NUMTONCE

NUMBER **U**SED **M**ORE **T**HAN **ONCE**

CHRISTOPH HACKENBERGER & PAUL KALAUNER

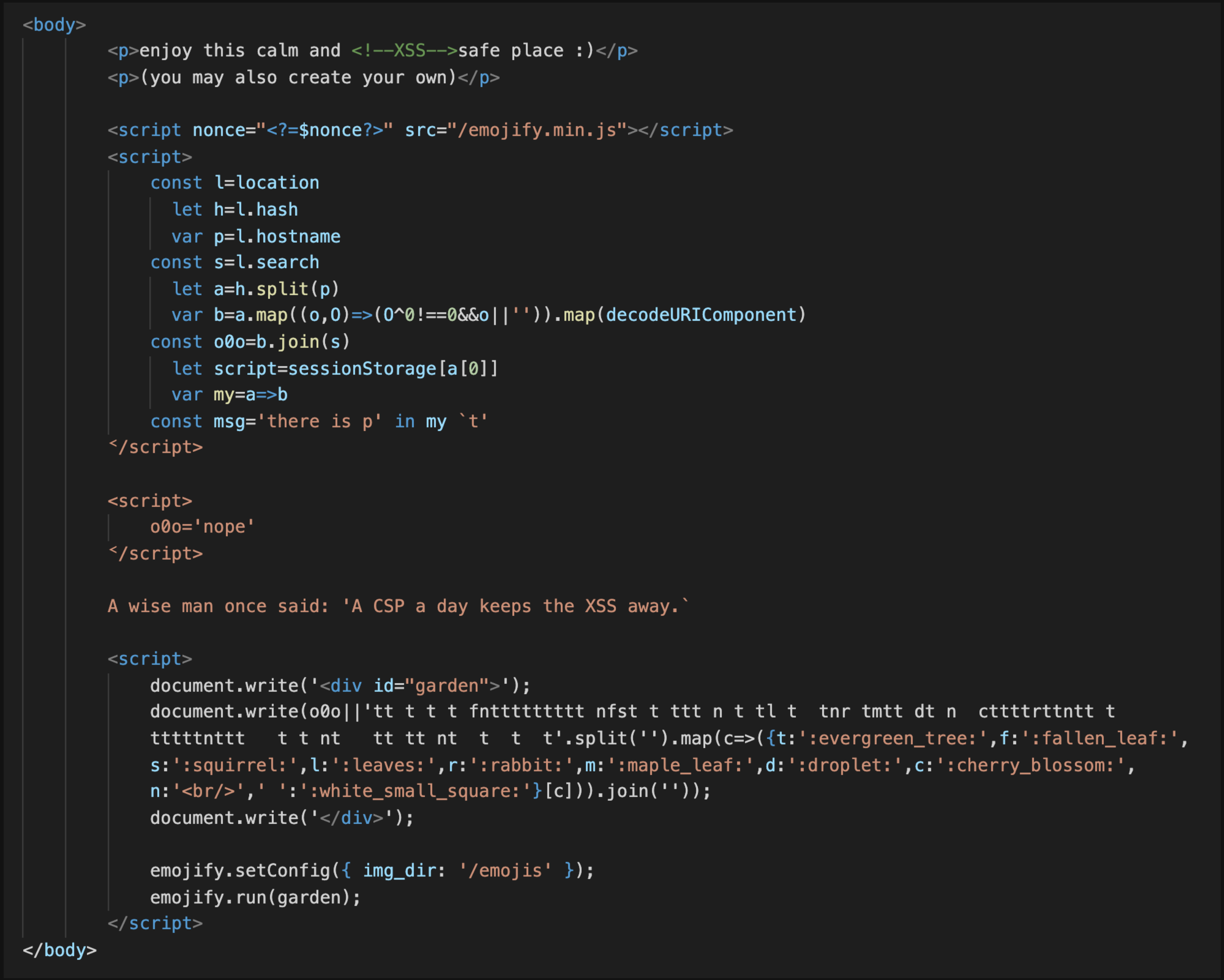
Hack.lu 2019

CHALLENGE DESCRIPTION

With all the bad news in the world, everyone needs a calm place to wind down. We built one but you have to help us keep it safe. If you find anything suspicious, tell the forest ranger! He might reward you with a cookie :)

