Cookie Security Attributes.

- HttpOnly: possible values are (true, false)
  - HttpOnly will instruct the browser, to not allow JavaScript to modify the cookies.
  - If HttpOnly is set to true, then the JavaScript won't be able to identify in the cookies.
  - Protections from stealing the cookies throw XXS attacks.
- Secure: possible values are (true, false)
  - Is to make the transmission of the cookies over encrypted HHTPS connection while the client is on the Website.
- SameSite: possible values / levels of protection are (None, Lax,Strict)
  - Offer the protection from CSRF attacks.
  - None => no protection
  - Lax => if the fram use the method GET then the cookies will only be sent on cross-site request
  - Strict => the cookies won't be sent across site

Combining the attributes HttpOnly and Secure wiil ensure the cookies and the protection from hijacking.

Example of Cookie Attrinutes:

"httpOnly":false,"secure":false,""sameSite":"Lax"

Hijacking

Hacker search for an existing network session or a suspended or saved session context and taking over it, just few minutes are enough to do bad things.

Clickjacking:

Let the victim click on something to do bad things.

CSRF:

Let the victim to click on a link to send a bad request.

Solution: using randomized tokens which also called as anti CSR tokens this will be generated randomly by the Web application in the backend and sent to the website, on every request it's validated so the other website cannot make cross-origin-cross-domain requests because it does not have any knowledge of the new token.

Cross Site Scripting – XXS:

Injection of data from untrusted sources throw web request.

X-Frame-Options - XFO:

XFO is a http header which is used to configure <iframe> in our web applications

<iframe>: is basically a html element which we use to render any inserted link onto the website

Possible values are:

- DENY:
  - Will block any url from being rendered in the <iframe>

- SAMEORIGIN:
  - Allow rendering of url, which are from the same host, in the <iframe>

Factors that can affect the security of the web applications.

User – Agent:

Is one of the most important headers in http request, it describes the device or the technology that the client use to access the website

User-Agent contain a lot of details like device type, version number, OS(operating system) numbers, the operating systems and the browser that has been used.

For example:

{"ua":"Mozilla/5.0 (iPhone; CPU iPhone OS 14_4 like Mac OS X) AppleWebKit/605.1.15 (KHTML, like Gecko) Version/16.0 Mobile/15E148 Safari/604.1","browser":{"name":"Mobile Safari","version":"16.0","major":"16"},"engine":{"name":"WebKit","version":"605.1.15"},"os":{"name":"iOS","version":"14.4"},"device":{"vendor":"Apple","model":"iPhone","type":"mobile"},"cpu":{}}

Using unprotected user-agent will lead to a specific type of attack, for example firefox desktop browser is safer than the firefox mobile browser.

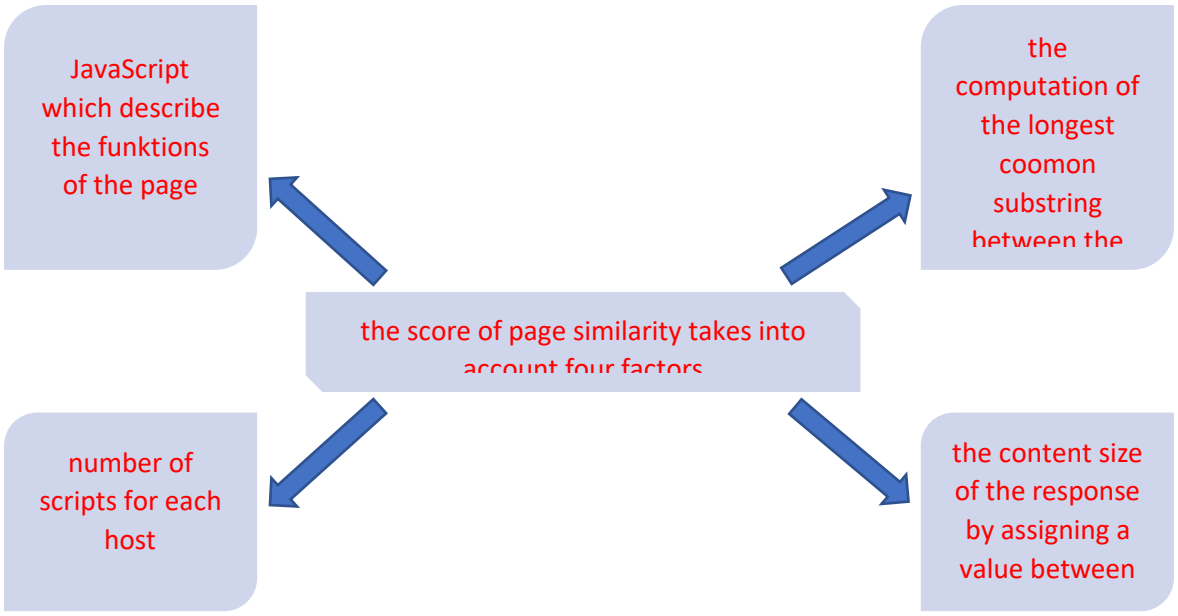
Vantage Point:

