Insecure Token Handling

One difference between cookies and tokens is that cookies are used to send and store arbitrary data, while tokens are explicitly used to send authorization data. When we perform token-based authentication such as OpenID, or OpenID Connect, we receive an `id` token from a trusted authority. This is often referred to as `JSON Web Token` (`JWT`) and token-based authentication.

A typical use case for `JWT` is continuous authentication for `Single Sign-On` (`SSO`). However, `JWT` can be used flexibly for any field where compact, signed, and encrypted information needs to be transmitted. A token should be generated safely but should be handled safely too. Otherwise, all its security could break apart.

Token Lifetime

A token should expire after the user has been inactive for a given amount of time, for example, after 1 hour, and should expire even if there is activity after a given amount of time, such as 24 hours. If a token never expires, the Session Fixation attack discussed below is even worse, and an attacker could try to brute force a valid session token created in the past. Of course, the chances of succeeding in a brute force attack are proportionate to the shortness of the cookie value itself.

Session Fixation

One of the most important rules about a cookie token is that its value should change as soon as the access level changes. This means that a guest user should receive a cookie, and as soon as they authenticate, the token should change. The same should happen if the user gets more grants during a `sudo-like` session. If this does not occur, the web application, or better any authenticated user, could be vulnerable to `Session Fixation`.

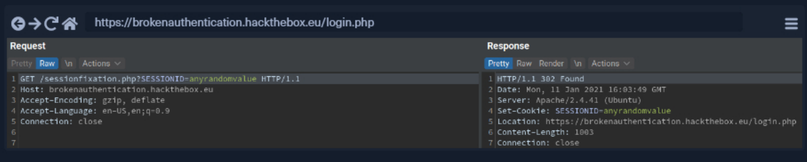
This attack is carried out by phishing a user with a link that has a fixed, and, unknown, by the web application, session value. The web application should bounce the user to the login page because, as discussed, the `SESSIONID` is not associated with any valid one. When the user logs in, the `SESSIONID` remains the same, and an attacker can reuse it.

A simple example could be a web application that also sets `SESSIONID` from a URL parameter like this:

- `https://brokenauthentication/view.php?SESSIONID=anyrandomvalue`

When a user that does not have a valid session clicks on that link, the web application could set `SESSIONID` as any random value.

Take the below request as an example.



At line 4 of the server's response, the `Set-Cookie` header has the value specified at the URL parameter and a redirect to the login page. If the web application does not change that token after a successful login, the phisher/attacker could reuse it anytime until it expires.

Token in URL

Following the Session Fixation attack, it is worth mentioning another vulnerability named `Token in URL`. Until recent days, it was possible to catch a valid session token by making the user browse away from a website where they had been authenticated, moving to a website controlled by the attacker. The `Referer` header carried the full URL of the previous website, including both the domain and parameters and the webserver would log it.

Nowadays, this attack is not always feasible because, by default, modern browsers strip the `Referer` header. However, it could still be an issue if the web application suffers from a `Local File Inclusion` vulnerability or the Referer-Policy header is set in an unsafe manner.

If we can read application or web server logs, we may also obtain a high number of valid tokens remotely. It is also possible to obtain valid tokens remotely if we manage to compromise an external analytics or log collection tool used by a web server or application. You can learn more and practice this attack by studying the File Inclusion / Directory Traversal module.

Session Security

Secure session handling starts from giving the counterpart, the user, as little information as possible. If a cookie contains only a random sequence, an attacker will have a tough time. On the other side, the web application should hold every detail safely and use a cookie value just as an `id` to fetch the correct session.

Some security libraries offer the feature of transparently encrypting cookie IDs also at the server level. Encryption is performed using some hardcoded values, concatenated to some value taken from the request, such as User-Agent, IP address or a part of it, or another environment variable. An excellent example of this technique has been implemented inside the Snuffleupagus PHP module. Like any other security measure, cookie encryption is not a silver bullet and could cause unexpected issues.

Session security should also cover multiple logins for the same user and concurrent usage of the same session token from different endpoints. A user should be allowed to have access to an account from one device at a time. An exception can be set for mobile access, which should use a parallel session check. Suppose the web application can identify the endpoint, for example, by using the user agent, screen size and resolution, or other tricks used by trackers. In that case, it should set a sticky session on a given endpoint to raise the overall security level.

Cookie Security

Most tokens are sent and received using cookies. Therefore, cookie security should always be checked. The cookie should be created with the correct path value, be set as `httponly` and `secure`, and have the proper domain scope. An unsecured cookie could be stolen and reused quite easily through Cross-Site Scripting (XSS) or Man in the Middle (MitM) attacks.

Previous Next

Mark Complete & Next

Cheat Sheet

Resources

Table of Contents

Broken Authentication

What is Authentication

Overview of Authentication Methods

Overview of Attacks Against Authentication

Login Bruteforcing

Default Credentials

Weak Bruteforce Protections

Bruteforcing Usernames

Bruteforcing Passwords

Predictable Reset Token

Password Attacks

Authentication Credentials Handling

Guessable Answers

Username Injection

Session Attacks

Bruteforcing Cookies

Insecure Token Handling

Skill Assessment

Skill Assessment - Broken Authentication

My Workstation

OFFLINE

Start Instance

∞ / 1 spawns left