Cookie and Session Attacks: Stealing the Thing That Says You Are You

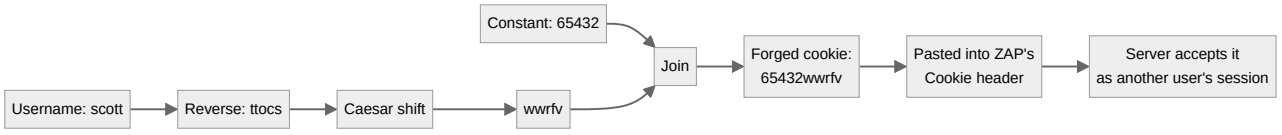
Fri, Sep 25, 10:00 AM EDT · https://scottslab.io/posts/cookie-and-session-attacks

> **TL;DR**
>
> I forged another user's session in a lab by editing one HTTP header, because the cookie was built from a guessable formula instead of random bytes. A stolen cookie gets reused wholesale in session replay, no formula required. The Secure attribute, blocking third party cookies, and clearing cookies on browser close each shut off a different piece of the exposure, and none of them alone is enough.

The forged cookie

In a lab exercise against a deliberately vulnerable app, the session cookie was not random. It was built from a formula: the fixed number 65432, followed by the username reversed and run through a Caesar shift. Once the pattern held across two accounts I controlled, reconstructing a third user's cookie stopped being guesswork and became arithmetic. I opened ZAP, intercepted a request, swapped the Cookie header for the value I had computed from another username, and the app treated me as that user. No password, no phishing, just a header edit.

That is the whole lesson about guessable session identifiers in one line: if a cookie is derived from something predictable, anyone who can see the pattern in their own account can compute someone else's. The fix sounds obvious and is not optional. The value has to come from a cryptographically random generator with enough entropy that guessing or brute forcing it is infeasible, not from concatenating known facts about the user.



The forged cookie

Session replay needs no formula at all

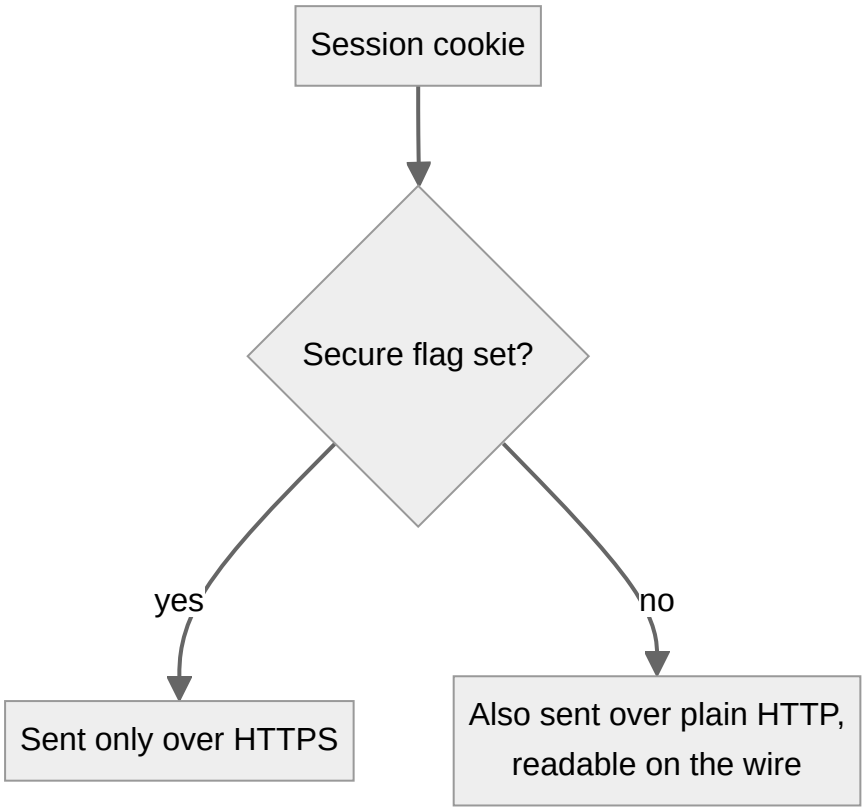
A guessable cookie is derived. Session replay is simpler and far more common: someone steals a valid cookie whole, by whatever route, an open wifi capture, an XSS payload reading `document.cookie`, an unlocked machine, a log file that should never have held it, and presents that exact value as their own. The server has no way to tell a legitimate browser from an attacker's,

because the cookie is the only proof of identity it checks. It's a bearer credential. Whoever holds it is the user, until it expires or gets revoked.

That's why session identifiers need a reasonable expiry and why regenerating the session ID on login and on any privilege change matters. A cookie that lives for weeks and never rotates gives replay a wide window to work, even if the original value was random and unguessable.

Secure closes one specific door

The single most concrete browser side defense is the Secure attribute on the cookie. Set it, and the browser sends that cookie only over HTTPS. Without it, the same cookie protecting an encrypted session can be handed over in plaintext the moment a user follows an http link, gets downgraded by a captive portal, or hits a misconfigured redirect, and anyone on the network path reads it off the wire. Secure does not stop replay or forgery by itself. It just guarantees the cookie can't leak over an unencrypted transport, which is one specific and very common way it does leak. Pair it with HttpOnly, so client side script can't read the cookie at all, and two attributes on one config line close two separate exposure paths.



Secure closes one specific door

What the browser can do on its own

Some controls sit on the user's side of the connection, not the server's, and they matter even though you can only enforce them for yourself. Blocking third party cookies stops a cookie set by one site

from being read on a different page that embeds it, which cuts down cross site tracking and closes off one class of attack that depends on a cookie being visible somewhere it shouldn't be. Clearing cookies on browser close is blunter: every session dies with the tab, killing replay value for anything stolen from that machine, at the cost of logging back into everything the next morning. Neither is a substitute for the server doing its job properly. They're a second layer for a user who doesn't fully trust the first one, and on a shared or public machine they're worth turning on regardless of what the site itself does right.

What I actually check

When I look at a session mechanism now, the questions are deliberately boring. Is the identifier generated with real entropy, or built from anything I could reconstruct from public or self observable facts. Is Secure set, and is the app actually served over HTTPS everywhere, not just on the login page. Does the session ID rotate on privilege change, not just on login. How long does a stolen cookie stay useful before it expires on its own. None of these are exotic questions. All of them were, in that lab, the exact difference between a header edit that worked and one that would have done nothing.

The uncomfortable part isn't that the forgery worked. It's how little skill it took once the pattern was visible. A random cookie doesn't require the developer to be clever. It just requires them not to be clever, and to let a library generate the value instead of building it out of parts anyone could reason about.

Written by Scott S. — Contact: scott@scottschlangen.com — scottslab.io — https://scottslab.io/posts/cookie-and-session-attacks