Leonardo Nodari

Riccardo Bonafede


Leonardo
Nodari
Riccardo
Bonafede

Web Security

- Pwning but without registers, memory and bit endianess
  - More bonaff friendly
- Divided into two Macro Areas
  - Server Side ← You pwn the server
  - Client Side ← You pwn the user.

Web Security

Browsers interact with a lot of resources at the same time. You have multiple tabs in potentially multiple windows, with multiple frames in it. And everything executes code by others.

HTTP is a way to distribute untrusted code to people's computers where they will put their credit card number and all sort of personal information.

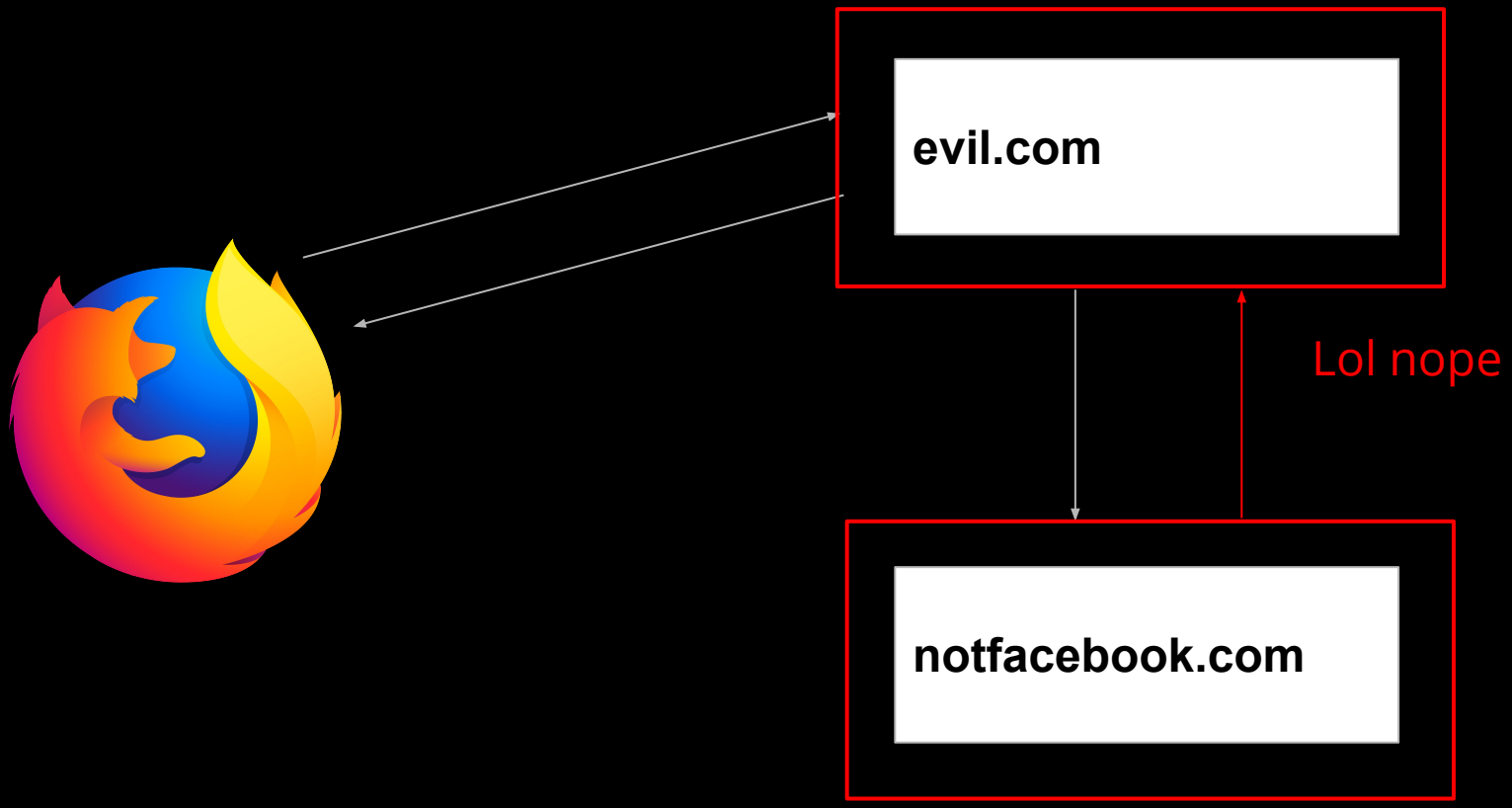
Same Origin Policy

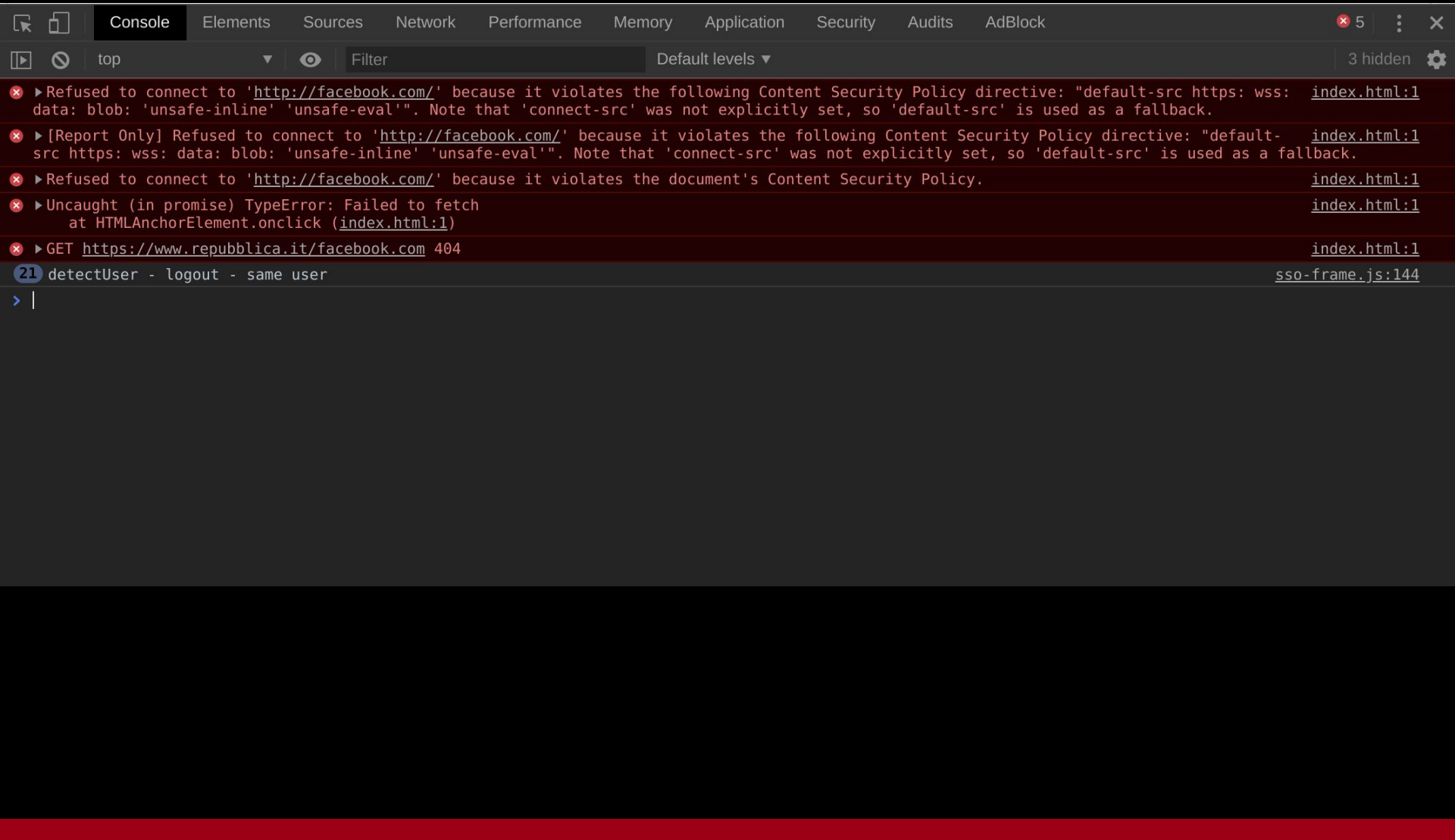
- What keeps fishywebsite.org from reading your messaging history on facebook?

Same Origin Policy!

A web browser permits scripts contained in a first web page to access data in a second web page, but only if both web pages have the **same origin**. -wiki

Same Origin Policy




Console
Elements
Sources
Network
Performance
Memory
Application
Security
Audits
AdBlock
5
top
Filter
Default levels
3 hidden
Refused to connect to 'http://facebook.com/' because it violates the following Content Security Policy directive: "default-src https: wss: data: blob: 'unsafe-inline' 'unsafe-eval'". Note that 'connect-src' was not explicitly set, so 'default-src' is used as a fallback.
index.html:1
[Report Only] Refused to connect to 'http://facebook.com/' because it violates the following Content Security Policy directive: "default-src https: wss: data: blob: 'unsafe-inline' 'unsafe-eval'". Note that 'connect-src' was not explicitly set, so 'default-src' is used as a fallback.
index.html:1
Refused to connect to 'http://facebook.com/' because it violates the document's Content Security Policy.
index.html:1
Uncaught (in promise) TypeError: Failed to fetch
at HTMLAnchorElement.onclick (index.html:1)
index.html:1
GET https://www.repubblica.it/facebook.com 404
index.html:1
21
detectUser - logout - same user
sso-frame.js:144

Origin

- The origin is composed by the scheme/host/port tuple

```
https://sub.foo.bar:8443
```

- Note that the host comprends ALL the domain, so for example

```
sub.foo.bar != foo.bar
```

Why do we want to break this SOP?