OVERVIEW OF THE APPLICATION

- Application to view and create “grids“ of emojis
- Create your own grid by modifying the URL



```
<body>
        <p>enjoy this calm and <!--XSS-->safe place :)</p>
        <p>(you may also create your own)</p>

        <script nonce="<?=$nonce?>" src="/emojify.min.js"></script>
        <script>
            const l=location
              let h=l.hash
              var p=l.hostname
            const s=l.search
              let a=h.split(p)
              var b=a.map((o,O)=>(O^0!==0&&o||'')).map(decodeURIComponent)
            const o0o=b.join(s)
              let script=sessionStorage[a[0]]
              var my=a=>b
            const msg='there is p' in my `t'
        </script>

        <script>
            o0o='nope'
        </script>

        A wise man once said: 'A CSP a day keeps the XSS away.`

        <script>
            document.write('<div id="garden">');
            document.write(o0o||'tt t t t fnttttttttt nfst t ttt n t tl t  tnr tmtt dt n  cttttrttntt t
            tttttnttt   t t nt   tt tt nt  t  t  t'.split('').map(c=>({t:':evergreen_tree:',f:':fallen_leaf:',
            s:':squirrel:',l:':leaves:',r:':rabbit:',m:':maple_leaf:',d:':droplet:',c:':cherry_blossom:',
            n:'<br/>',' ':':white_small_square:'}[c])).join(''));
            document.write('</div>');

            emojify.setConfig({ img_dir: '/emojis' });
            emojify.run(garden);
        </script>
</body>
```

ATTEMPTS

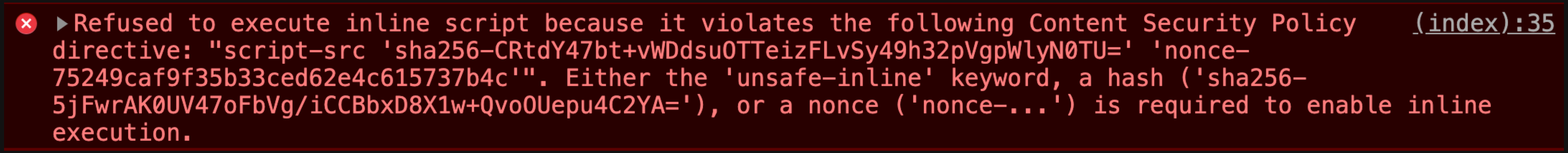
- Trying to get around the CSP and use XSS ;)
- Trying to find a vulnerability in the emojify.js library
- Trying to use `X-XSS-Protection: 1; mode=block` as a side-channel

XSS ATTEMPTS

http://hostname/#hostname<script>alert(1);</script>

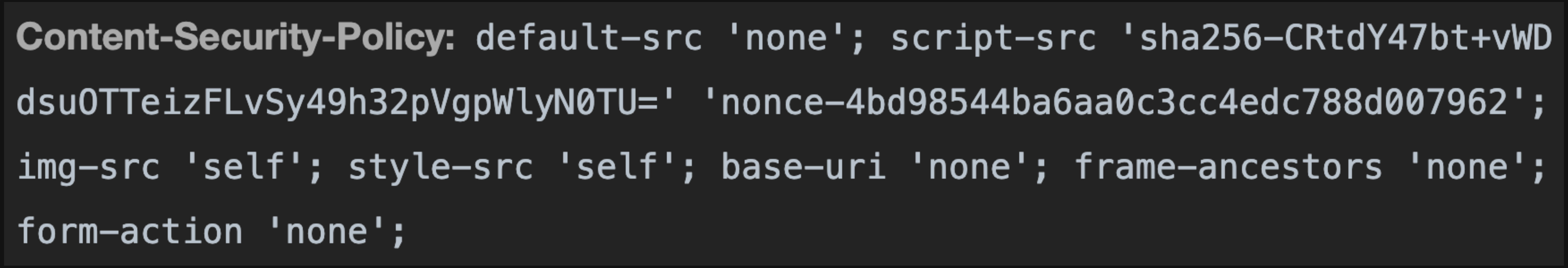
http://hostname/#hostname<img src=“x“onerror=“javascript:alert(1)“/>

http://hostname/#hostname<iframe src=“javascript:alert(1)“</iframe>

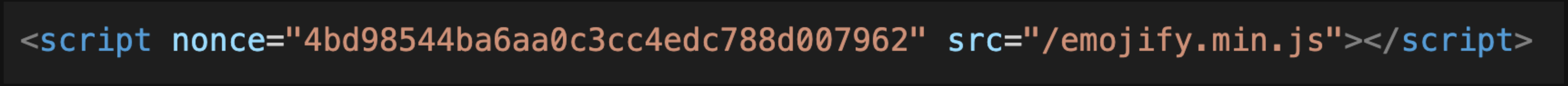


```
Refused to execute inline script because it violates the following Content Security Policy    (index):35
directive: "script-src 'sha256-CRtdY47bt+vWDdsuOTTeizFLvSy49h32pVgpWlyN0TU=' 'nonce-
75249caf9f35b33ced62e4c615737b4c'". Either the 'unsafe-inline' keyword, a hash ('sha256-
5jFwrAK0UV47oFbVg/iCCBbxD8X1w+QvoOUepu4C2YA='), or a nonce ('nonce-...') is required to enable inline
execution.
```

CSP-HEADERS



```
Content-Security-Policy: default-src 'none'; script-src 'sha256-CRtdY47bt+vWD
dsuOTTeizFLvSy49h32pVgpWlyN0TU=' 'nonce-4bd98544ba6aa0c3cc4edc788d007962';
img-src 'self'; style-src 'self'; base-uri 'none'; frame-ancestors 'none';
form-action 'none';
```



```html
<script nonce="4bd98544ba6aa0c3cc4edc788d007962" src="/emojify.min.js"></script>
```

