


HTTP Security Headers



Subgroup = 13.1 (CS)

Web Security

Lab Sheet 04

Year 2, Semester 2

Abeywickrama O.D.

HTTP Security Headers.

- When a user visits a site through his/her browser, the server responds with HTTP response headers. These headers tell the browser how to behave during communication with the site. These headers mainly comprise of metadata.
- Or
- HTTP security headers are a subset of HTTP headers that are sent between a web client (typically a browser) and a server to describe the security aspects of HTTP communication. HTTP security headers are HTTP headers that are indirectly related to privacy and security.



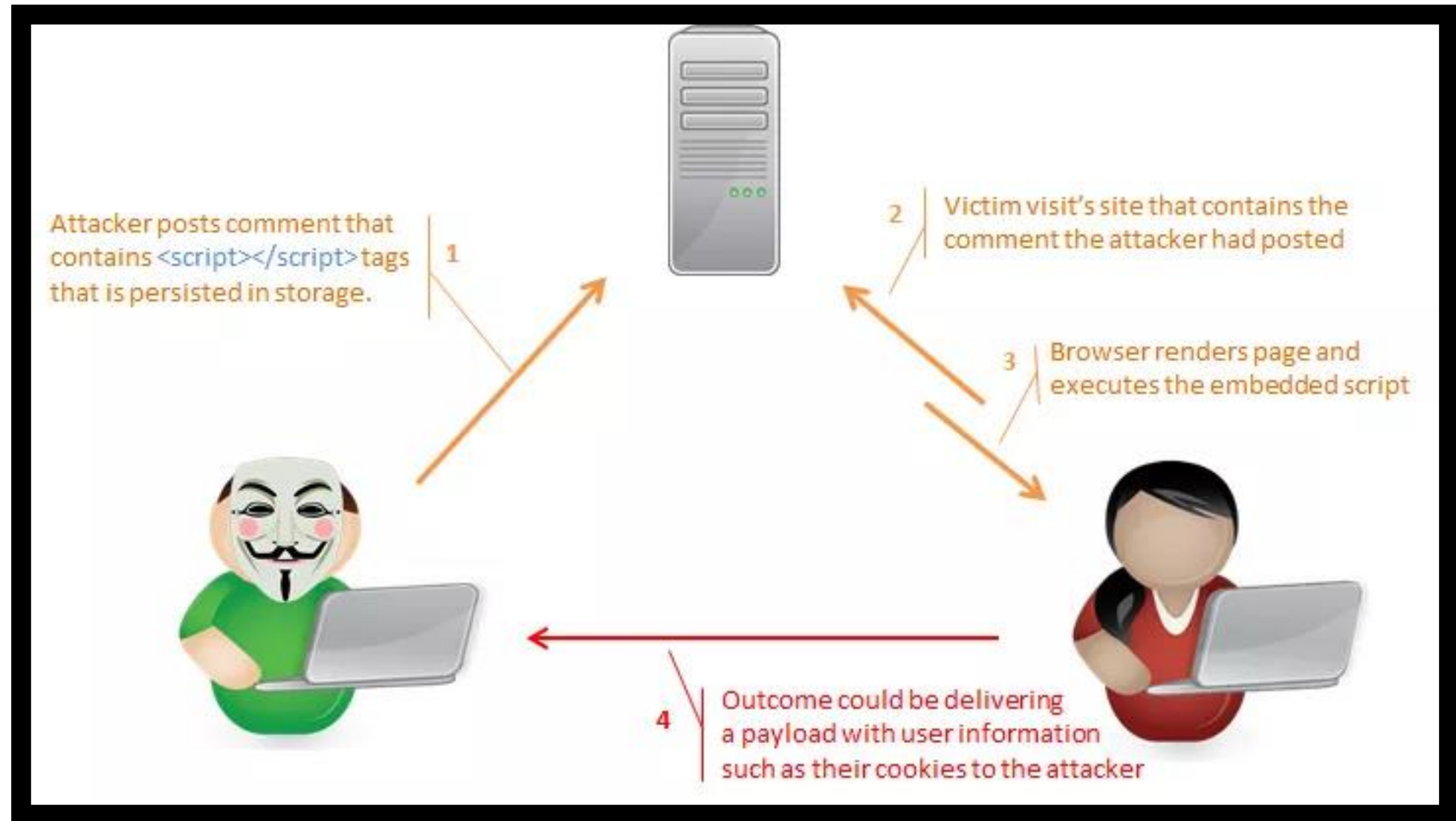
**HTTP
SECURITY
HEADERS**

Cross-Site Scripting Protection (X-XSS).

Cross-Site Scripting Protection (X-XSS).

