Encrypt it Or Forget it

In this lesson, we'll look at the secure directive.

We'll cover the following

- Introduction
- Example

Introduction

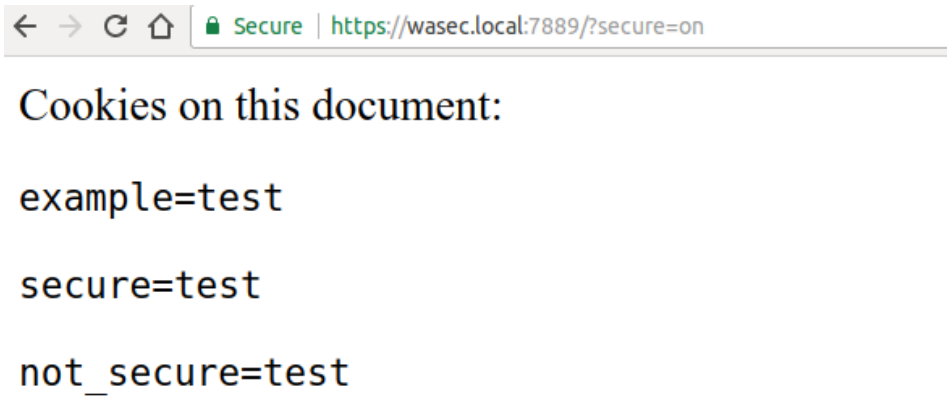
Cookies contain very sensitive information. If attackers can get a hold of a session ID, they can impersonate users by hijacking their sessions.

Most *session hijacking* attacks usually happen through a *man-in-the-middle* who can listen to the unencrypted traffic between the client and server and steal any information that's been exchanged. If a cookie is exchanged via HTTP, then it's vulnerable to MITM attacks and session hijacking.

To overcome the issue, we can use HTTPS when issuing the cookie and add the `Secure` flag to it. This instructs browsers to never send this cookie in plain HTTP requests.

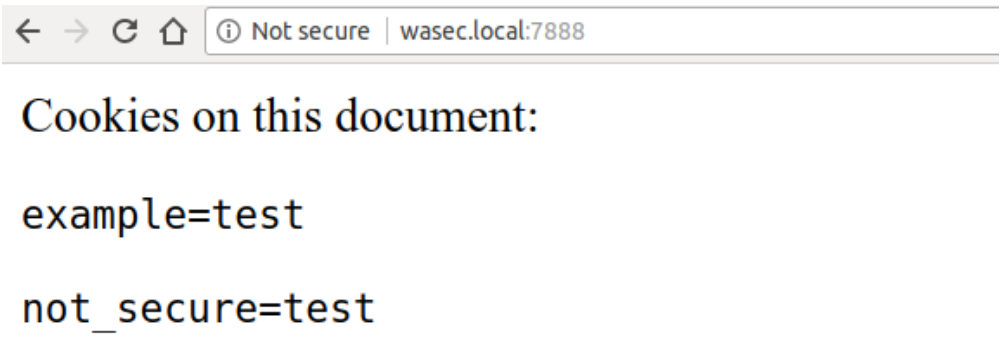
Example

Going back to our practical example at https://github.com/odino/wasec/tree/master/cookies, we can test this out by navigating to https://wasec.local:7889/?secure=on. The server sets two additional cookies, one with the `Secure` flag and one without



Cookies with secure flag

When we go back and navigate to the HTTP version of the site, we can clearly see that the `Secure` cookie is not available in the page, try navigating to wasec.local:7888:



Cookies without secure flag

We can clearly see that the HTTPS version of our app sets a cookie that's available to the HTTP one (the `not_secure` one), but the other cookie, flagged as `Secure`, is nowhere to be seen.

Marking sensitive cookies as `Secure` is an incredibly important aspect of cookie security. Even if you serve all of your traffic to HTTPS, attackers could find a way to set up a plain old HTTP page under your domain and redirect users there. Unless your cookies are `Secure`, they will have access to a very delicious meal.

In the next lesson, we'll look at the `HttpOnly` directive

In the next lesson, we'll look at the `httponly` directive.