To steal secrets!

- Csrf Tokens
- Api Tokens
- Session Tokens
- Mo4r Tokens!

web guys love tokens

So, let’s steal some juicy token!


I CAN SEE
XSS
imgflip.com

Cross Site Scripting - XSS

- The first, and the most easy way to not throw a xorigin exception it’s just to not be xorigin
- What if we can execute javascript code inside another origin?

Cross Site Scripting - XSS

How does an XSS happen?

- Reflection! When you print user supplied input on your page.

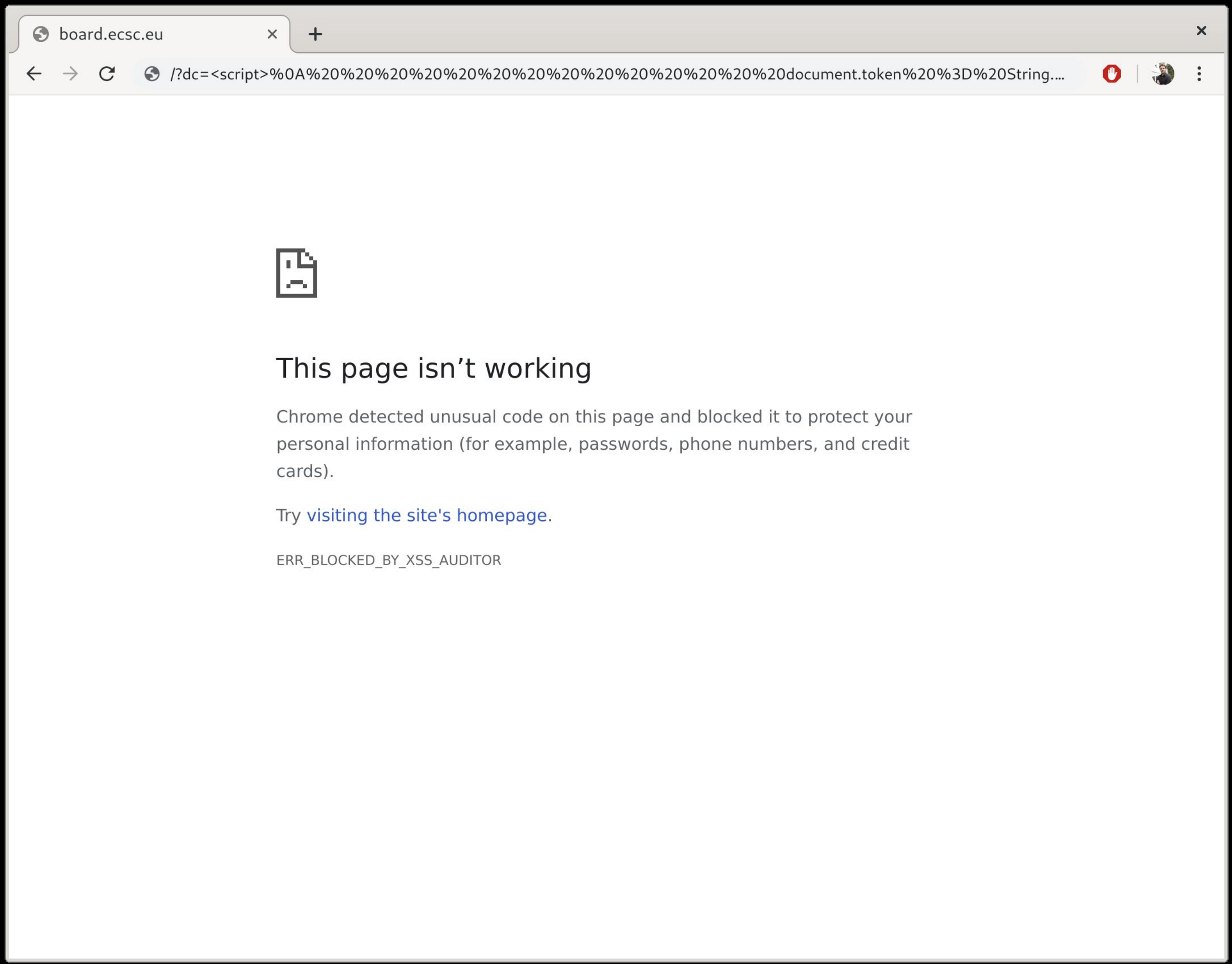
`http://foo.bar/?search=<tag>`

```
<strong>Cannot find <tag></strong>
```

Xss Auditors

- In 2010 xss were wild
- Pwns were spreading because of reflection
- How did devs managed to mitigate this? templates? properly sanitizing untrusted user input? nah
- Lets the browser guess if there is reflection or not!
- Welcome to the now dying XSS auditor!

