Web Security in 2022

luca@doyensec.com

HELLO!

I am Luca

♥ AppSec since 2004

Doyensec Co-founder

Former Lead of AppSec (LinkedIn),
Senior Security Researcher (Matasano), …

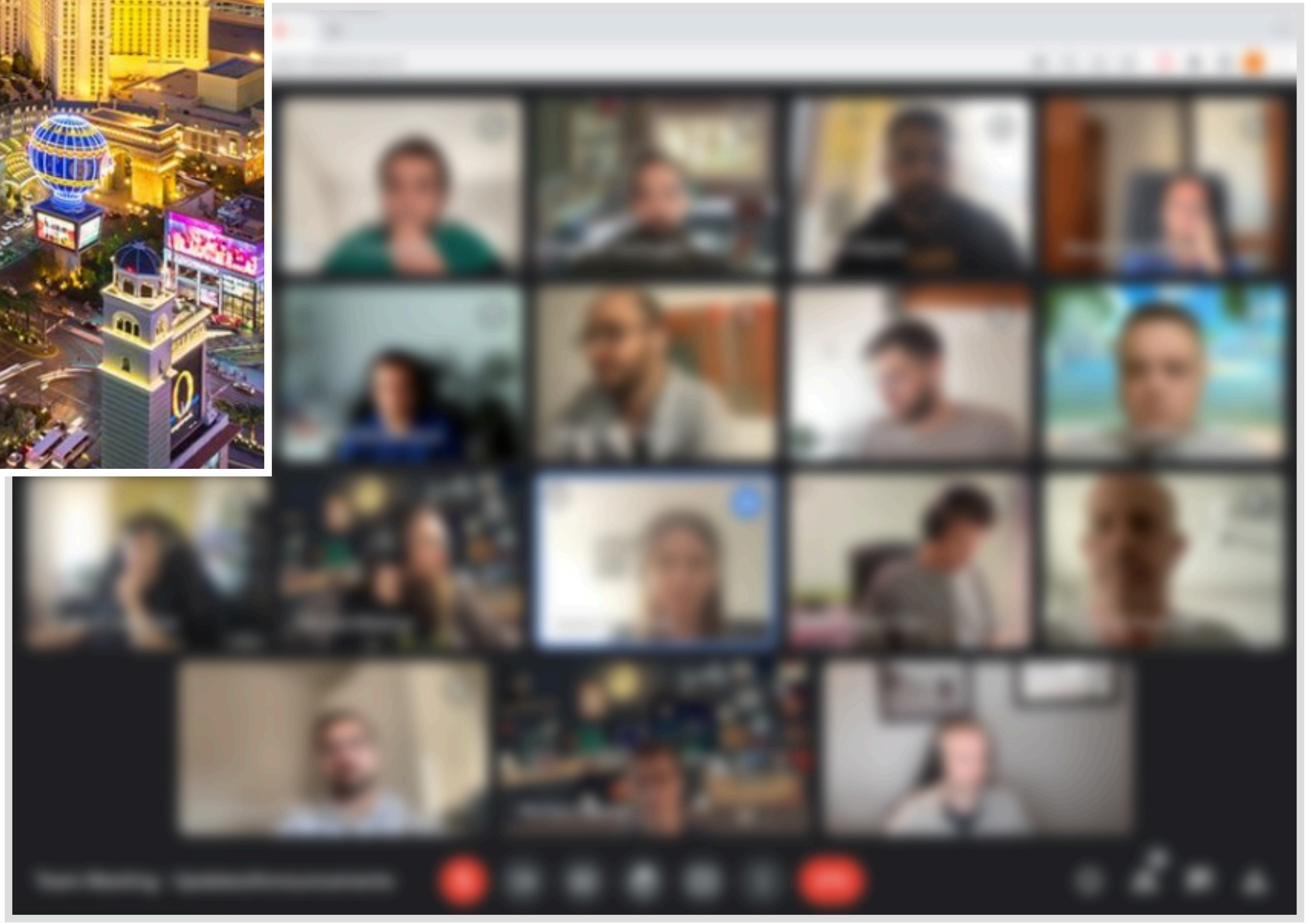
You can find me at
luca@doyensec.com
@lucacarettoni

“

We work at the intersection of software development and offensive engineering to help companies craft secure code.

doyensec.com/research

Best Bugs @Doyensec



INSTRUCTIONS FOR USE

Web Security Centric

Based on web tech, but not necessarily web app

Credit where credit's due

Not all bugs are mine. Thanks team!

