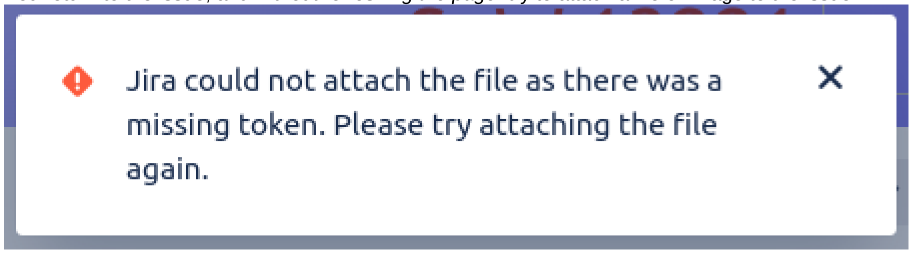
Missing token errors when attaching files or screenshots to Jira

This is *mostly* a tour of how Jira does CSRF (cross-site request forgery), written for my own benefit while investigating the problems eventually logged as JRASERVER-73880 - Attachments intermittently failing with 'missing token' error GATHERING IMPACT .

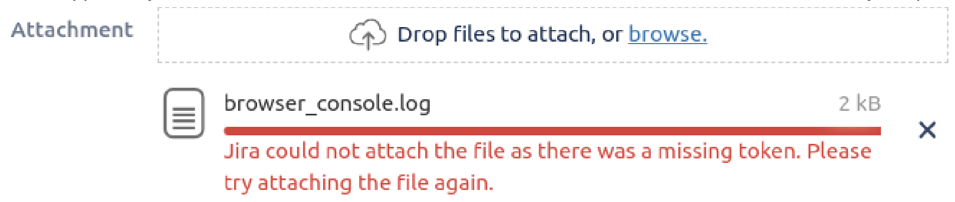
Have you ever tried to attach a file or image to a Jira issue, and got the error **Jira could not attach the file as there was a missing token. Please try attaching the file again.** ? This commonly happens when:

1. You are viewing a Jira issue
2. You leave your computer for over 5 hours ***--or--*** in another tab you log out and back in to Jira.
3. You return to the issue, and *without refreshing the page* try to attach a file or image to the issue.



Jira could not attach the file as there was a missing token. Please try attaching the file again.

It also happens if you click the 'Create' button to create a new issue, and on the 'Attachment' field try to upload an attachment.



Jira could not attach the file as there was a missing token. Please try attaching the file again.

Background: CSRF attacks

The 'missing token' in question is a CSRF token used to prevent CSRF (cross site request forgery) attacks.

An example of a CSRF attack would be as follows. An attacker creates a HTML page at https://hax0rs.com containing:

```
<a href="https://jira.yourcompany.com/secure/admin/user/AddUser.jspa?email=evil%40hax0rs.
com&fullname=Mr+Hax0r&username=hacker&password=hacker123&selectedApplications=jira-software&Create=Create+user"
>Click me!</a>
```

If you can trick a Jira administrator into visiting that site and clicking that link, a new `hacker` account is created in Jira.

Why did Jira, upon receiving that `AddUser.jspa` HTTP request, trust the sender? Because your browser identified the request as "coming from you" by including a identifying JSESSIONID cookie, which is scoped to `jira.yourcompany.com`.

SameSite=Lax

Until 2020, you could even embed the malicious URL in a `<img>` tag, and not even need the user to click That is no longer possible since `SameSite=Lax` became the default, 'Lax' meaning the `JSESSIONID` cookie is sent when a link is clicked, but not for non-interactive URLs like in <img>.

Jira's CSRF defenses

The `atl_token` CSRF token

Jira's main defense against CSRF is to generate a random bit of text (the 'CSRF token') just for you when you log in, and embed it in every HTML form that, if submitted, would changes server state (like adding users). The field is called `atl_token`:

```
<div class="hidden">
  <input name="atl_token" type="hidden" value="BTQP-72AW-MJ1A-4XOC_8de77ac52b28a9ddaad03e2a51ca7dcc90551c3e_lin">
</div>
```

When the user submits a form in Jira (on the 'Add User' admin page, for example), this `atl_token` parameter is submitted in the POST body. Jira checks that the `atl_token` value is correct before performing the operation. The CSRF token value is a secret known only to Jira and your browser, and changes after every login or session expiry (5h of inactivity). An attacker won't know the correct `atl_token` parameter to include in their forged URL.