If You Ever Saw This



Then You Were Near to Be Pwned!

Thanks Mr. XSS Auditor!

/s

Seriously, How Does This Work

- Since some months ago, active by default on chrome and other browsers (safari, opera, edge..)
- The devs can decide whether or in which mode activate it, by setting the **X-XSS-Protection** header
  - X-XSS-Protection: 0 ← disabled
  - X-XSS-Protection: 1 ← filtering
  - X-XSS-Protection: 1; mode=block ← blocking

Seriously, How Does This Work

```
http://foo.bar/?x=<script>alert(1);</script>
```

```
<html>
[...]
<script>alert(1);</script>
[...]

<!-- yay, I saved another user! -->
```

And What About False Positives?

```
http://foo.bar/?x=<script>token=”superse
```

```
<html>
[...]
<script>token=”supersecretrandomtoken”</script>
[...]
```

```
<!-- yay, I saved another user ! -->
```

Getting an Oracle

- The auditor wants to avoid false positives, so it looks for *at least* N characters reflected in the response
- We can use it to bruteforce secrets inside a page in linear time

`http://foo.bar/?x=<script>token=”a` ← nope

`http://foo.bar/?x=<script>token=”s` ← triggered!

And now?

- The page just crashed because you guess the first character of the *supersecrettoken*
- How do you find this out? There is sop, so no way to know that there is no page

Welcome to some neat browser “”features””

Iframes

- For whatever reason, **frame.contentWindow.length** is not subject to SOP restrictions.
- Guess what, the crashed page has no iframes...
- Everything we need is a reference to the `window` object of the xorigin page and an iframe in the page from which we want to leak secrets from
- For example by using iframes, or `window.open()`

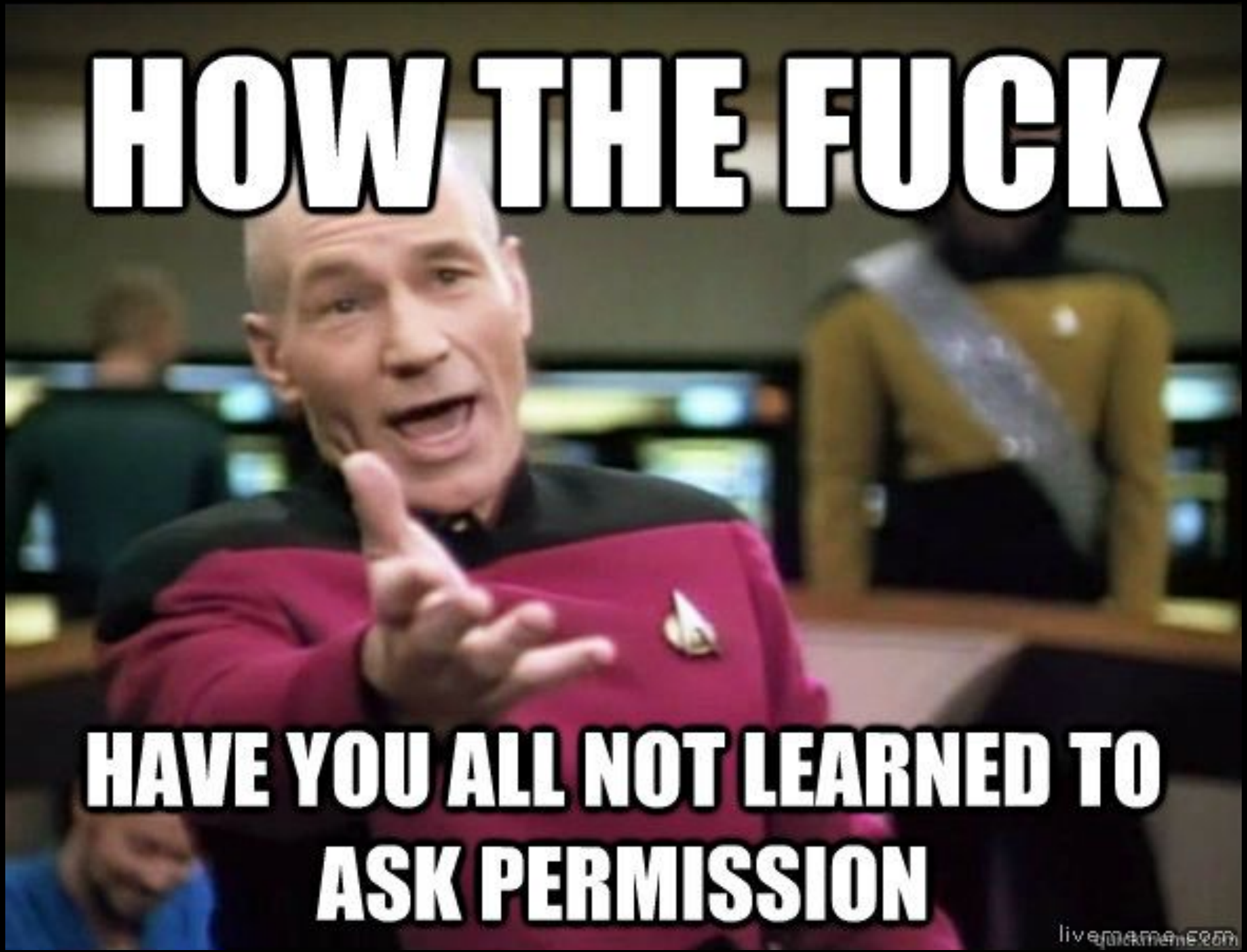
So now let's brute

```
var url = 'http://otherdomain.com//test.php?x=';
var iframe = document.createElement('iframe');
[...]
iframe.location.replace(url+guess); // try to load the iframe
iframe.onload=function(){
    if(!this.contentWindow.length){
        //found a char!
        [..]
    }else{
        //try with a new guess
        [..]
    }
}
```