Tech / FinTech Centric

We mainly work within these industries

Statistically non-significant

Not that the OWASP Top10 is…

Modern frameworks and languages only

I definitely spend too much time on Js/Ts

Omitting well understood new classes

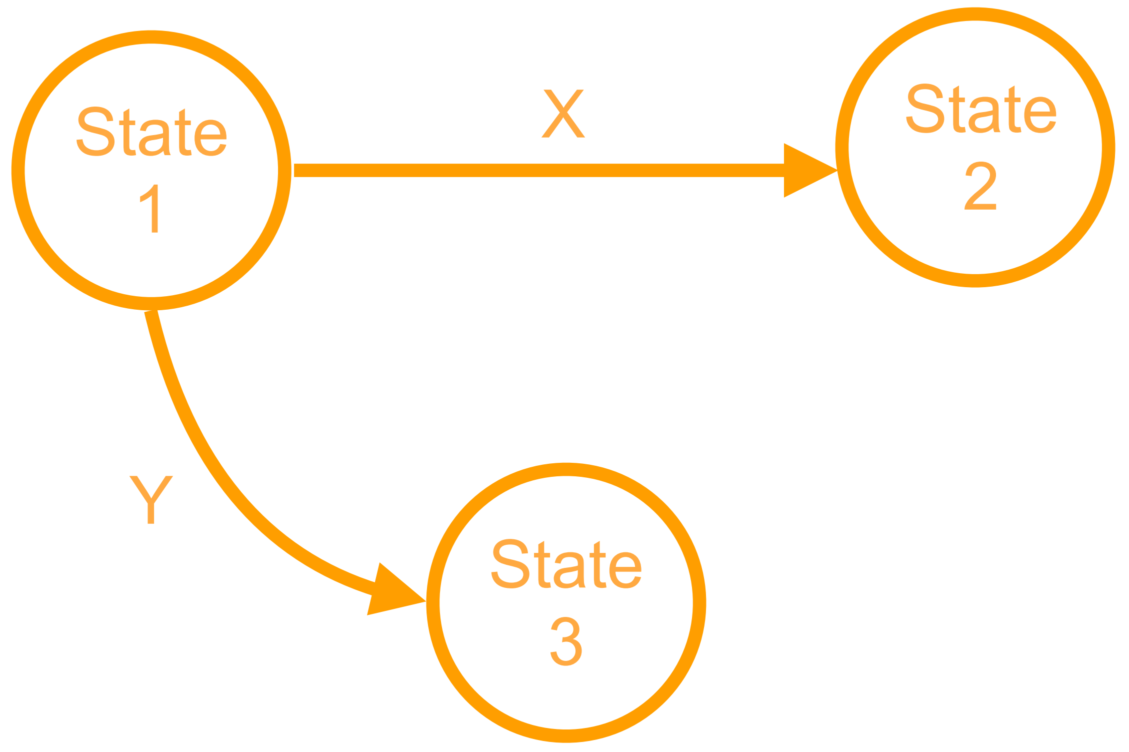
SSRF, HTTP request smuggling and other @albinowax tricks are removed for brevity. They're indeed new interesting attacks

1

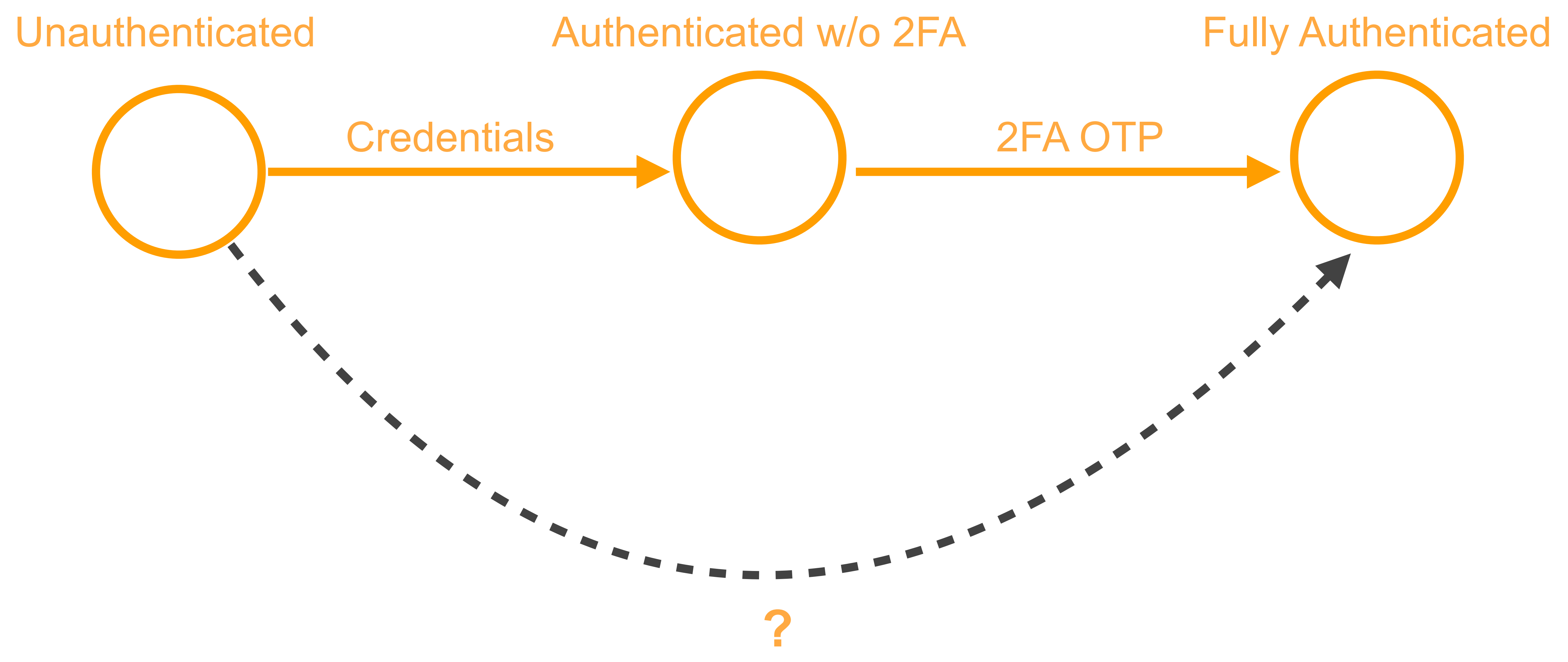
"A computer is a state machine. Threads are for people who can't program state machines"