This is known in the literature as the synchronizer token pattern (OWASP)

⚠ Only some Jira operations get token-based CSRF protection. It seems a bit haphazard, e.g. creating a Jira issue is CSRF protected, but adding a comment isn't. Specifically, CSRF protection is applied to:

- *.jspa Webwork actions like CreateIssueDetails.jspa, whose `execute()` method has a `@requiresXsrfCheck` annotation. For the implementation, see JiraActionFactory.java which invokes `DefaultXsrfInvocationChecker.java`.
- POST, DELETE AND put operations on REST resources, like , submitted via HTTP forms. See Atlassian REST API design guidelines for the rules here. This is implemented in XsrfResourceFilter, which also eventually leads to `DefaultXsrfInvocationChecker.java`

Origin/Referer checking

In addition to the `atl_token` mechanism, Jira also implements the 'Defence In Depth Techniques' on the OWASP page. Specifically, the `Origin` and `Referer` request headers are checked for REST resources (but perhaps not yet for `.jspa` requests - see **JRASERVER-63915** - Implement the origin csrf/xsrf checks that atlassian-rest has for JIRA actions. GATHERING INTEREST). `Origin` and `Referer` tell Jira where the request originated – https://hax0rs.com in our example. This is implemented in `XsrfResourceFilter.java` .

Jira's CSRF Implementation Details

Back to those `atl_token` parameters.

We know that the token is initially generated by Jira when you log in, and is embedded in the returned HTML:

```
<div class="hidden">
  <input name="atl_token" type="hidden" value="BTQP-72AW-MJ1A-4XOC_8de77ac52b28a9ddaad03e2a51ca7dcc90551c3e_lin">
</div>
```

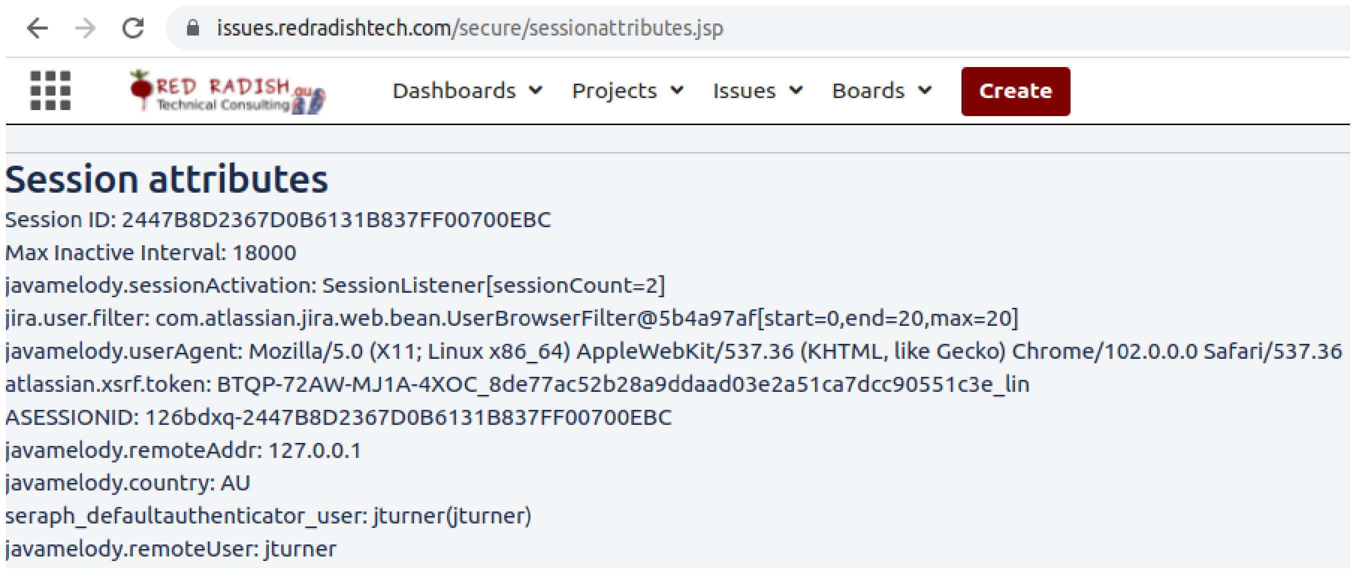
But where does Jira store its copy of the parameter? In two places actually:

1. the user session
2. in the user's browser as a cookie.

The `atlassian.xsrf.token` session attribute

While you are actively using Jira, Jira maintains a 'session' for you. A session is just a handful of key:value pairs in memory, grouped by 'session id', which is associated with your browser requests with the JSESSIONID cookie your browser sends.