- The X-XSS-Protection header is used to enable current web browsers' cross-site scripting (XSS) filter.
- This is normally turned on by default, however enabling it will make it mandatory.
- Internet Explorer 8 and above, Chrome, and Safari are all compatible with it.
- Set this header to the following value to enable XSS protection and advise the browser to block, rather than sanitize, the response if a malicious script is injected from user input.
- Cross-site scripting, often known as XSS, is a method of injecting code into a user's browser to perform operations on behalf of a website. The user may detect this at times, but it may sometimes go unnoticed in the background.

How an XSS attack works.



How an X-XSS attack works.

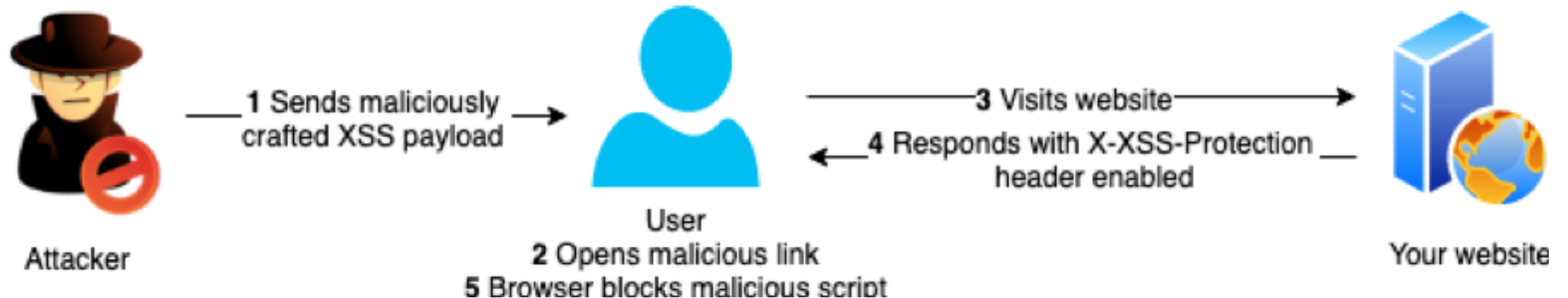
- If the headers are specified on the server, the X-XSS-Protection security header can be delivered to the user's browser. It has three alternatives that can be selected based on the situation.
 - ❖ X-XSS-Protection: 0; Disables the filter entirely. More on why this is used in the shortcomings section
 - ❖ X-XSS-Protection: 1; Enables the filter but only sanitizes the malicious script
 - ❖ X-XSS-Protection: 1; mode=block Enables the filter and completely blocks the page
- When you visit a website, the server will return the headers, as well as the HTML and any other information needed to view the page. The headers will be seen by the browser, and it will follow their regulations. Here's an example of a response that's been stripped of some superfluous headers.


```
1 HTTP/1.1 200 OK
2 Date: Thu, 01 Aug 2019 14:29:22 GMT
3 Server: Apache
4 X-FRAME-OPTIONS: DENY
5 X-XSS-Protection: 1; mode=block
6 X-Content-Type-Options: nosniff
7 Cache-Control: max-age=600
8 Strict-Transport-Security: max-age=31536000
```

- If your web application is vulnerable to cross-site scripting, this control would prevent the vulnerability from affecting your users.

- **Below is the narrative for each flow.**

- The attacker sends a malicious link to a user after identifying a cross-site scripting vulnerability on your web site. <https://your-website.com/search.php?>
- The user clicks the link and visits the website
- The website begins to load
- The web server responds with the headers it wants the browser to abide by
- The browser sees the X-XSS-Protection header set to block and blocks the attack





Why is it important.

- When all users are anonymous and all information is public, the impact is frequently minor in a brochureware application.
 - The impact on an application that stores sensitive data, such as banking transactions, emails, or medical records, is frequently significant.
 - The impact will usually be serious if the affected user has elevated rights within the application, allowing the attacker to gain complete control of the vulnerable application and compromise all users and their data.
-

Which Vulnerabilities can it prevent.

- **Reflected XSS**, *where the malicious script comes from the current HTTP request.*
- **Stored XSS**, *where the malicious script comes from the website's database.*
- **DOM-based XSS**, *where the vulnerability exists in client-side code rather than server-side code*

Advantages & Disadvantages

Advantages	Disadvantages
1. Filters are easy to implement in a web application.	1. But this filtering can be easily bypassed as XSS vectors are not finite and cannot be maintained.
2. Reduces XSS possibilities to a very good extent.	2. More often this works by accepting unsafe or unsensitized HTML, parses them and constructs a safe HTML, and responds back to the user.
3. If done correctly, contextual encoding eliminates XSS risk completely.	3. It treats all user data as unsafe. Thus, irrespective of the user data being safe or unsafe, all HTML content will be encoded and will be rendered as plain text.
4. Input validation not only reduces XSS but protects almost all vulnerabilities that may arise due to trusting user content.	4. Regular expression testing is performance intensive. All parameters in all requests to a server must be matched against a regular expression.

