SameSite

This is a summary of our current understanding of the impact of the Chrome SameSite change that has been well-publicized, and is due to appear in Chrome 80 on February 17, 2020.

While this page discusses the direct implications of this change on the operation of the SP software, the fact is that the bulk of the concerns in most cases lie within the application space and are not going to be remedied by anything here. Nor can we provide any sort of yes/no or good/bad conclusion for anybody as to whether "their system is affected". That is going to depend entirely on the individual case and the only real answer is to test.

Firefox is generally the most effective way to test this because its hidden settings will take effect instantly and do not include the 2 minute grace window that Chrome uses, so you can see the real impact of the change quickly.

The SP is impacted in four primary ways because it uses cookies in four significant ways. See also the CookieUsage topic.

All of these cookies can be controlled with the `cookieProps` setting and given a SameSite=None attribute, but that **will** break all non-current macOS and iOS Safari browsers. We do not encourage using that at this time because an update is planned that will provide more fine-grained control with appropriate workarounds.

Finally, note that a typical source of problems for most applications is going to be load balancer behavior. If you're using cookies for node affinity, you're going to have problems with SameSite unless you do something about it.

Session Cookies

There is at least one, and possibly two (if the optional recovery feature is enabled), cookies created to track each session. These cookies are created only **after** the form POST is delivered from the IdP, and so are technically not cross-site in the general sense. In most cases, they do not need to be marked SameSite, and in fact to do so is to defeat the entire rationale behind the change Google is making.

However, some applications expect to be able to compose themselves with other applications in a cross-site way that may include forms, and to do so will absolutely break under most conditions, or at least cause a new login to be necessary. In most cases, the best thing for such applications is to avoid all use of the SP session after login, and simply transition to its own mechanism, which will of course need to be adapted to deal with the SameSite change.

RelayState

The SP can, but does not default to, using a cookie to track the URL accessed to enable deep linking. The use of this mechanism can be easily identified by the presence of the "cookie" keyword in the RelayState tokens it generates in its requests. This has not been the default for many years, but is sometimes used because the recommended alternative relies on server state and so requires stickiness from the very beginning of access. Since stickiness is a general requirement anyway, this doesn't usually impose much of an additional burden.

This is a truly cross-site cookie case, and definitely will cause the relay state to be lost and land the browser on the `homeURL` or root of the site. Deep linking will only function when Chrome's 2 minute window is large enough to cover the login process, which is certain for SSO but definitely not certain otherwise.

Realistically, if memory-backed state really isn't an option, most deployers are probably better off just removing the `relayState` setting altogether and letting the URL pass by value, though this is technically not permitted by the standard due to size.

Form Recovery

The most unavoidable cross-site use case is the POST recovery feature that allows form data to be kept by the server and replayed after a login. This does rely on a cookie, and if the cookie doesn't appear during the SAML POST step, the form will not be replayed.