SIDE-CHANNEL ATTEMPT[1]

- `X-XSS-Protection: 1; mode=block` set in headers
- Page is blocked if value of any GET parameter is found in the scripting part of the page source
- In theory, allows attacker to leak data from the page source using the behaviour of the page as a side-channel

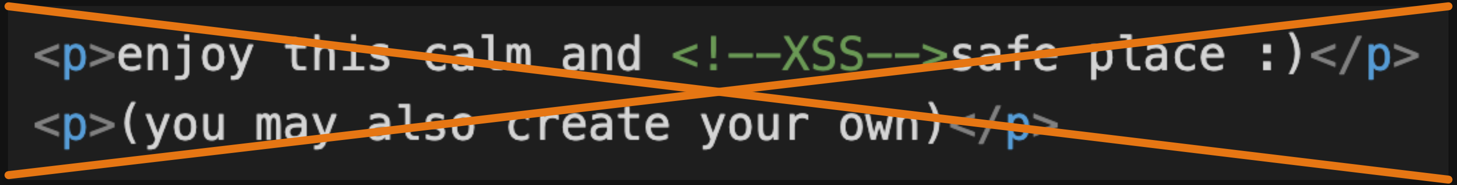
[1] https://stackoverflow.com/a/57802070

SIDE-CHANNEL EXAMPLE[1]

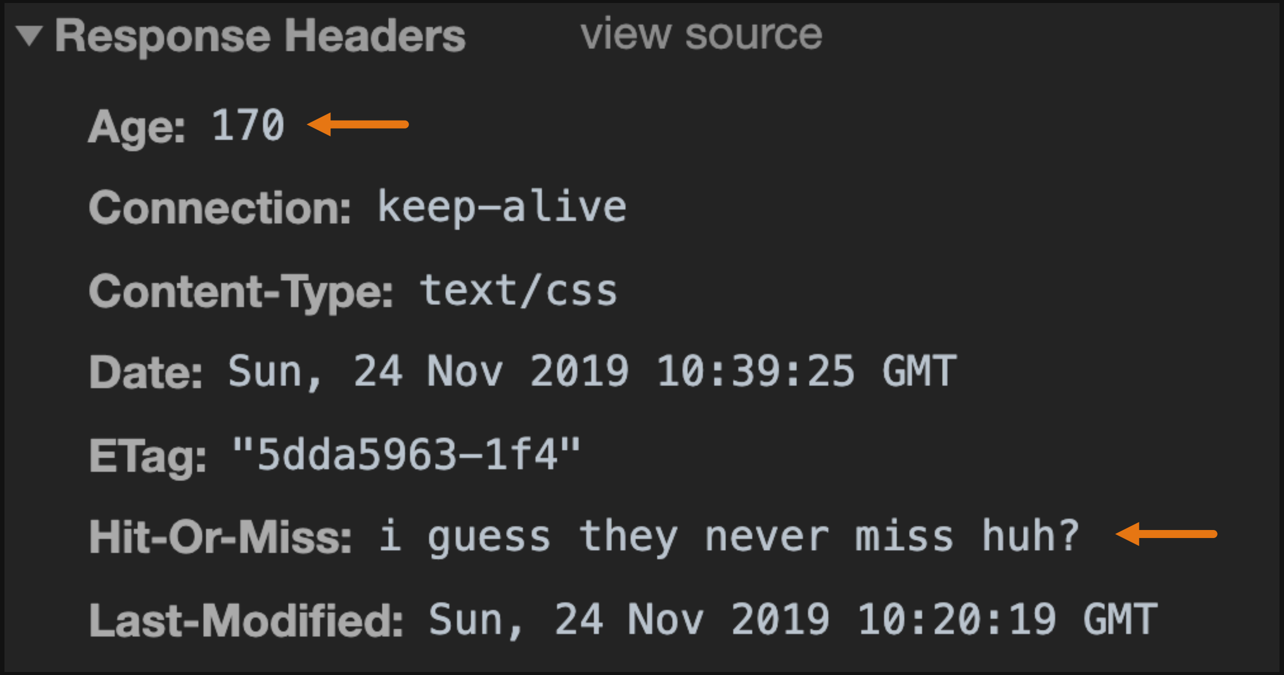
- Page contains Javascript code `var secret="2345"`
- Attacker uses URL with `?leak=var secret="1` → page is not blocked
- URL with `?leak=var secret="2` → page is blocked
- Extract information character by character

[1] https://stackoverflow.com/a/57802070

HINTS?