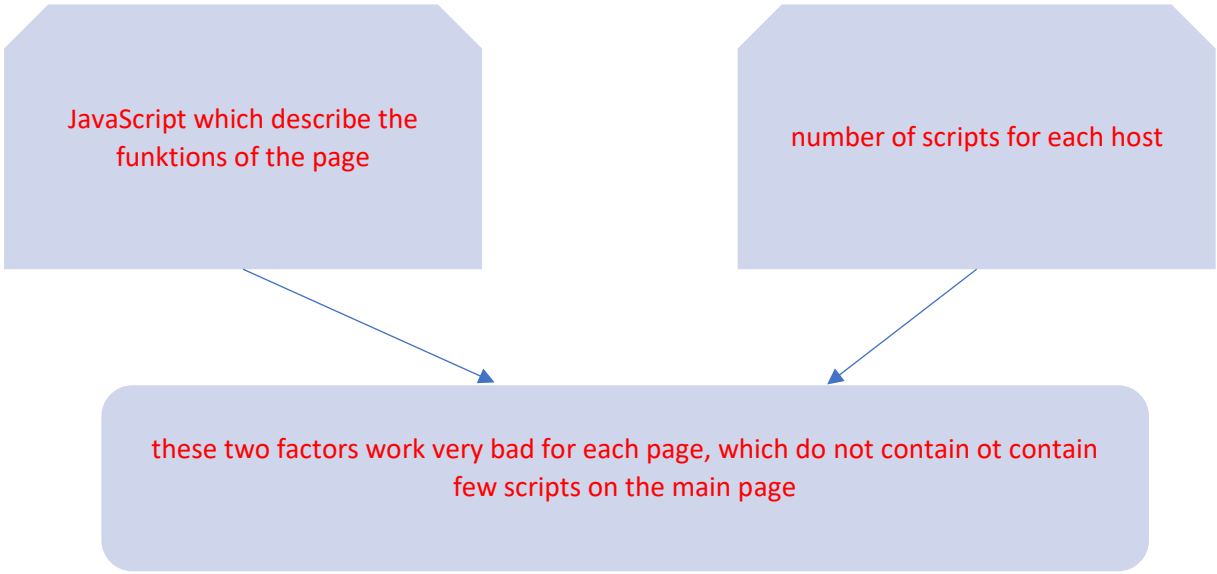
Accessing a specific website from different vantage points, can lead to different security policies.

Client Configuration:

The Interaction between the client and the website with different configuration settings, for example the language of the client (Accept-Language header)

Page Similarity.





Combining the last four factors and compute their average and we take the resulting value (between 0 and 1) to determine the similarity between pages.

Two pages are similar if the similarity score is at bigger or equal to 0.8

Different pages under the same host could force different HSTS policies for the same object (host).

The relation x $<_{\sim m}$ y means that x is no more secure than y.

$r_1 <_{\sim ck} r_2$, where r_1 and r_2 are two responses, if all cookies **c** in both response fulfilled the following conditions:

- If c marked as HttpOnly in one response, then it should be also marked as HttpOnly in the other one.
- If c marked as Secure in one response, then it should be also marked as Secure in the other one.
- If c marked as SameSite in one response, then it should be also marked as SameSite in the other one, but at least with the same level of protection.