Referrer-Policy..

Referrer-Policy..

- Referrer-Policy is a security header that can (and should) be sent from your website's server to a client.
- When a user follows a link that takes them to another page or website, the Referrer-Policy instructs the web browser on how to handle referrer information that is given to the website.
- The Referrer-Policy can be set to not send any URL information, some information, or the entire URL route to the destination site.
- It is best to have a policy in place. The policy can be implemented in a variety of ways, including website code (PHP, etc).



How it works.

- no-referrer

The **Referrer** header will be completely skipped through. Requests are not transmitted with any referrer information.

- no-referrer-when-downgrade

Send the origin, path, and query string in Referrer when the protocol security level stays the same or improves (HTTP→HTTP, HTTP→HTTPS, HTTPS→HTTPS). Don't send the Referrer header for requests to less secure destinations (HTTPS→HTTP, HTTPS→file).

- Origin

Send the origin (only) in the Referrer header. For example, a document at <https://example.com/page.html> will send the referrer <https://example.com/>.

- origin-when-cross-origin

When making a same-origin request, send the origin, path, and query string to the same protocol level. For cross-origin requests and requests to less secure sites, send origin (alone).

- same-origin

Send the origin, path, and query string for same-origin requests. Don't send the Referrer header for cross-origin requests.

- strict-origin

Send the origin (only) when the protocol security level stays the same (HTTPS→HTTPS). Don't send the Referrer header to less secure destinations (HTTPS→HTTP).

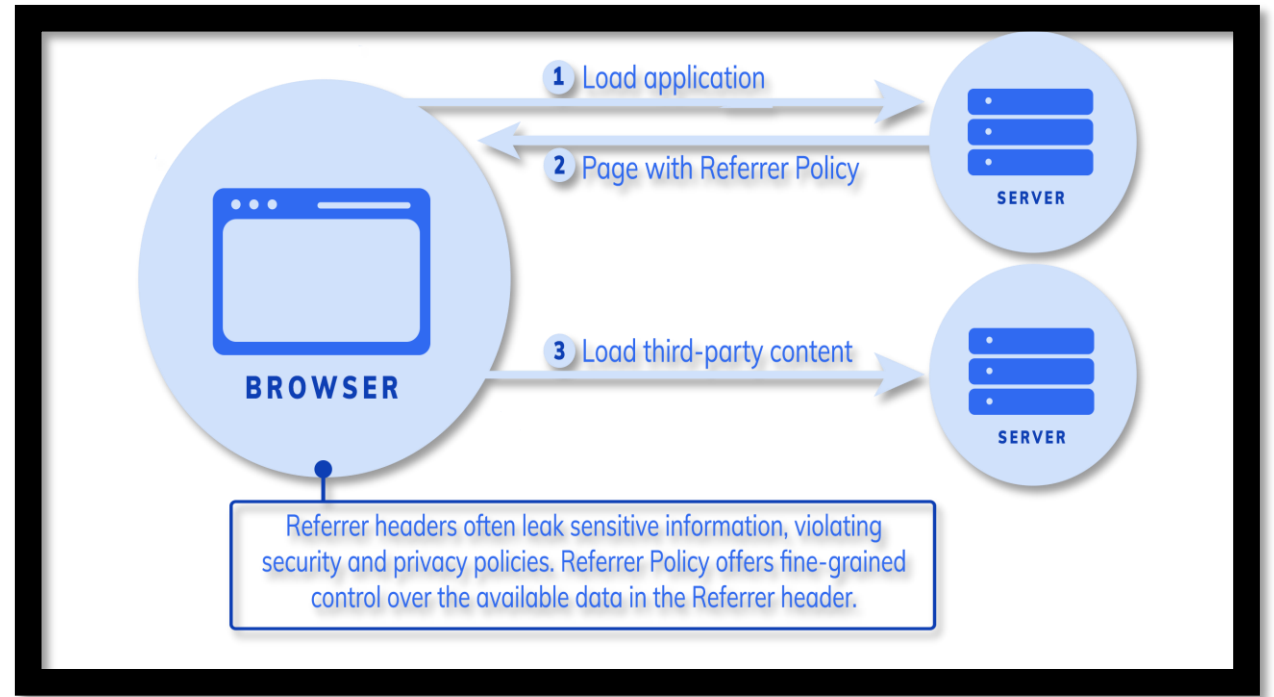
- strict-origin-when-cross-origin (default)

When making a same-origin request, include the origin, path, and querystring. Send the origin (only) for cross-origin requests when the protocol security level remains the same (HTTPS to HTTPS). Sending the Referrer header to less secure locations is not recommended.


