JavaScript Can't Touch This

In this lesson, we'll study how the HttpOnly flag works.

We'll cover the following

As we've seen earlier, XSS attacks allow a malicious user to execute arbitrary JavaScript on a page. Considering that you could read the contents of the cookie jar with a simple `document.cookie`, protecting our cookies from untrusted JavaScript access is a very important aspect of hardening cookies from a security standpoint.

Luckily, the HTTP spec took care of this with the `HttpOnly` flag. By using this directive we can instruct the browser not to share the cookie with JavaScript. The browser then removes the cookie from the `window.cookie` variable, making it impossible to access the cookie via JS.

Example

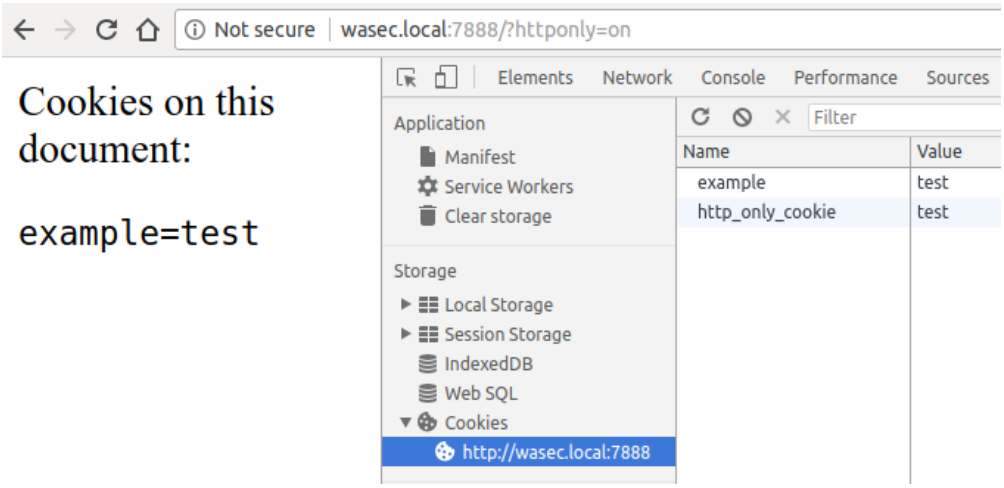
If we look at the example at https://x6jr4kg.educative.run/?httponly=on we can clearly see how this works.

```
var qs = require('querystring')
var url = require('url')
var fs = require('fs')

function server (req, res) {
  if (req.url === '/same-site-form') {
    return res.end(`
<html>
  <form action="http://wasec.local:7888/" method="POST">
    <input type="hidden" name="destination" value="attacker@email.com" />
    <input type="hidden" name="amount" value="1000" />
    <input type="submit" value="CLICK HERE TO WIN A HUMMER" />
  </form>
</html>
`)
  }
  let query = qs.parse(url.parse(req.url).query)

  if (query.clear === "on") {
```

```
    res.writeHead('302', {
      'Set-Cookie': [
        'example=a;Expires=Wed, 21 Oct 2015 07:28:00 GMT',
        `example_with_domain=a;Domain=wasec.local;Expires=Wed, 21 Oct 2015 07:28:00 GMT`,
        `supercookie=a;Expires=Wed, 21 Oct 2015 07:28:00 GMT`,
        `secure=a;Expires=Wed, 21 Oct 2015 07:28:00 GMT`,
        `not_secure=a;Expires=Wed, 21 Oct 2015 07:28:00 GMT`,
      ],
      'Location': '/'
    })
    res.end()
    return
  }

  let headers = {}
  headers['Set-Cookie'] = []

  if (!req.headers.host.startsWith('sub.wasec.local')) {
    headers['Set-Cookie'].push('example=test')

    if (query.domain === 'on') {
      headers['Set-Cookie'].push(`example_with_domain=test_domain_cookie;Domain=wasec.local`)
    }
  }

  if (query.super === 'on') {
    headers['Set-Cookie'].push(`supercookie=test;Domain=local`)
  }

  if (query.secure === 'on') {
    headers['Set-Cookie'].push(`secure=test;Secure`)
  }

  if (query.secure === 'on') {
    headers['Set-Cookie'].push(`not_secure=test`)
  }

  if (query.httponly === 'on') {
    headers['Set-Cookie'].push(`http_only_cookie=test;HttpOnly`)
  }

  if (query.samesite === 'on') {
    headers['Set-Cookie'].push(`same_site_cookie=test;SameSite=Lax`)
  }

  res.writeHead(200, headers)
  res.end(`
<html>
  <div id="output"/ >
  <script>
    let content = "none";
    if (document.cookie) {
      let cookies = document.cookie.split(';')
      content = ''
      cookies.forEach(c => {
        content += "<p><code>" + c + "</code></p>"
      })
    }
    document.getElementById('output').innerHTML = "Cookies on this document: <div>" + content + "</
  </script>
<html>
`)
```

```
}

const options = {
  key: fs.readFileSync(__dirname + '/../wasec.local-key.pem'),
  cert: fs.readFileSync(__dirname + '/../wasec.local.pem')
};

require('http').createServer(server).listen(7888)
require('https').createServer(options, server).listen(7889)
```

The browser has stored the cookie (as seen on the DevTools) but won't share it with JavaScript.



Cookie not shared with JavaScript

The browser will then keep sending the cookie to the server in subsequent requests, so the server can still keep track of the client through the cookie. The trick, in this case, is that the cookie is never exposed to the end-user, and remains private between the browser and the server.

The `HttpOnly` flag helps mitigate XSS attacks by denying access to critical information stored in a cookie: using it makes it harder for an attacker to hijack a session.

ℹ Circumventing `HttpOnly`

In 2003, researchers found an interesting vulnerability around the `HttpOnly` flag, Cross-Site Tracing (XST).

In a nutshell, browsers wouldn't prevent access to `HttpOnly` cookies when using the `TRACE` request method. While most browsers have now disabled this method, my recommendation would be to disable `TRACE` at your webserver's level, returning the `405 Not allowed` status code

In the next lesson, we'll study the `SameSite` flag.