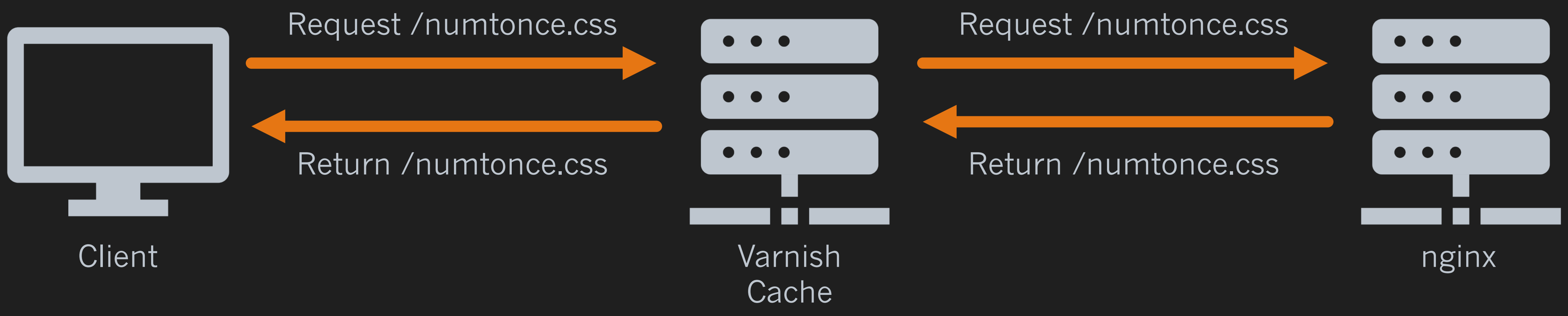
```
~~<p>enjoy this calm and <!--XSS-->safe place :)</p>~~
~~<p>(you may also create your own)</p>~~
```



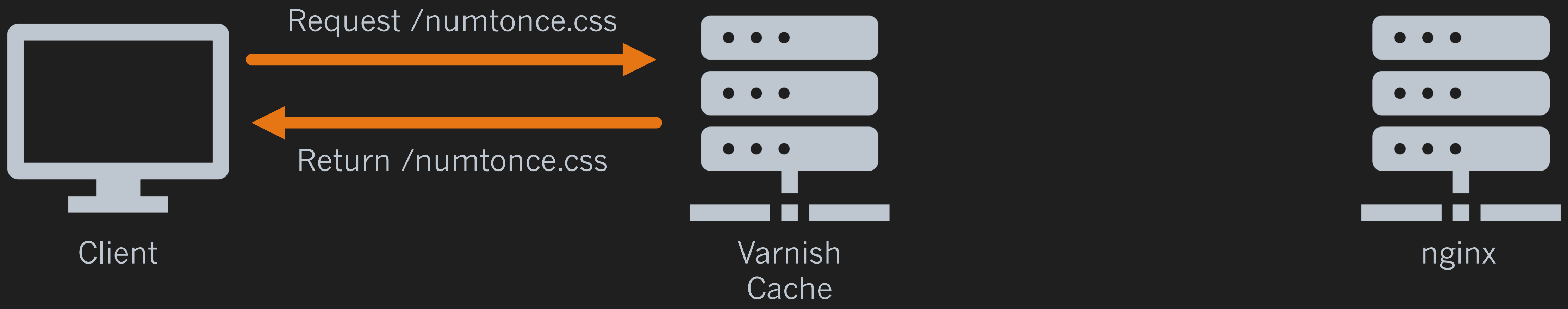
▼ **Response Headers** view source

```
Age: 170
Connection: keep-alive
Content-Type: text/css
Date: Sun, 24 Nov 2019 10:39:25 GMT
ETag: "5dda5963-1f4"
Hit-Or-Miss: i guess they never miss huh?
Last-Modified: Sun, 24 Nov 2019 10:20:19 GMT
```