- unsafe-url

Send the origin, path, and query string when performing any request, regardless of security.

How it works.



- Referrer Policy allows you to regulate the value of the Referrer in incoming HTTP/S requests.
- The origin of a user's visit to a new website is defined as the referrer value.
- Today, web search engines like Google mostly use this figure for visitor analysis and determining where a site's traffic originates.

A vertical column of blue dots on the left side of the slide.

Why is it important..



This section does not serve as a guideline.



A protected resource can prevent referrer leaking by setting its Referrer-Policy header to no-referrer:



Referrer-Policy: no-referrer

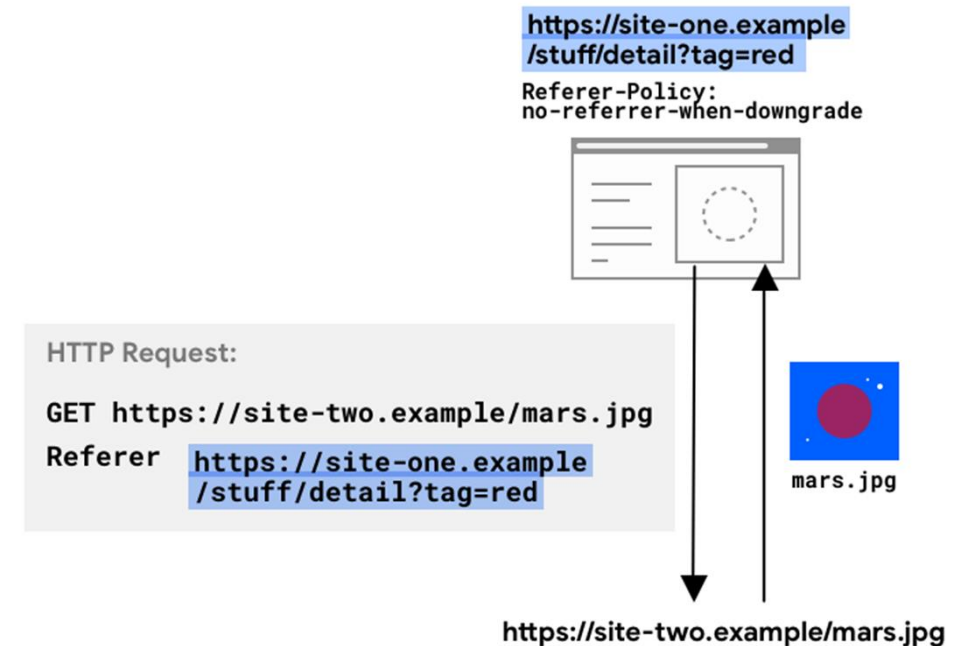


This will result in an empty Referrer [sic] header on all requests sent from the protected resource's context.

Which Vulnerabilities can it prevent.

- Insecure Referrer Policy

Referrer Policy governs the behavior of the Referrer header, which indicates the request's origin or web page URL. The web application has an insecure Referrer Policy setup, which could expose user information to third-party websites.

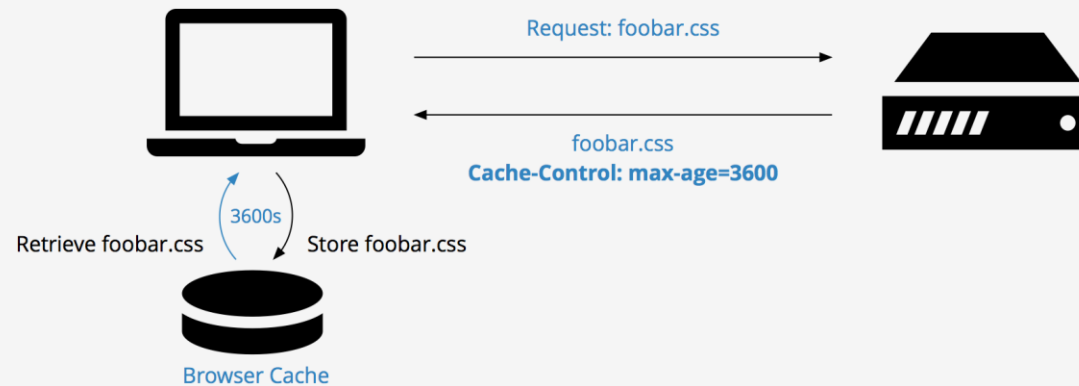


Advantages & Disadvantages..

