# Cookie-based auth for REST APIs

**ON THIS PAGE**

**Cookie-based authentication is *deprecated***

Jira Cloud has deprecated cookie-based authentication in favor of basic authentication with API tokens or OAuth. We strongly recommend you use either of these authentication methods in place of cookie-based authentication.

See the deprecation notice for more information.

This page shows you how to allow REST clients to authenticate themselves using cookies ↗ . This is one of three methods that you can use for authentication against the Jira REST API; the other two being basic authentication and OAuth.

## Before you begin

Have you picked the right authentication method?

- **Building a Connect app**? If you are building an app that uses Atlassian Connect, authentication is built into the Atlassian Connect libraries. See Security for Connect apps.
- **Building a non-Connect integration**? If you are building an integration that doesn't use Connect, we recommend that you use OAuth 2.0 authorization code grants (3LO) for apps over other authentication methods, such as basic authentication and OAuth 1.0a. See Security for other integrations.

To complete this tutorial, you need to know the following:

- The basics of using REST APIs, e.g. requests, responses, headers.
- The basics of using and administering Jira.

- If you want to use the Node JS example, you'll need to know how to use Node.js.

## Overview

Jira's REST API is protected by the same restrictions which are provided via Jira's standard web interface. This means that if you do not log in, you are accessing Jira anonymously. Furthermore, if you log in and do not have permission to view something in Jira, you will not be able to view it using the Jira REST API either.

In most cases, the first step in using the Jira REST API is to authenticate a user account with your Jira site. Any authentication that works against Jira will work against the REST API. In this tutorial, we will use cookie-based (session) authentication.

This is how cookie-based authentication works in Jira at a high level:

1. The client creates a new session for the user, via the Jira REST API .
2. Jira returns a session object, which has information about the session including the session cookie. The client stores this session object.
3. The client can now set the cookie in the header for all subsequent requests to the Jira REST API.

Before you begin, please be aware that although cookie-based authentication has many benefits, such as performance (not having to make multiple authentication calls), it also has security risks. For example, your session cookies can be hijacked if handled improperly. This document does not go into the security implications of cookies, but if you should make yourself aware of the risks, before considering this approach.

**About these instructions**

You can use any text editor or REST client to do this tutorial. These instructions were written using the Sublime Text Editor ↗ and Chrome Advanced REST Client ↗ . If you are using other tools, you should use the equivalent operations for your specific environment.

## Step 1. Create a new session using the Jira REST API

We need to get a session cookie from Jira, so the first thing we need to do is create a new session using the `session` resource in the Jira REST API. *Tip: You can also use the* `session` *resource to get information about the currently authenticated user in the current session (GET), or log the current user out of Jira (DELETE).*

To do this, just POST the desired user credentials (as JSON) to the session resource:

| | |
|---|---|
| Example resource | `http://jira.example.com:8090/jira/rest/auth/1/session` |
| Example credentials | `{ "username": "myuser", "password": "mypassword" }` |

This will create a new session and return the requested session information, which will look similar to the following:

```
{
    "session":
        {
            "name":"example.cookie.name",
            "value":"6E3487971234567896704A9EB4AE501F"
        }
}
```

More importantly, you will get the session cookie (in the header of the response) from the server, which you can use in subsequent requests. You can see an example of this below. You'll notice that the cookie name and value are the same as the cookie name and value in the session response above.

```
Set-Cookie: example.cookie.name=6E3487971234567896704A9EB4AE501F; Path=/; HttpOnly
```

## Step 2. Use the session cookie in a request

Now that you've created a session, it's just a matter of setting the cookie in all subsequent requests to the server.

1. Store the session object on the client. The way that you do this will depend on how your client is implemented.
2. When you want to make a request, take cookie name and value from the session and use them to set the 'cookie' field in the header of your request. You can see an example of this below:

```
headers: {cookie: example.cookie.name=6E3487971234567896704A9EB4AE501F}
```

That's it! Now, when you submit the request, the session cookie will be used to authenticate you to the Jira server until the cookie expires.

## Example code

The following example is written for Node.js. It demonstrates how you get the session information from Jira, set the cookie using the session information, and then execute a request (in this case, a request for search results) with the cookie.

```
    if (response.statusCode == 200) {
        console.log('succesfully logged in, session:', data.session);
        var session = data.session;
        // Get the session information and store it in a cookie in the header
        var searchArgs = {
            headers: {
                // Set the cookie from the session information
                cookie: session.name + '=' + session.value,
                "Content-Type": "application/json"
            },
            data: {
                // Provide additional data for the Jira search. You can modify the JQL to search
                jql: "type=Bug AND status=Closed"
            }
        };
        // Make the request return the search results, passing the header information including t
        client.post("http://localhost:8090/jira/rest/api/2/search", searchArgs, function(
            console.log('status code:', response.statusCode);
            console.log('search result:', searchResult);
        });
    } else {
        throw "Login failed :(";
    }
});
```

## Advanced topics

### Cookie expiration

One disadvantage of using cookies compared to basic authorization is that they expire. You have probably noticed this when accessing Jira through a web browser. Every once in a while, especially if you have not used Jira in a while, you need to log in again because your cookie has expired. The same phenomenon occurs when using REST. If you are writing a script or code which involves REST API calls and:

- Only runs for a few minutes, then you should not have to worry about cookies expiring.
- Runs for a longer period of time due to more complex integration activities, then expiring cookies may cause problems.

If you use REST with a cookie that has expired, you will receive a 401 error response from Jira. The response body will contain a message telling you that your cookie is invalid. At that point, you will need to re-authenticate to the session resource on the "auth" API.

### CAPTCHAs

CAPTCHA upon login is 'triggered' after several consecutive failed log in attempts, after which the user is required to interpret a distorted picture of a word and type that word into a text field with each subsequent log in attempt.

Be aware that you cannot use Jira's REST API to authenticate with a Jira site, once Jira's CAPTCHA upon login feature has been triggered.

When you get an error response from Jira, you can check for the presence of an `X-Seraph-LoginReason` header in the response, which will contain more information. A value of `AUTHENTICATION_DENIED` means the application rejected the login without even checking the password, which most commonly indicates that Jira's CAPTCHA feature has been triggered.

## Unverified Email

If you receive a 409 error response from Jira, your email address is unverified and you will need to verify it. To do so, log in to Jira through your browser which will send you an email, allowing you to verify your account. Once this is done, you will be able to create a session with this resource.

## Form token checking

Jira employs a token authentication mechanism, which is used whenever Jira actions are performed either through link request or form submission. This provides Jira with the means to validate the origin and intent of the request, thus adding an additional level of security against cross-site request forgery.

This affects any REST endpoints that use form encoding, e.g. /issue/{issueIdOrKey}/attachments. If you need to use REST endpoints like these and you are not using basic authentication, you will need to disable the form token checking in your request. This is done by adding the appropriate header in your request. For details, see: Form Token Handling.