
        "userName": "bobp@redwings.com",
        "firstName": "Bob",
        "lastName": "Probert",
        "password": "password",
        "phoneNumber": "5019",
        "role": "ROLE_AGENT"
    }
]

X_CSRF_HEADER = "x-csrf-header"
X_CSRF_TOKEN = "x-csrf-token"

jsessionid = None
csrfHeaderName = None
csrfTokenValue = None

http = httplib2.Http(".cache")

def create_request_headers():
    request_headers = dict()
    request_headers["Content-Type"] = "application/json"
    request_headers["Authorization"] = "Basic " + base64.b64encode(ADMIN_USERNAME + ":" +
ADMIN_PASSWORD)

    if jsessionid:
        request_headers["Cookie"] = jsessionid;
        print "Using JSESSIONID %s" % jsessionid;

    if csrfHeaderName and csrfTokenValue:
        print "Adding csrf header [%s] with value [%s]..." % (csrfHeaderName, csrfTokenValue)
        print
        request_headers[csrfHeaderName] = csrfTokenValue
    else:
        print "No csrf token, skipping..."
```

```
        print

    return request_headers

def post(uri, content):
    request_headers = create_request_headers()
    body = json.dumps(content, sort_keys=True, indent=4)

    print "POST %s (%s/%s)..." % (uri, ADMIN_USERNAME, ADMIN_PASSWORD)
    print body
    print

    response_headers, response_content = http.request(uri, "POST", body = body, headers =
request_headers)
    status = response_headers["status"]

    ugly_response = json.loads(response_content)
    pretty_response = json.dumps(ugly_response, sort_keys=True, indent=4)

    print "Response: %s" % (status)
    print "%s" % (pretty_response)
    print

    return response_headers, ugly_response

def get(uri):

    global csrfHeaderName
    global csrfTokenValue
    global jsessionid

    request_headers = create_request_headers()
    print "GET %s (%s/%s)..." % (uri, ADMIN_USERNAME, ADMIN_PASSWORD)
    print

    response_headers, response_content = http.request(uri, "GET", headers = request_headers)
    status = response_headers["status"]
    if response_headers["set-cookie"]:
        jsessionid = response_headers["set-cookie"]
        print "Set JSESSIONID %s..." % jsessionid

    ugly_response = json.loads(response_content)
    pretty_response = json.dumps(ugly_response, sort_keys=True, indent=4)

    print "Response: %s" % (status)
    print "%s" % (pretty_response)
    print

    if X_CSRF_HEADER in response_headers:
        csrfHeaderName = response_headers[X_CSRF_HEADER]
        print "Saved csrf header name [%s]" % csrfHeaderName

    if X_CSRF_TOKEN in response_headers:
        csrfTokenValue = response_headers[X_CSRF_TOKEN]
        print "Saved csrf token value [%s]" % csrfTokenValue
        print

    return response_headers, ugly_response

def check_response(response_headers, expected_code):
    if response_headers["status"] != expected_code:
        print "Request failed."
        exit(-1)
```

```
def create_user(user_info):
    user_name = user_info["userName"]
    print "Creating user [%s]..." % (user_name)

    uri = "%s/users" % (GWS_BASE_URI)

    user = {
        "userName": user_name,
        "password": user_info["password"],
        "firstName": user_info["firstName"],
        "lastName": user_info["lastName"],
        "roles": [user_info["role"]]
    }

    response_headers, response_content = post(uri, user)
    check_response(response_headers, "200")

    user_id = response_content["id"]
    print "User [%s] created. User id [%s]." % (user_name, user_id)

    return user_id

def assign_device_to_user(user_id, phone_number):
    print "Creating device [%s] and assigning to user [%s]..." % (phone_number, user_id)

    uri = "%s/users/%s/devices" % (GWS_BASE_URI, user_id)

    device = {
        "phoneNumber": phone_number
    }

    response_headers, response_content = post(uri, device)
    check_response(response_headers, "200")

    device_id = response_content["id"]
    print "Device [%s] created and assigned to user id [%s]." % (device_id, user_id)

def create_users_and_devices():
    for user_info in CONTACT_CENTER_USERS:
        user_id = create_user(user_info)
        assign_device_to_user(user_id, user_info["phoneNumber"])

def getToken():
    uri = "%s/diagnostics/version" % (GWS_BASE_URI)

    response_headers, response_content = get(uri)
    check_response(response_headers, "200")

if __name__ == "__main__":

    getToken()
    create_users_and_devices()
```