Advantages	Disadvantages
1) Style, image, script load, and form submission requests will all include a referrer header.	1) not supported equally

Cache-Control..

Cache-Control..

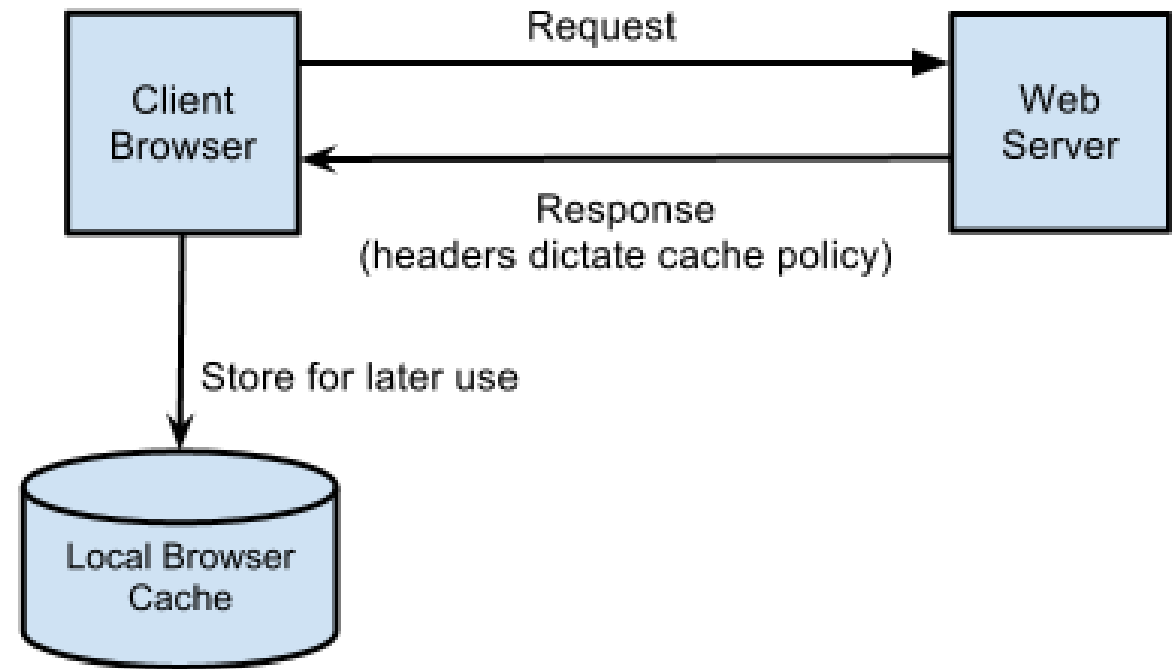


Cache-Control

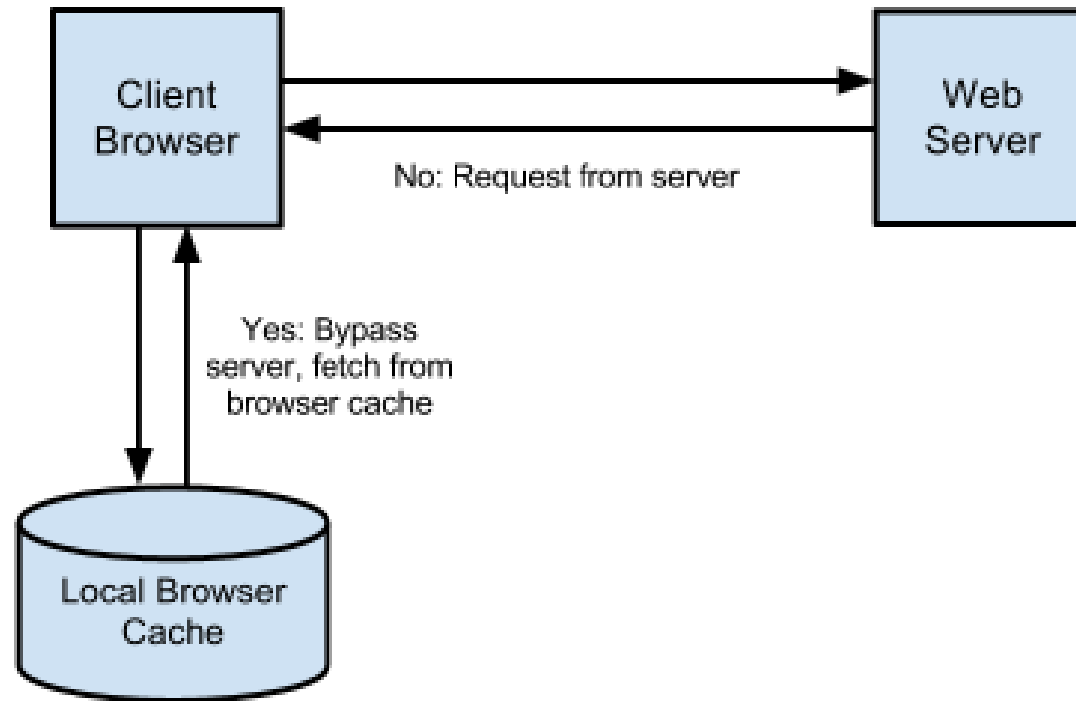
- Cache-control is an HTTP header that specifies how the browser caches data.
- In a nutshell, when someone visits a website, their browser saves certain resources in a cache, such as photos and page data.
- When the user returns to the same page, cache-control determines whether those resources will be loaded from the user's local cache or whether the browser will need to issue a request to the server for new ones.
- A rudimentary grasp of browser caching and HTTP headers is required to understand cache-control in greater depth.
- Key-value pairs separated by a colon form the headers. The 'key' for cache-control is always 'cache-control,' which is the part to the left of the colon.
- The 'value' can be one or more comma-separated values for cache control, and it can be located to the right of the colon.