SIDE
CHANNELS
imgflip.com

What's next?

We can now leak some secrets if the page is made in a certain way and our unlucky friend does not use firefox, but what ifanothersite.com actually needs to be access by anothersite.com?


HOW THE FUCK
HAVE YOU ALL NOT LEARNED TO ASK PERMISSION
livememe.com

Let Me Introduce You To My Friend CORS

Cross-Origin Resource Sharing is a way to “relax” the Same-Origin Policy. You know, sometimes sites have the necessity of retrieving information from an xorigin api.

Cross-Origin Resource Sharing

In order to do a Cross Origin request, the resource needs to do a **Preflight** request, in which the origin requests the permission to read and/or sending credentials/special headers.

```
fetch('http://another.com', {
   credentials: "include"
});
```

Cross-Origin Resource Sharing

```
OPTIONS / HTTP/1.1
Host: example.com
Origin: http://foo.bar
[..]
```

```
HTTP/1.1 200 OK
Access-Control-Allow-Origin: http://foo.bar
Access-Control-Allow-Credentials: true
[..]
```

Trusting Multiple Origins

What if I my fancy api is used by multiple sites?

- Access-Control-Allow-Origin: http://ghesbo.ro / http://staccastacca.com ← nope,
- Access-Control-Allow-Origin: *.ghesbo.ro ← nope
- Access-Control-Allow-Origin: * ← ok?
  - Access-Control-Allow-Credentials: true ← but we can’t use this

