Learning Objectives: Cookies

Learners will be able to...

- **Describe cookies types and how it works**
- **Determine what can and cannot be stored in cookies**
- **Use cookies best practices: HttpOnly, Secure, SameSite, etc.**

info

Make Sure You Know

- HTML basics
- JavaScript basics

What is a Cookie?

An HTTP cookie (web cookie, browser cookie) is a small piece of data that a server sends to a user's web browser. The browser may store the cookie and send it back to the same server with later requests. Typically, an HTTP cookie is used to tell if two requests come from the same browser—keeping a user logged in, for example. It remembers stateful information for the stateless HTTP protocol.

Cookies are mainly used for three purposes:

1. **Session management** - Logins, shopping carts, game scores, or anything else the server should remember
2. **Personalization** - User preferences, themes, and other settings
3. **Tracking** - Recording and analyzing user behavior

Cookies were once used for general client-side storage. While this made sense when they were the only way to store data on the client, modern storage APIs are now recommended. Cookies are sent with every request, so they can worsen performance (especially for mobile data connections).

Creating cookies

After receiving an HTTP request, a server can send one or more `Set-Cookie` headers with the response. The browser usually stores the cookie and sends it with requests made to the same server inside a Cookie HTTP header. You can specify an expiration date or time period after which the cookie shouldn't be sent. You can also set additional restrictions to a specific domain and path to limit where the cookie is sent. For details about the header attributes mentioned below, refer to the Set-Cookie reference article.

The Set-Cookie HTTP response header sends cookies from the server to the user agent. A simple cookie is set like this:

```
Set-Cookie: <cookie-name>=<cookie-value>
```

This instructs the server sending headers to tell the client to store a pair of cookies:

```
HTTP/2.0 200 OK
Content-Type: text/html
Set-Cookie: cookie_name=value
Set-Cookie: second_cookie_name=value2
[page content]
```

Then, with every subsequent request to the server, the browser sends all previously stored cookies back to the server using the Cookie header.

```
GET /page.html HTTP/2.0
Host: example.org
Cookie: cookie_name=value; second_cookie_name=value2
```

Types of Cookies

First-party Cookies

First-party cookies are set by the website visited by the user. The data collected using first-party cookies is used for purposes like calculating pageviews, sessions, and number of users.

Primarily, publishers have access to data collected using first-party cookies, which can later be shared with advertisers or agencies for ad targeting. Apart from that, analytics tools – like Google Analytics – use first-party cookies to understand user behavior and present it in tabular or graphical form for the publisher's understanding.

Third-party Cookies

Third-party cookies are set by domains that are not directly visited by the user. This happens when publishers add third-party elements (like chatbot, social plugins, or ads) on their website.

Once installed, third-party cookies also track users and save their information for ad targeting and behavioral advertising. For example: Let's say that you added a YouTube link to one of your blogs. Whenever this YouTube link gets a click, a YouTube cookie will be added to the user's browser. This cookie can track him/her until it expires.

Session Cookies

Session cookies either expire immediately or within a few seconds of the user leaving the web browser. Among other uses, these cookies are used by e-commerce websites to remember the product placed in cart by the user, to keep users logged in, and to calculate each user session for analytical purposes.

For example, if an e-commerce website does not use session cookies, then the items added in cart will be removed by the time the user reaches the checkout page. And the server will forget the user and treat him/her like a completely new visitor.

Persistent Cookies

As the name suggests, persistent cookies stay on the user's browser for a very long time. Generally, persistent cookies are required to have an expiration date which could be anything between a second to 10 years. The

above shared screenshot is an example of persistent cookies with expiration date.

Persistent cookies are used by publishers to track a single user and his/her interaction with their website. To check whether your browser has persistent cookies, try this. If you are logged in to Gmail on the browser, then close the tab(s) and restart your device. When your device turns back on, open the same browser and visit the same service or account (Gmail), if you are still logged in, then you have persistent cookies saved on the browser, dropped by Google mail service.

What could I store in cookies?

HTTP cookies were born to standardize certain mechanisms across browsers. They're nothing more than a way to store data sent by the server and send it along with future requests. The server sends a cookie, which contains small bits of data. The browser stores it and sends it along with future requests to the same server.

Why would we bother about cookies from a security perspective? Because the data they contain is, more often than not, extremely sensitive. Cookies are generally used to store session IDs or access tokens, an attacker's holy grail. Once they are exposed or compromised, attackers can impersonate users, or escalate their privileges on your application.

In addition, you should remember that you cannot store information in cookies that an attacker can use directly. Such information includes: user passwords in any form, their personal data, card data, and so on. If possible, it is necessary to minimize the information that is stored and transmit only the user session, the use of which is limited in time and place. This is necessary because the browser sends all the cookies that are available for a given request in the clear and attackers can find a workaround to get it.

Cookies Security: Secure

The **Secure** cookie attribute instructs web browsers to only send the cookie through an encrypted HTTPS (SSL/TLS) connection. This session protection mechanism is mandatory to prevent the disclosure of the session ID through MitM (Man-in-the-Middle) attacks. It ensures that an attacker cannot simply capture the session ID from web browser traffic.

Forcing the web application to only use HTTPS for its communication (even when port TCP/80, HTTP, is closed in the web application host) does not protect against session ID disclosure if the Secure cookie has not been set - the web browser can be deceived to disclose the session ID over an unencrypted HTTP connection. The attacker can intercept and manipulate the victim user traffic and inject an HTTP unencrypted reference to the web application that will force the web browser to submit the session ID in the clear.