How it works..

- HTTP caching is when a browser saves local copies of web resources for speedier retrieval when the resource is needed again.
- Cache headers can be included to the response as your application provides resources, indicating the preferred cache behavior.

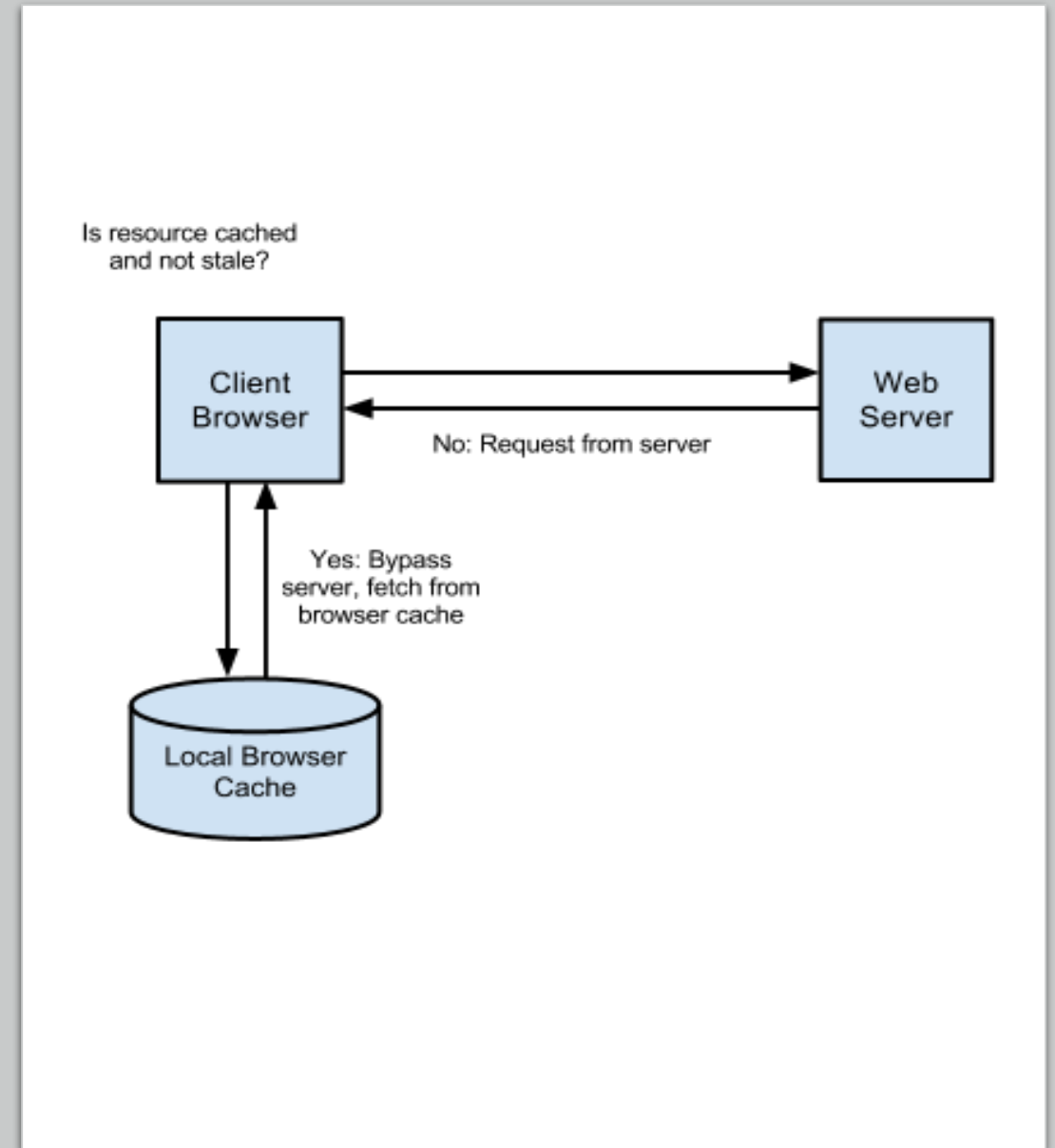


Is resource cached
and not stale?



