20. Cookies and Session Management

20. Cookies and Session Management

HTTP is a stateless protocol, which means each request and response is independent from all other requests and responses. However, many features on the web require maintaining some form of state. For example, when you log into your email account, you can stay logged in across many requests and responses. If you enable dark mode on a website and make subsequent requests to the website, you want the pages returned to have a dark background. If you're browsing an online shopping website, you want the items in your cart to be saved across many requests and responses. Browser and servers store HTTP cookies to support these features.

At a high level, you can think of cookies as pieces of data stored in your browser. When you make a request to enable dark mode or add an item to your shopping cart, the server sends a response with a `Set-Cookie` header, which tells your browser to store a new cookie. These cookies encode state that should persist across multiple requests and responses, such as your dark mode preference or a list of items in your shopping cart. In future requests, your browser will automatically attach the relevant cookies to a request and send it to the web server. The additional information in these cookies helps the web server customize its response.

20.1. Cookie Attributes

Every cookie is a name-value pair. For example, a cookie `darkmode=true` has name `darkmode` and value `true`.

For security and functionality reasons, we don't want the browser to send every cookie in every request. A user might want to enable dark mode on one website but not on another website, so we need a way to only send certain cookies to certain URLs. Also, as we'll see later, cookies may contain sensitive login information, so sending all cookies in all requests poses a security risk. These additional cookie attributes help the browser determine which cookies should be attached to each request.

- The `Domain` and `Path` attributes tell the browser which URLs to send the cookie to. See the next section for more details.
- The `Secure` attribute tells the browser to only send the cookie over a secure HTTPS connection.
- The `HttpOnly` attribute prevents JavaScript from accessing and modifying the cookie.
- The `expires` field tells the browser when to stop remembering the cookie.

20.2. Cookie Policy: Domain and Path

The browser sends a cookie to a given URL if the cookie's `Domain` attribute is a domain-suffix of the URL domain, and the cookie's `Path` attribute is a prefix of the URL path. In other words, the URL domain should end in the cookie's `Domain` attribute, and the URL path should begin with the cookie's `Path` attribute.

For example, a cookie with Domain=example.com and Path=/some/path will be included on a request to http://foo.example.com/some/path/index.html, because the URL domain ends in the cookie domain, and the URL path begins with the cookie path.

Note that cookie policy uses a different set of rules than the same origin policy. This has caused problems in the past.

20.3. Cookie Policy: Setting Domain and Path

For security reasons, we don't want a malicious website `evil.com` to be able to set a cookie with domain `bank.com`, since this would allow an attacker to affect the functionality of the legitimate bank website. To prevent this, the cookie policy specifies that when a server sets a cookie, the cookie's domain must be a URL suffix of the server's URL. In other words, for the cookie to be set, the server's URL must end in the cookie's `Domain` attribute. Otherwise, the browser will reject the cookie.

For example, a webpage with domain `eecs.berkeley.edu` can set a cookie with domain `eecs.berkeley.edu` or `berkeley.edu`, since the webpage domain ends in both of these domains.

This policy has one exception: cookies cannot have domains set to a top-level domain, such as `.edu` or `.com`, since these are too broad and pose a security risk. If `evil.com` could set cookies with domain `.com`, the attacker would have the ability to affect all `.com` websites, since this cookie would be sent to all `.com` websites. The web browser maintains a list of top-level domains, which includes two-level TLDs like `.co.uk`.

The cookie policy allows a server to set the `Path` attribute without any restrictions.[1]

Further reading: Cookies

20.4. Session Management

Cookies are often used to keep users logged in to a website over many requests and responses. When a user sends a login request with a valid username and password, the server will generate a new session token and send it to the user as a cookie. In future requests, the browser will attach the session token cookie and send it to the server. The server maintains a mapping of session tokens to users, so when it receives a request with a session token cookie, it can look up the corresponding user and customize its response accordingly.

Secure session tokens should be random and unpredictable, so an attacker cannot guess someone else's session token and gain access to their account. Many servers also set the `HttpOnly` and `Secure` flags on session tokens to protect them from being accessed by XSS vulnerabilities or network attackers, respectively.

It is easy to confuse session tokens and cookies. Session tokens are the values that the browser sends to the server to associate the request with a logged-in user. Cookies are how the browser stores and sends session tokens to the server. Cookies can also be used to save other state, as discussed earlier. In other words, session tokens are a special type of cookie that keep users logged in over many requests and responses.

[1] The lack of restriction on the `Path` attribute has caused problems in the past, as cookies are presented to the server and JavaScript as an unordered set of name/value pairs, but is stored internally as name/path/value tuples, so if two cookies with the same name and host but different path are present, both will be presented to the server in unspecified order.