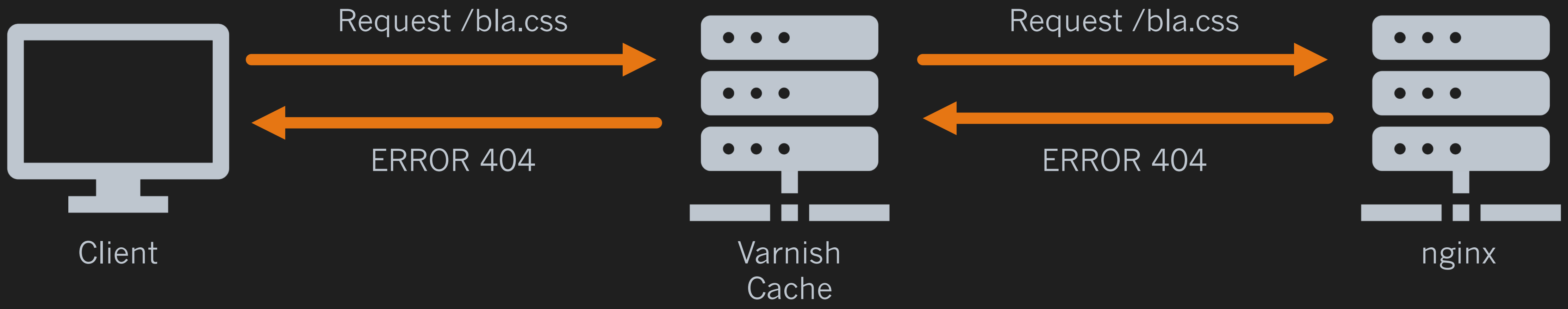
EXPLOITATION



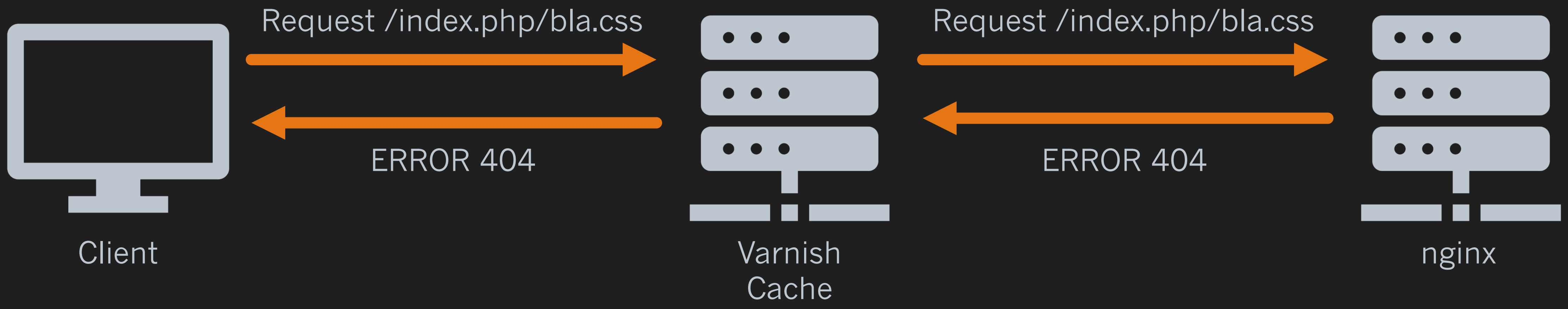
EXPLOITATION



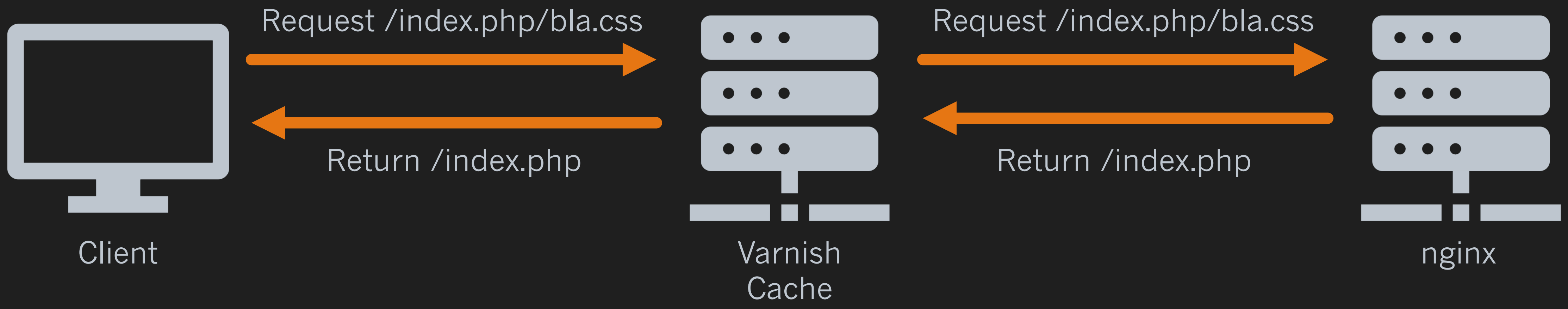
EXPLOITATION



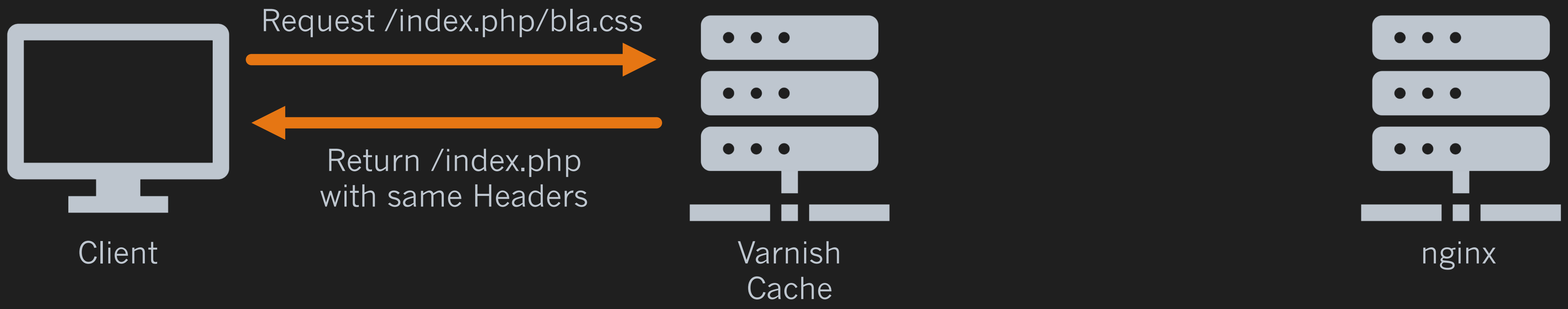
EXPLOITATION



EXPLOITATION



EXPLOITATION


Request /index.php/bla.css
Return /index.php with same Headers
Client
Varnish Cache
nginx