- When an item is entirely cached, the browser may choose to utilize its own cached copy instead of contacting the server:

- CSS stylesheets from your application, for example, don't need to be downloaded again during the user's session once they've been downloaded by the browser.
- This is true for a wide range of asset types, including JavaScript files, photos, and even static dynamic material that changes seldom.
- It is advantageous for the user's browser to cache this file locally and use that copy if the resource is requested again in these circumstances.
- An application can manage this caching behavior and reduce server load by utilizing HTTP cache headers.



- The **Cache-Control header** is the most critical to set because it effectively turns on browser caching. The browser will cache the file for as long as indicated if this header is present and set to a value that allows caching.
- The browser will re-request the file on each subsequent request if this header is missing.
- **Public resources** can be cached not just by the end-browser, user's but also by any intermediate proxies serving a large number of other users.

```
Cache-Control:public
```

- **Private** resources are bypassed by intermediate proxies and can only be cached by the end-client.

```
Cache-Control:private
```

- The **Cache-Control header's** value is a composite value that indicates whether the resource is public or private, as well as the maximum period of time it can be cached before being considered stale. The **max-age** value specifies how long the resource should be cached (in seconds).

```
Cache-Control:public, max-age=31536000
```

- The **Cache-Control** header enables client-side caching and specifies a resource's **maximum age**, whereas the **Expires** header specifies a period when the resource is no longer valid.
- When used in conjunction with the **Cache-Control** header, **Expires** simply specifies a date after which the cached resource is no longer valid. The browser will request a new copy of the resource from this point on. Until then, the local cached copy of the browser will be used:



If both `Expires` and `max-age` are set `max-age` will take precedence.

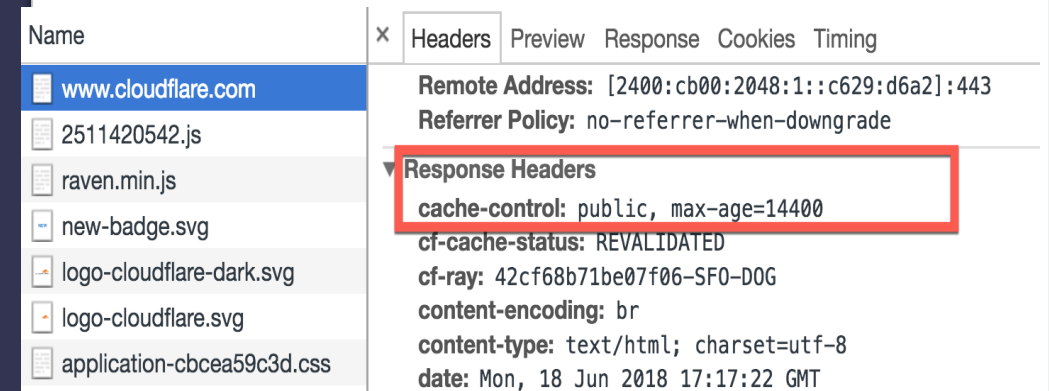
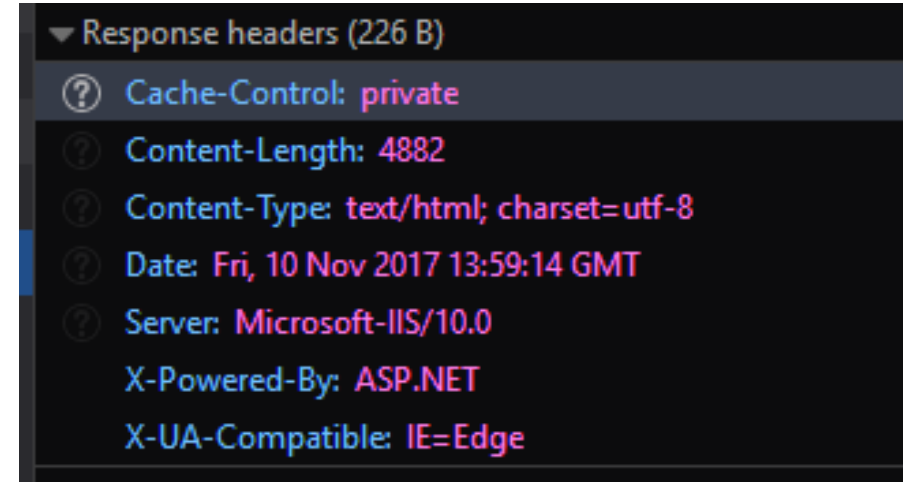
```
Cache-Control:public
```

```
Expires: Mon, 25 Jun 2012 21:31:12 GMT
```

- While **Cache-Control** and **Expires** inform the browser when to get the resource from the network again, a few more headers tell the browser how to get it. Conditional requests are requests that are made in this manner.