Trusting Multiple Origins

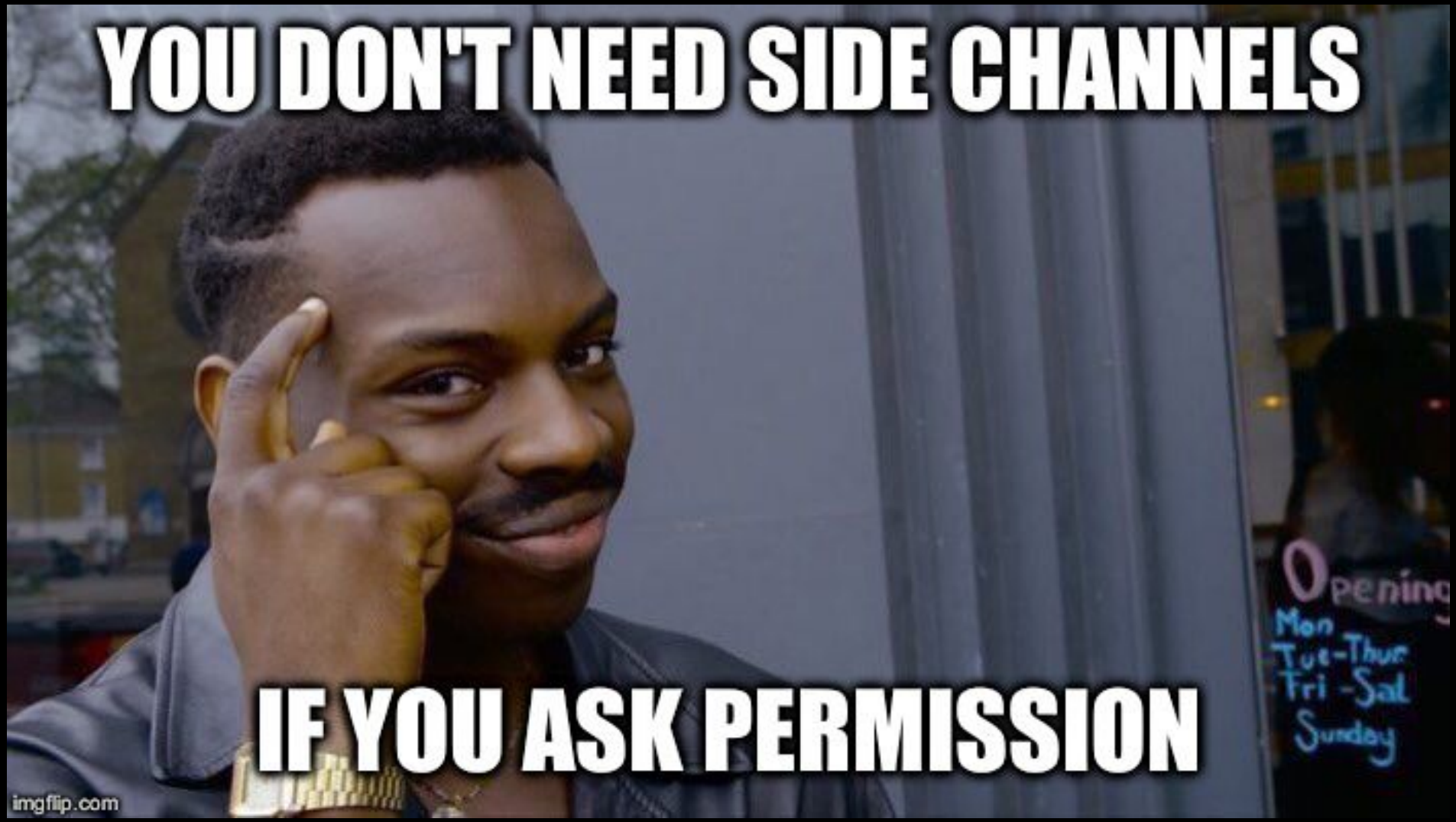
You can use a regex in Apache

```
SetEnvIf Origin "http(s)?://(.+\.)?domain\.com(:\d{1,5})?$" CRS=$0

Header always set Access-Control-Allow-Origin "%{CRS}e" env=CRS
```

Or some PHP

```
header('Access-Control-Allow-Origin:' + SERVER['HTTP_ORIGIN']);
header('Access-Control-Allow-Credentials: true');
```


YOU DON'T NEED SIDE CHANNELS
IF YOU ASK PERMISSION
Mon
Tue-Thur
Fri -Sat
Sunday
imgflip.com

Yup, devs actually do this

```
<?php

 header('Access-Control-Allow-Origin:' + SERVER['HTTP_ORIGIN']);
 header('Access-Control-Allow-Credentials: true');


 /* yay, now everyone can read from our site */
```

How do we exploit this?

From any domain, just serve this snippet

```
resp = fetch('http://goodsite.com', {
  credentials: "include"
});
alert(resp); // this time no exception!
```

Devs are getting smarter……

```
<?php
 if(strpos($_SERVER['HTTP_ORING'], 'goodsite.com')!==FALSE){
    //goodsite.com it’s in the origin. Seems legit
    header([...])
}
```

Not really

you know,

```
strpos(‘goodsite.com.evilsite.com’,’goodsite.com’) !== FALSE);
```

...

What about null?

Sometimes you will see this funny header:

```
Access-Control-Allow-Origin: null
```

This happens for a lot of reasons. Normally just misconfigurations of the reverse proxy, that simply doesn't know what it is doing with its life.

But is this exploitable?

null has no protocol, no host/no port. Seems to be an invalid origin right?

