Session Management Testing

→ outlines :

1. Bypassing Session Management Schema
2. Cookie attribute Testing.
3. Exposed Session Variables
4. logout functionality.
5. Session Timeout
6. Session puzziling

* In this test, the tester wants to check that cookies and other session tokens are created in a secure and unpredictable way. An attacker who is able to predict and forge a weak cookie can easily hijack the sessions of legitimate users.

* Bypassing Session Management Schema

1. Cookie collection
2. cookie reverse engineering
3. Cookie manipulate

conditions:

1. if cookie can be forced over encrypted transport
2. are any cookie persistent?
3. expire times.

* if cookie does not expired after logout, then there is vulnerability

* cookie attribute

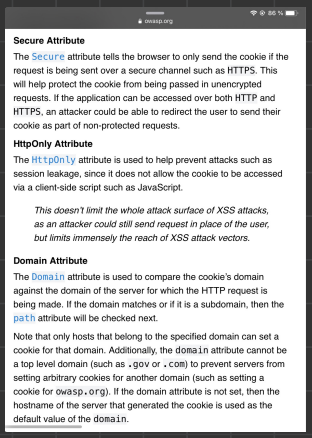
The most used session storage mechanism in browsers is cookie storage. Cookies can be set by the server, by including a Set-Cookie header in the HTTP response or via JavaScript. Cookies can be used for a multitude of reasons, such as:

- session management
- personalization
- tracking

In order to secure cookie data, the industry has developed means to help lock down these cookies and limit their attack surface. Over time cookies have become a preferred storage mechanism for web applications, as they allow great flexibility in use and protection.

The means to protect the cookies are:

- Cookie Attributes
- Cookie Prefixes



Secure Attribute

The `Secure` attribute tells the browser to only send the cookie if the request is being sent over a secure channel such as `HTTPS`. This will help protect the cookie from being passed in unencrypted requests. If the application can be accessed over both `HTTP` and `HTTPS`, an attacker could be able to redirect the user to send their cookie as part of non-protected requests.

HttpOnly Attribute

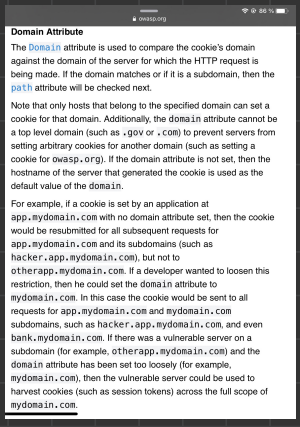
The `HttpOnly` attribute is used to help prevent attacks such as session leakage, since it does not allow the cookie to be accessed via a client-side script such as JavaScript.

> *This doesn't limit the whole attack surface of XSS attacks, as an attacker could still send request in place of the user, but limits immensely the reach of XSS attack vectors.*

Domain Attribute

The `Domain` attribute is used to compare the cookie's domain against the domain of the server for which the HTTP request is being made. If the domain matches or if it is a subdomain, then the `path` attribute will be checked next.

Note that only hosts that belong to the specified domain can set a cookie for that domain. Additionally, the `domain` attribute cannot be a top level domain (such as `.gov` or `.com`) to prevent servers from setting arbitrary cookies for another domain (such as setting a cookie for `owasp.org`). If the domain attribute is not set, then the hostname of the server that generated the cookie is used as the default value of the `domain`.

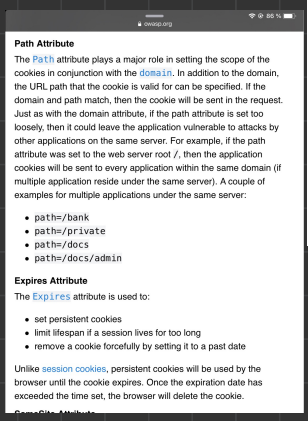


Domain Attribute

The `Domain` attribute is used to compare the cookie's domain against the domain of the server for which the HTTP request is being made. If the domain matches or if it is a subdomain, then the `path` attribute will be checked next.

Note that only hosts that belong to the specified domain can set a cookie for that domain. Additionally, the `domain` attribute cannot be a top level domain (such as `.gov` or `.com`) to prevent servers from setting arbitrary cookies for another domain (such as setting a cookie for `owasp.org`). If the domain attribute is not set, then the hostname of the server that generated the cookie is used as the default value of the `domain`.

For example, if a cookie is set by an application at `app.mydomain.com` with no domain attribute set, then the cookie would be resubmitted for all subsequent requests for `app.mydomain.com` and its subdomains (such as `hacker.app.mydomain.com`), but not to `otherapp.mydomain.com`. If a developer wanted to loosen this restriction, then he could set the `domain` attribute to `mydomain.com`. In this case the cookie would be sent to all requests for `app.mydomain.com` and `mydomain.com` subdomains, such as `hacker.app.mydomain.com`, and even `bank.mydomain.com`. If there was a vulnerable server on a subdomain (for example, `otherapp.mydomain.com`) and the `domain` attribute has been set too loosely (for example, `mydomain.com`), then the vulnerable server could be used to harvest cookies (such as session tokens) across the full scope of `mydomain.com`.