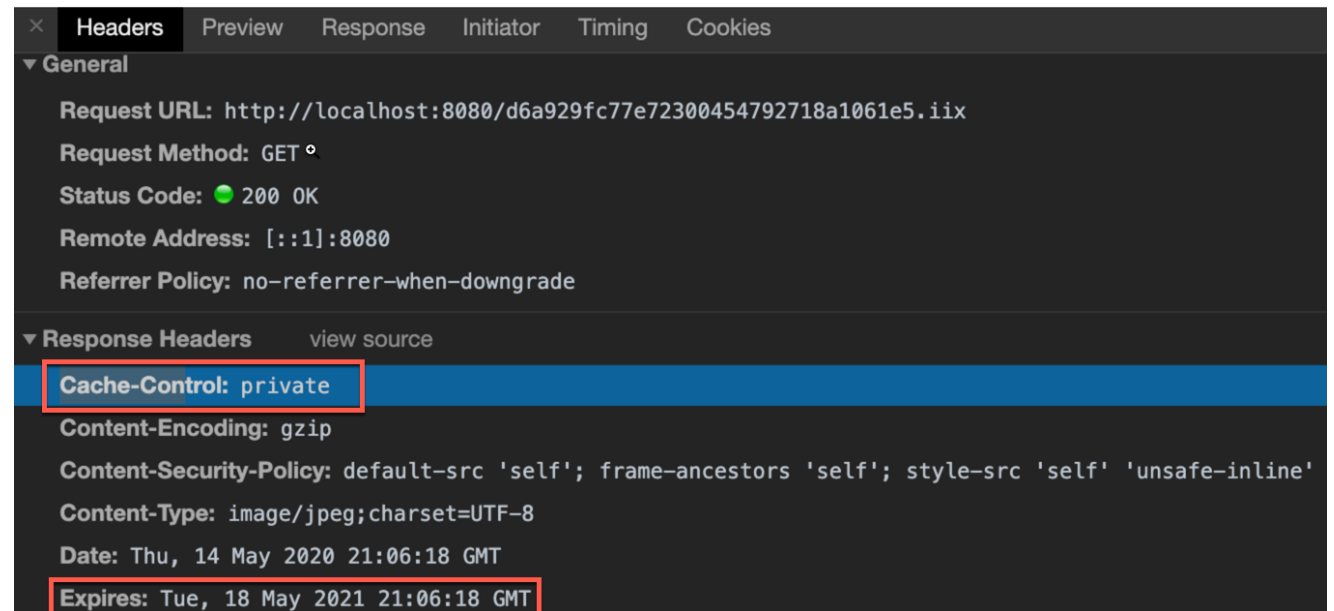
Why is it important..

- The purpose of properly implemented caching headers is to avoid proxies storing individualized information.
- If the response may be cached, the server must include the proper headers.
- When it comes to security, this is possibly the most crucial heading.
- This header can be customized in several ways.
- The page can be either "private" or "public," which is the most significant feature.



Which Vulnerabilities can it prevent..

- In some situations, this could break digital signatures. "no-transform" will prevent this from happening.
- The Expires header is used less frequently in modern browsers. It's best to stick to a routine.
- To avoid caching, set an expiration time in the past or use the value "0."



Advantages & Disadvantages..

Advantages	Disadvantages
<ol style="list-style-type: none">1. cache is not as efficient as HTTP2. increase data overhead and power consumption	<ol style="list-style-type: none">1. security measures will be affected

How HTTP Security Headers Can Improve Web Application Security..

- We usually mean detecting exploitable flaws and addressing them in application code when we talk about web application security, especially on this blog.
- HTTP security headers add an extra layer of protection by limiting the behaviors that the browser and server are allowed to perform while the web application is starting.
- HTTP headers, like other web technologies, come and die based on browser vendor support.
- Headers that were widely supported a few years ago, especially in the subject of security, may already be deprecated.
- Simultaneously, wholly new suggestions can obtain widespread acceptance in a few of months.
- It's difficult to keep up with all of these developments.



Thank you.....