Session Token Doesn’t Expire On Password Change #PT12068 1

Pending Fix Medium · Authentication and Sessions · Aug 09, 2022

(pranavhivar...)

Vulnerability Type

Authentication and Sessions > Failure to Invalidate Session > On Password Reset and/or Change

Description

Session termination is an important part of the session lifecycle. Reducing to a minimum the lifetime of the session tokens decreases the likelihood of a successful session hijacking attack. This can be seen as a control against preventing other attacks like Cross Site Scripting and Cross Site Request Forgery. Such attacks have been known to rely on a user having an authenticated session present. Not having a secure session termination only increases the attack surface for any of these attacks.

A secure session termination requires at least the following components:

- Availability of user interface controls that allow the user to manually log out.
- Session termination after a given amount of time without activity (session timeout).
- Proper invalidation of server-side session state.

There are multiple issues which can prevent the effective termination of a session. For the ideal secure web application, a user should be able to terminate at any time through the user interface. Every page should contain a log out button on a place where it is directly visible. Unclear or ambiguous log out functions could cause the user not trusting such functionality.

Another common mistake in session termination is that the client-side session token is set to a new value while the server-side state remains active and can be reused by setting the session cookie back to the previous value. Sometimes only a confirmation message is shown to the user without performing any further action. This should be avoided.

Some web application frameworks rely solely on the session cookie to identify the logged-on user. The user's ID is embedded in the (encrypted) cookie value. The application server does not do any tracking on the server-side of the session. When logging out, the session cookie is removed from the browser. However, since the application does not do any tracking, it does not know whether a session is logged out or not. So by reusing a session cookie it is possible to gain access to the authenticated session. A well-known example of this is the Forms Authentication functionality in ASP.NET.

Users of web browsers often don't mind that an application is still open and just close the browser or a tab. A web application should be aware of this behavior and terminate the session automatically on the server-side after a defined amount of time.

The usage of a single sign-on (SSO) system instead of an application-specific authentication scheme often causes the coexistence of multiple sessions which have to be terminated separately. For instance, the termination of the application-specific session does not terminate the session in the SSO system. Navigating back to the SSO portal offers the user the possibility to log back in to the application where the log out was performed just before. On the other side a log out function in a SSO system does not necessarily cause session termination in connected applications.

Affected URL(s)

```
https://pentest.beta.nuxeocloud.com
```

Proof of Concept

1] Login into your account on https://pentest.beta.nuxeocloud.com in two separate browsers
2] From one browser, go to https://pentest.beta.nuxeocloud.com/nuxeo/ui/#!/profile and change your password
3] Now, check the old session tokens, they will be still active.

Severity

Assuming user's account is compromised by an attacker and attacker is logged into user's account. User changing/resetting his/her password so as to secure his/her account won't log out the attacker. Attacker can still use the account and enjoy privileges as the user.

Suggested Fix

Properly invalidate all user sessions server-side when the user resets their password and at a minimum, invalidate all non-current user sessions sever-side when the user changes their password.