If you are a Jira administator you can see your session's attributes by dropping sessionattributes.jsp into your `atlassian-jira/secure/` directory, and hitting the `/secure/sessionattribute.jsp` URL:


issues.redradishtech.com/secure/sessionattributes.jsp
RED RADISH.au Technical Consulting
Dashboards
Projects
Issues
Boards
Create

Session attributes

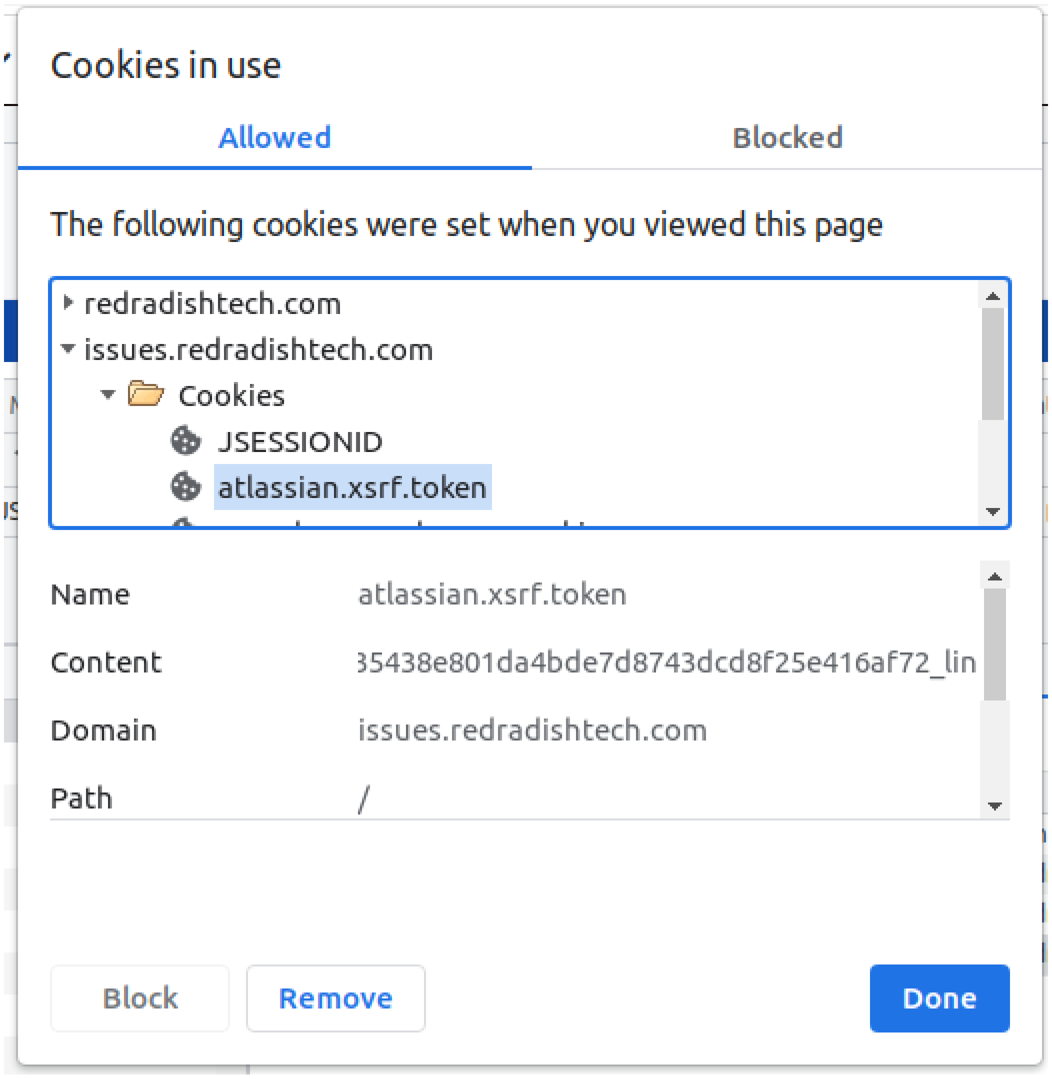
```
Session ID: 2447B8D2367D0B6131B837FF00700EBC
Max Inactive Interval: 18000
javamelody.sessionActivation: SessionListener[sessionCount=2]
jira.user.filter: com.atlassian.jira.web.bean.UserBrowserFilter@5b4a97af[start=0,end=20,max=20]
javamelody.userAgent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/102.0.0.0 Safari/537.36
atlassian.xsrf.token: BTQP-72AW-MJ1A-4XOC_8de77ac52b28a9ddaad03e2a51ca7dcc90551c3e_lin
ASESSIONID: 126bdxq-2447B8D2367D0B6131B837FF00700EBC
javamelody.remoteAddr: 127.0.0.1
javamelody.country: AU
seraph_defaultauthenticator_user: jturner(jturner)
javamelody.remoteUser: jturner
```

There in the session, you see where Jira is storing your CSRF token: in the **atlassian.xsrf.token** attribute (Jira uses 'XSRF' as a synonym for CSRF).