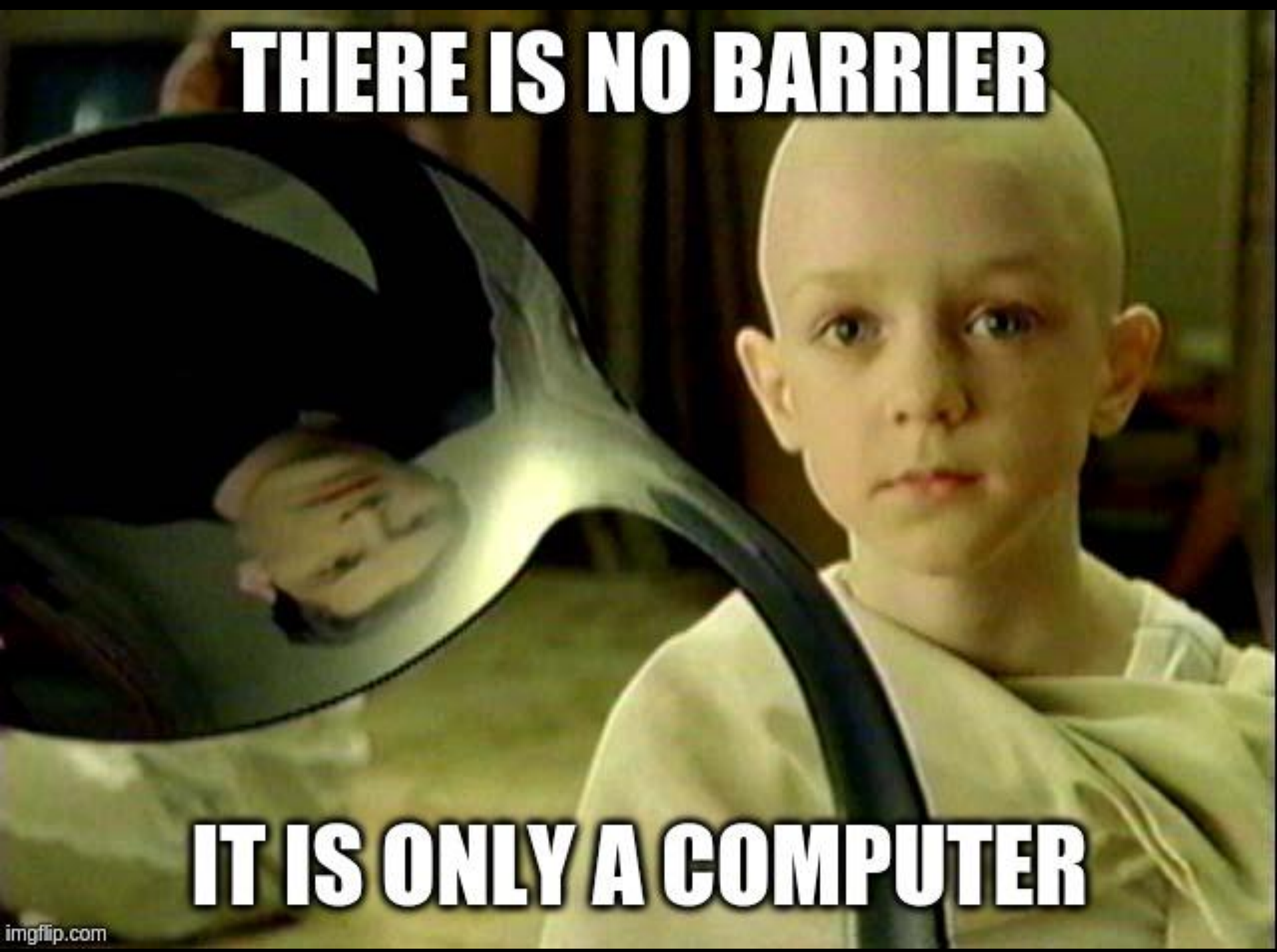
Guess the Origin

```
<iframe
src="data:text/html,<script>console.log(location.origin);</script>
">
</iframe>
```

What If There Is No SOP?

Do not try to leak the content of *index.php*, that's impossible. Instead, only *try* to realize the truth... index.php is a .css file. Then you will see that it will be cached.

-- every CDN to every webguy


THERE IS NO BARRIER
IT IS ONLY A COMPUTER
imgflip.com

Web Cache Deception

- A nice not so new, not so client side attack
  - BlackHat 2017
- It is actually a SOP bypass. Or maybe not, but I find it cool so whatever

Server Side Caching 101

1. Client issue a request
2. The CDN look if has a cached version of the response. If yes return it, if not ...
3. The backend generates a response
4. The CDN/reverse proxy/whatever decides if the response needs to be cached
5. The response is returned to the user

Server Side Caching 101

When does a resource need to be cached?

It’s generally configured by the sysop:

- It is static (a .css file, a .js file etc)
- It is under a certain directory that contains only static files

Rule of thumb: If it is a .css file 9/10 it is cached

Web Cache Deception

What happens if we make a CDN think that a dynamic page is static/needs to be cached?

It will serve it to everyone!

We can actually poison the cache to happily give every user payloads (this improves *a lot* the impact of a xss)

But..

What if we make an user to poison the cache?

Web Cache Deception

- Our “victim” is logged into goodsite.com
- The attacker make him click on goodsite.com/*magic*
- goodsite.com/*magic* it’s now cached, with every secret of victim
- Attacker can now simply fetch goodsite.com/*magic* from his computer, and he will see secrets of victim
- Ez

Web Cache Deception

How do we make the CDN cache a dynamic file?

In a PHP stack, a lot of time, you just need to append another filename

`https://foo.bar/index.php/randomstring.css`

`index.php` will now be cached, and to retrieve it's content just get the full url(with the token obv)

Web Cache Deception

Other times you need to de-routing the backend, and make the url interpreted in two different way by the CDN and the backend.