WE define CSP as a safe CSP if one of the following directives 'script-src' or 'default-src' is included in the SCP policy and the <source> fulfilled the following conditions:

- 'unsafe-inline' is not included in the <source>, if <source> does not contain nonces or hashes
- 'unsafe-inline' is included in the <source>, if <source> contains nonces or hashes
- Wildcard* any Protocol like http: , https:, or data are not included in the <source>, if <source> does not contain the keyword 'strict-dynamic'
- Wildcard* any Protocol like http: , https:, or data are included in the <source>, if <source> contains the keyword 'strict-dynamic'

CSP Nonces: nonces are numbers used once and basically every time a page is rendered this random number will be generated, so it's generated on every page load and it's made to be random and not guessable, it's included on the header and also on all of our inline scripts:

For example :

content-security-policy: script-src: 'nonce-fuzzer11' 'self' ahamd.com 'unsafe-inline'

<html>

<body>

….....

<script nonce=”fuzzer11”>

….....

</script>

….....

</body>

</html>

Nonce ensure the protection from inline script attacks.

Strict Transport Security.

Http Strict Transport Security – HSTS.

HSTS directives:

- ❑ max-age: Duration in seconds to automatically request site over https, if a user hit http://www.website.com and the website support only https requests, then the browser will automatically redirect the user to the https version of website.com
- ❑ Include SubDomain: Automatically apply to all sub domains, it tells the browser to apply this automatic https redirection to all sub domains as well as the root domain itself.
- ❑ HSTS preloading to prevent SSL Stripping attack: the way preloading work as there is a static list of HSTS website that are maintained locally by every browser, and it contains a list of websites that no matter what should always be visited over https even on the first visit.

Classes of responses according to the value of max-age:

- HSTS deactivated for the host of the responses, if mas-age equal to zero.
- If HSTS does not appear in the responses, HSTS will not be activated but also not forcibly deactivate it.
- Max-age for less than one year of protection => responses does not qualify the host for preload list inclusion.
- Max-age for more than one year of protection => responses qualify the host for preload list inclusion.

Let assume we have two responses r_1 and r_2, we say that $r_1 <_{\sim hsts} r_2$ if:

- The class of response r_1 at most equal to the class of r_2.
- If the response r_1 contain the directive includeSubDomain, then r_2 should also contain it.
- If the response contains the directive preload, then should also contain it.

Cach Controll:

Cach control headers tell the browser whether or not the server wants the browser to store copies of the files usually caching is desired we want to make sure that the browser keeps copies of JavaScript, styles, images and pictures, so that it does not need to download these resources multiple times, that would cause the server to have to work a lot harder to send duplicate files.

Why we need cache control ?

To prevent caching files, which might contain sensitive data, for example we want to prevent the browser from caching the cookies.

- ❑ Allow caching of eligible mime-types except when response contain cookies.

=> cache-control: no-cache=”set-cookie”

- ❑ Do not allow caching of other content

=> cache-control: no-store, no-cache

pragma: no-chache

no-cache: dose not mean “don’t cache”, it means it must check with the server before using the cached resource.

no-store: tells the browser bot to cache it at all.

must-revalidate: does not mean “must revalidate”, it means the local resource can be used, if it’s younger than the provided max-age, otherwise must revalidate. (Hint: usually ‘must-revalidate’ used with ‘max-age’)

private: the response can be stored only in a private cache, like local caches in browsers.

x-cache: HIT means that your request was served by CDN, not origin servers, CDN is a special network designed to cache content, so that the user request served faster.

Different response headers could be caused by caching, because of the different values of the cache-control and x-cache in all headers and this lead to Intra-Test Inconsistencies.

Flawed User-Agent parsing, or wrong handling of the parsed browser information is the most important reason that lead to Inter-Test Inconsistencies

SSL Stripping attack:

If a user made a request with http to a website but the website support only https requests, the attacker will be in the middle of the connection between the client and the server, so the attacker will receive the https request and send the same request to the server but this time with https, the server will respond with the website over https and the attacker will receive it and decrypt it and pass it along to the client over http, let assume that in the last https session the webserver asks the client for sensitive information to login, so the attacker will send that to the client over http and the client will send this information to the attacker.

Same Origin Policy:

When we talk about origin, we talk about URLs, we say that two URLs have the same origin, if the scheme, the host and the port are the same in both URLs.

For example,

Same origin policy covers what a script in one origin is allowed to do.

What determines the origin of a script is the document in which it lives in, for example: if the <script> pulled from another origin to the website ahmad.com, then the origin of the script is ahmad.com