There are a few more wrinkles, but that is the essence of Jira's CSRF protection: all state-modifying HTTP requests must have an `atl_token` field, whose value must match the **atlassian.xsrf.token** session parameter.

The `atlassian.xsrf.token` cookie

If you examine the cookies your browser associates with Jira, you'll see an `atlassian.xsrf.token` cookie:


Cookies in use

Allowed | Blocked

The following cookies were set when you viewed this page

- redradishtech.com
- issues.redradishtech.com
  - Cookies
    - JSESSIONID
    - atlassian.xsrf.token

Name	atlassian.xsrf.token
Content	35438e801da4bde7d8743dcd8f25e416af72_lin
Domain	issues.redradishtech.com
Path	/

This is sent along with every HTTP request.

It would therefore be easy to think that *this atlassian.xsrf.token cookie is the thing in the request that needs validating*. Isn't the server just comparing the **atlassian.xsrf.token** request cookie with the **atlassian.xsrf.token** session attribute is knows is correct, and if they match, passing the CSRF check?

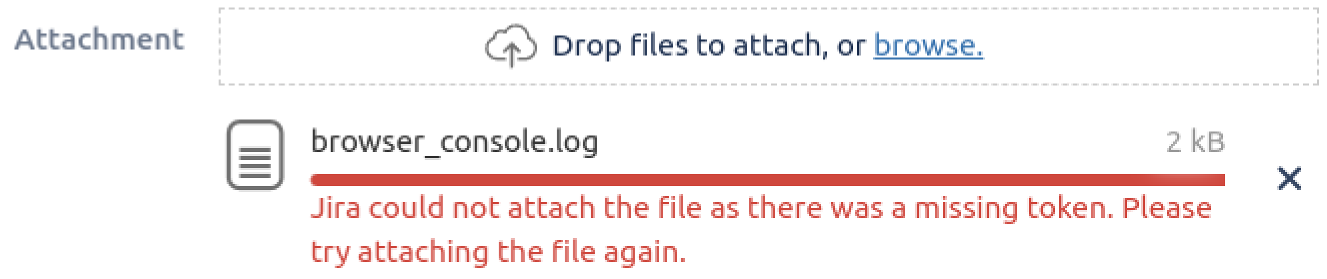
No! Remember that the **atlassian.xsrf.token** cookie is sent by the browser, and will always be present in malicious *and* non-malicious requests. Its value will always be correct, because cookies aren't under the attacker's control. The presence of a correct **atlassian.xsrf.token** cookie tells Jira nothing about whether the request has a malicious origin. It is the `atl_token` form parameter which needs validating, not the **atlassian.xsrf.token** request cookie.

In fact, when the server receives a request, it could compare the `atl_token` form parameter with either the **atlassian.xsrf.token** request cookie, or the **atlassian.xsrf.token** session attribute. In fact Jira does both (in `XsrfTokenStrategy.java`).

Why does Jira even set the **atlassian.xsrf.token** cookie? No idea. The general question of "why send the CSRF token as a cookie" is asked and answered on Stackoverflow. I'm not sure if any of those advantages actually apply to Jira. Certainly Jira does not populate the `atl_token` form field from the cookie, as it might do. and does not implement the "double submit cookie" pattern where the cookie value is compared with the form value, removing the need for server-side state. Perhaps it is handy to have AJAX Javascript-initiated requests automatically CSRF-protected.

When sessions expire..

One nice thing Jira *could* have done with the incoming **atlassian.xsrf.token** cookie it is re-initialize the **atlassian.xsrf.token** session parameter from the cookie if the session has expired. Then if your session expired overnight, and you had a Jira form half-filled in, the submit (next morning) would work, as the session **atlassian.xsrf.token** would be seeded from the **atlassian.xsrf.token** cookie. Currently that doesn't happen: if the session has expired, `XsrfTokenAdditionRequestFilter.java` (very early in the filter chain) just generates a new **atlassian.xsrf.token** session attribute, which is then compared to the `atl_token` field, fails to match, and you get a 'missing token' error to brighten your Monday morning:



Jira could not attach the file as there was a missing token. Please try attaching the file again.

..to Be Continued

Apologies to anyone reading, but there is no conclusion yet. The fix suggested above (copy the token from cookie to session) could be implemented as a custom servlet filter, even implemented as a plugin. If I ever do so I will update this post.