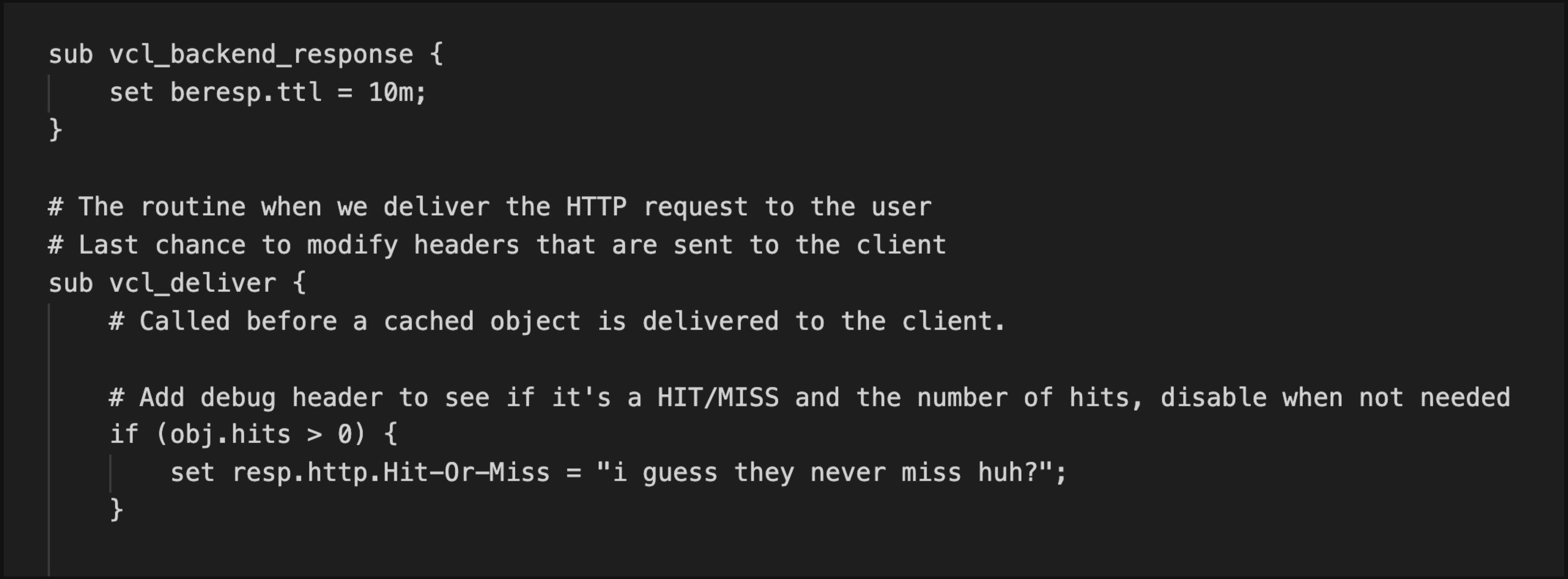
DEMO

SUMMARY OF EXPLOIT

- Abuse caching in combination with wrong configuration of nginx
- Reuse nonce for own script
- Send cookie to attacker controlled site