`https://foo.bar/index.php/randomstring.css`

- http://foo.bar/index.php ← backend
- http://foo.bar/index.php/randomstring.css ← CDN

Web Cache Deception

Tomcat is so enterprise.

`take /foo;lol/bar;lol == /foo/bar`

- `/index;.css == /index` for the backend
- `/index;.css == /index;.css` For the CDN

And now think about

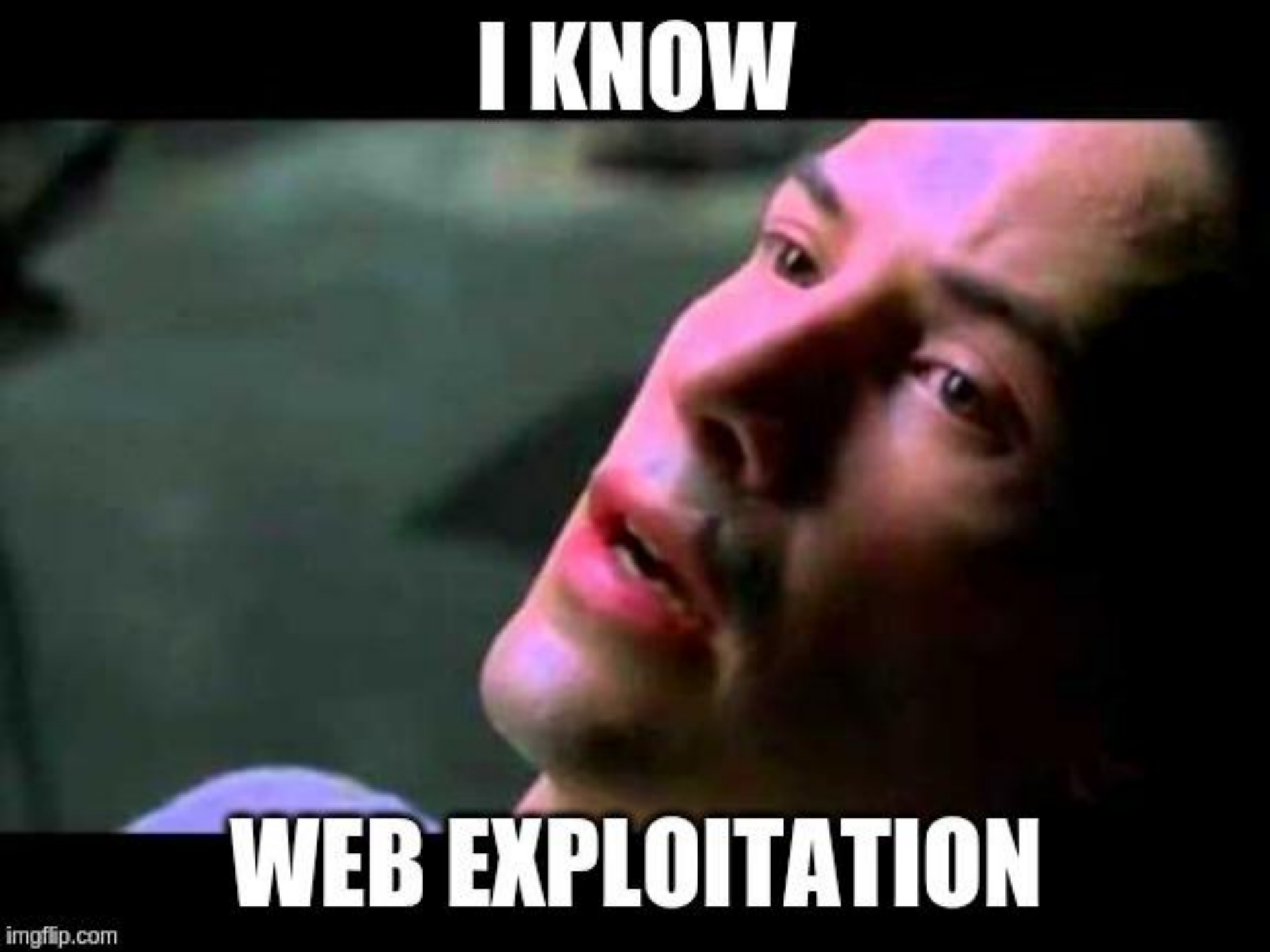
- `/assets/..;/index;.css`

`..;` it's a valid dirname!, but for tomcat, it's actually the parent dir

Other tricks

Other useful tricks are:

- Url encoding: `/../ == %2f%2e%2`
  - Make it double: `%252f%252e%252e%252f`
- Just try windows things:
  - `/../ == \..\`
- Unicode (pretty uncommon):
  - \u012f


I KNOW
WEB EXPLOITATION
imgflip.com

Contacts

https://spritze.rs/

@thenodi95
(me@leonardonodari.it)

@n0pwnintended

@bonaff3
(bonaff@live.it)