The **Secure** attribute is meant to protect against man-in-the-middle (MITM) attacks. However, it protects only the confidentiality of the cookie, not its integrity. In a MITM attack, an attacker located between the browser and the server will not receive the cookie from the server via an unencrypted connection but can still send a forged cookie to the server in plain text.

```
HTTP/2.0 200 OK
Content-Type: text/html
Set-Cookie: cookie_name=value; Secure


[page content]
```

Cookies Security: HttpOnly

The **HttpOnly** cookie attribute instructs web browsers not to allow scripts (e.g. JavaScript or VBscript) an ability to access the cookies via the DOM document.cookie object. This session ID protection is mandatory to prevent session ID stealing through XSS attacks. However, if an XSS attack is combined with a **CSRF** attack, the requests sent to the web application will include the session cookie, as the browser always includes the cookies when sending requests. The **HttpOnly** cookie only protects the confidentiality of the cookie; the attacker cannot use it offline, outside of the context of an **XSS** attack.

While the **HttpOnly** attribute protects the confidentiality of sensitive cookies, it does not protect them from being overwritten. This is because a browser can only store a limited number of cookies for a domain. An attacker may use the cookie jar overflow attack to set a large number of cookies for a domain, deleting the original HttpOnly cookie from browser memory and allowing the attacker to set the same cookie without the flag.

```
HTTP/2.0 200 OK
Content-Type: text/html
Set-Cookie: cookie_name=value; HttpOnly


[page content]
```

Cookies Security: SameSite

SameSite defines a cookie attribute preventing browsers from sending a **SameSite** flagged cookie with cross-site requests. The main goal is to mitigate the risk of cross-origin information leakage, and to provide some protection against cross-site request forgery attacks.

The SameSite cookie attribute may have one of the following values:

- **SameSite**=**Strict**: The cookie is only sent if you are currently on the site that the cookie is set for. If you are on a different site and click a link to the site that the cookie is set for, the cookie is not sent with the first request.
- **SameSite**=**Lax**: The cookie is not sent for embedded content, but it is sent if you trigger top-level navigation, e.g. by clicking on a link to the site that the cookie is set for. It is sent only with safe request types that do not change state, such as GET.
- **SameSite**=**None**: The cookie is sent even for embedded content. Note that you can expect different browser behaviors when the SameSite attribute is not set.

```
HTTP/2.0 200 OK
Content-Type: text/html
Set-Cookie: cookie_name=value; SameSite=Strict


[page content]
```

Cookies Security: Domain and Path

The **Domain** cookie attribute instructs web browsers to only send the cookie to the specified domain and all subdomains. If the attribute is not set, by default the cookie will only be sent to the origin server. The Path cookie attribute instructs web browsers to only send the cookie to the specified directory or subdirectories (or paths or resources) within the web application. If the attribute is not set, by default the cookie will only be sent for the directory (or path) of the resource requested and setting the cookie.

It is recommended to use a narrow or restricted scope for these two attributes. In this way, the **Domain** attribute should not be set (restricting the cookie just to the origin server) and the Path attribute should be set as restrictive as possible to the web application path that makes use of the session ID.

Setting the **Domain** attribute to a value that is too permissive, such as example.com allows an attacker to launch attacks on the session IDs between different hosts and web applications belonging to the same domain, known as cross-subdomain cookies. For example, vulnerabilities in www.example.com might allow an attacker to get access to the session IDs from secure.example.com.

Additionally, it is recommended not to mix web applications of different security levels on the same domain. Vulnerabilities in one of the web applications would allow an attacker to set the session ID for a different web application on the same domain by using a permissive **Domain** attribute (such as example.com) which is a technique that can be used in session fixation attacks.

Although the **Path** attribute allows the isolation of session IDs between different web applications using different paths on the same host, it is highly recommended not to run different web applications (especially from different security levels or scopes) on the same host. Other methods can be used by these applications to access the session IDs, such as the document.cookie object. Also, any web application can set cookies for any path on that host.

Cookies are vulnerable to DNS spoofing/hijacking/poisoning attacks, where an attacker can manipulate the DNS resolution to force the web browser to disclose the session ID for a given host or domain.

```
HTTP/2.0 200 OK
Content-Type: text/html
Set-Cookie: cookie_name=value; Domain=codio.com; Path=/api


[page content]
```

Cookies Security: Expire and Max-Age

Session management mechanisms based on cookies can make use of two types of cookies, non-persistent (or session) cookies, and persistent cookies. If a cookie presents the **Max-Age** (that has preference over **Expires**) or **Expires** attributes, it will be considered a persistent cookie and will be stored on a device's disk by the web browser until the expiration time.

Typically, session management capabilities to track users after authentication make use of non-persistent cookies. This forces the session to disappear from the client if the current web browser instance is closed. Therefore, it is highly recommended to use non-persistent cookies for session management purposes, so that the session ID does not remain on the web client cache for long periods of time, where an attacker can obtain it. Other recommendations include:

- Ensure that sensitive information is not compromised by checking that a cookie is not persistent, encrypting it, and storing it only for the duration of the user's need
- Ensure that unauthorized activities cannot take place via cookie manipulation
- Ensure that the Secure flag is set to prevent accidental transmission over the web in a non-secure manner
- Determine if all state transitions in the application code properly check for cookies and enforce their use
- Ensure entire cookie is encrypted if sensitive data is persisted in the cookie
- Define all cookies being used by the application, their name, and why they are needed

```
HTTP/2.0 200 OK
Content-Type: text/html
Set-Cookie: cookie_name=value; Expires=Thu, 31 Oct 2021 07:28:00
GMT


[page content]
```