VULNERABILITY & COUNTER MEASURES

VARNISH CONFIGURATION

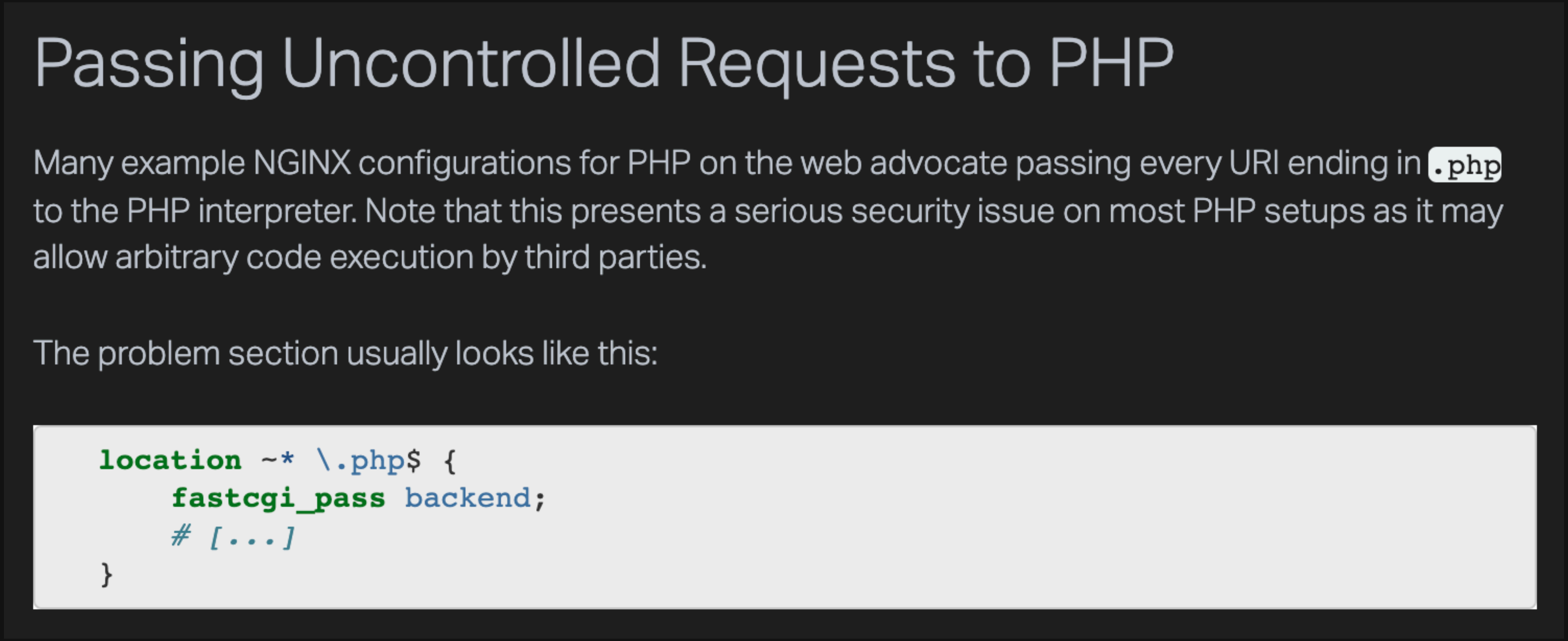


```
sub vcl_backend_response {
    set beresp.ttl = 10m;
}

# The routine when we deliver the HTTP request to the user
# Last chance to modify headers that are sent to the client
sub vcl_deliver {
    # Called before a cached object is delivered to the client.

    # Add debug header to see if it's a HIT/MISS and the number of hits, disable when not needed
    if (obj.hits > 0) {
        set resp.http.Hit-Or-Miss = "i guess they never miss huh?";
    }
```

FROM THE NGINX MANUAL



Passing Uncontrolled Requests to PHP

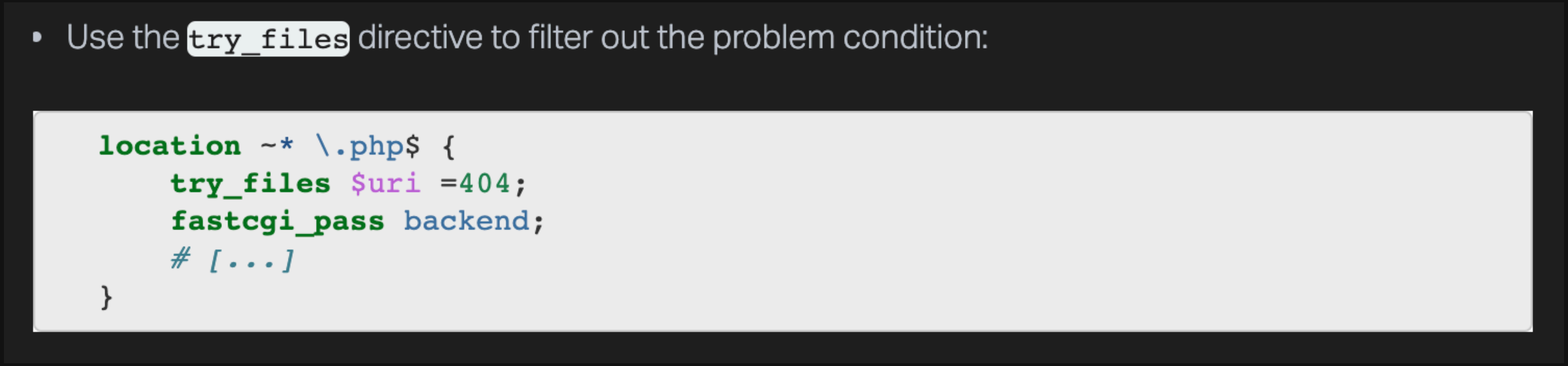
Many example NGINX configurations for PHP on the web advocate passing every URI ending in `.php` to the PHP interpreter. Note that this presents a serious security issue on most PHP setups as it may allow arbitrary code execution by third parties.