Path Attribute

The `Path` attribute plays a major role in setting the scope of the cookies in conjunction with the `domain`. In addition to the domain, the URL path that the cookie is valid for can be specified. If the domain and path match, then the cookie will be sent in the request. Just as with the domain attribute, if the path attribute is set too loosely, then it could leave the application vulnerable to attacks by other applications on the same server. For example, if the path attribute was set to the web server root `/`, then the application cookies will be sent to every application within the same domain (if multiple application reside under the same server). A couple of examples for multiple applications under the same server:

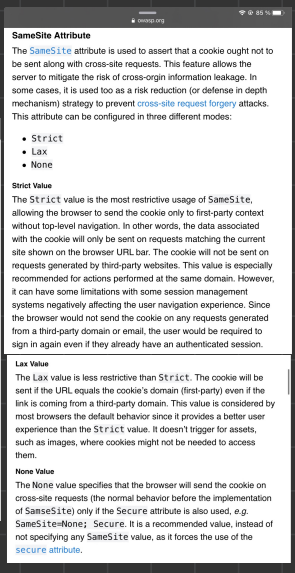
- `path=/bank`
- `path=/private`
- `path=/docs`
- `path=/docs/admin`

Expires Attribute

The `Expires` attribute is used to:

- set persistent cookies
- limit lifespan if a session lives for too long
- remove a cookie forcefully by setting it to a past date

Unlike session cookies, persistent cookies will be used by the browser until the cookie expires. Once the expiration date has exceeded the time set, the browser will delete the cookie.



SameSite Attribute

The `SameSite` attribute is used to assert that a cookie ought not to be sent along with cross-site requests. This feature allows the server to mitigate the risk of cross-origin information leakage. In some cases, it is used too as a risk reduction (or defense in depth mechanism) strategy to prevent cross-site request forgery attacks. This attribute can be configured in three different modes:

- `Strict`
- `Lax`
- `None`

Strict Value

The `Strict` value is the most restrictive usage of `SameSite`, allowing the browser to send the cookie only to first-party context without top-level navigation. In other words, the data associated with the cookie will only be sent on requests matching the current site shown on the browser URL bar. The cookie will not be sent on requests generated by third-party websites. This value is especially recommended for actions performed at the same domain. However, it can have some limitations with some session management systems negatively affecting the user navigation experience. Since the browser would not send the cookie on any requests generated from a third-party domain or email, the user would be required to sign in again even if they already have an authenticated session.

Lax Value

The `Lax` value is less restrictive than `Strict`. The cookie will be sent if the URL equals the cookie's domain (first-party) even if the link is coming from a third-party domain. This value is considered by most browsers the default behavior since it provides a better user experience than the `Strict` value. It doesn't trigger for assets, such as images, where cookies might not be needed to access them.

None Value

The `None` value specifies that the browser will send the cookie on cross-site requests (the normal behavior before the implementation of `SamseSite`) only if the `Secure` attribute is also used, *e.g.* `SameSite=None; Secure`. It is a recommended value, instead of not specifying any `SameSite` value, as it forces the use of the `secure` attribute.

1. Secure : this attribute tells browser to only send cookie if request is being sent over secure channel.
   ↳ look in https website if they are not using secure flag in cookie then there is a vulnerability.

2. http only : this attribute prevent attacks such xss, so no client side script can access cookie.
   ↳ if there is no http only flag then there is a vulnerability

3. domain : this attribute use to compare the domains

4 expires : cookie expired when session is close.
   ↳ if cookie does not expired, then it is a bug vulnerability

✗ To sum up: See cookie, if there is no secure flag or httponly.

exposed session variables

Testing for Transport Vulnerabilities
All interaction between the Client and Application should be tested at least against the following criteria.

- How are Session IDs transferred? e.g., GET, POST, Form Field (including hidden fields)
- Are Session IDs always sent over encrypted transport by default?
- Is it possible to manipulate the application to send Session IDs unencrypted? e.g., by changing HTTP to HTTPS?
- What cache-control directives are applied to requests/responses passing Session IDs?
- Are these directives always present? If not, where are the exceptions?
- Are GET requests incorporating the Session ID used?
- If POST is used, can it be interchanged with GET?

logout functionality

you can report this vulnerability as session not expired after logout

```
1 Login to hackerone.
2 Capture any request.
3 Send it to burp intruder.
4 Logout from hackerone.
5 Now start intruding that captured
  request,which is carrying the old destroyed
  session.
6 Try log in from other device.
7 Every time burp intruder sends a request,the
  present logged in account will be logged out
  automatically. So if you keep continue
  intruding,that user will never be able to
  login to his own ID.
```

session timeout

```
3 steps
4 1. login into any application
5
6 2. get a cup of cofee , or lunch ,
  or go home for the vening
7
8 3. when u return , try to perform
  an actiity that requires previous
  authentication
9
```

* Session Management Vulnerability on password reset or on etc

Steps:

① reset password
② use link again to reset password again
③ if it worked, then it is a vulnerability.

* another senario:
① get two reset password link
② try with the first one
③ then with the second one
④ if it worked, it is a vulnerability.

* Another Senario:

```
1 1.login
2
3 2. change ur email id
4
5 3. forgot password page
6
7 4. use old email to reset password
8
9 5. if you will be able to reset
  then there is a vulnerabili
```