The problem section usually looks like this:

```
location ~* \.php$ {
    fastcgi_pass backend;
    # [...]
}
```

https://www.nginx.com/resources/wiki/start/topics/tutorials/config_pitfalls/

FROM THE NGINX MANUAL

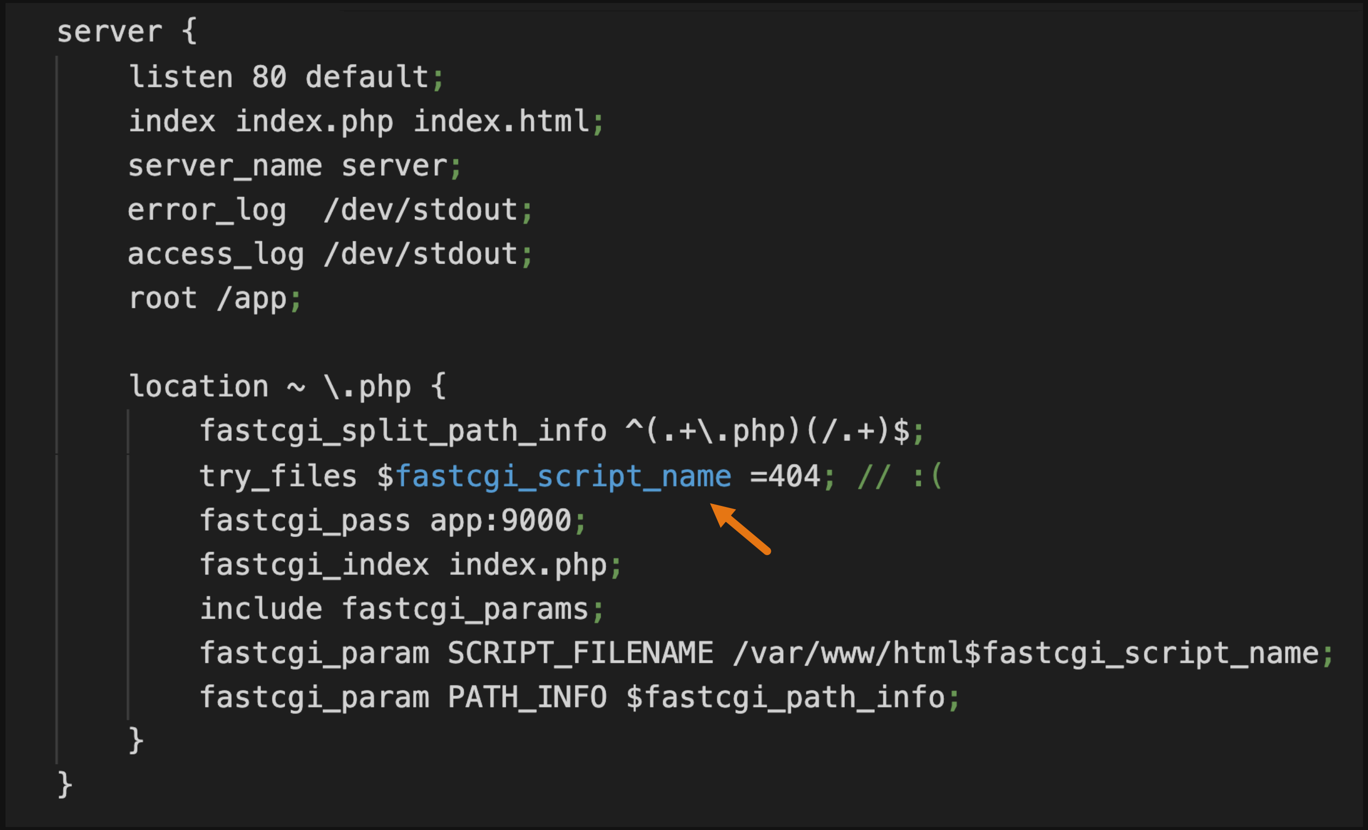


- Use the `try_files` directive to filter out the problem condition:

```
location ~* \.php$ {
    try_files $uri =404;
    fastcgi_pass backend;
    # [...]
}
```

https://www.nginx.com/resources/wiki/start/topics/tutorials/config_pitfalls/

NGINX CONFIGURATION



```
server {
    listen 80 default;
    index index.php index.html;
    server_name server;
    error_log  /dev/stdout;
    access_log /dev/stdout;
    root /app;

    location ~ \.php {
        fastcgi_split_path_info ^(.+\.php)(/.+)$;
        try_files $fastcgi_script_name =404; // :(
        fastcgi_pass app:9000;
        fastcgi_index index.php;
        include fastcgi_params;
        fastcgi_param SCRIPT_FILENAME /var/www/html$fastcgi_script_name;
        fastcgi_param PATH_INFO $fastcgi_path_info;
    }
}
```

UNSAFE JAVASCRIPT CODE

- Document.write() with unfiltered user input allows XSS
- Check user input (in this case the anchor part of the URL)

IMPACT IN REALISTIC SCENARIOS

- Bypassing CSP → possibility of XSS
- Use of a cache is very common (Facebook, Wikipedia, etc.)

QUESTIONS?