Alan Cox

- A state machine is a mathematical abstraction used to design algorithms
- A state machine reads a set of inputs and changes to a different state based on those inputs
- They’re everywhere, including WebRTC and login flows
  - https://bugs.chromium.org/p/project-zero/issues/detail?id=1943



MY TINY STATE MACHINE BUG



LOGIN (Credentials)

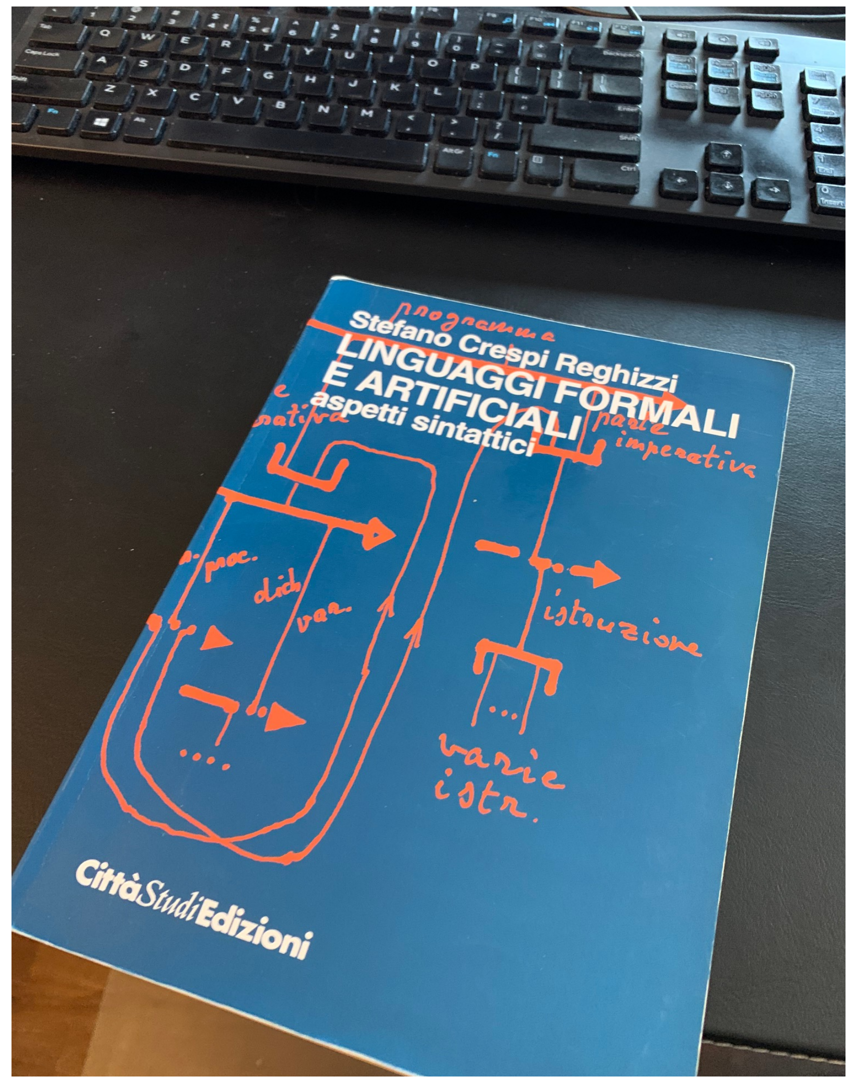
```
try {
  const account = await login(kClient, email, password, req.ipAddress);

  const result = {
    login: {
      accountId: account.id
    }
  };

  // if MFA is required, redirect to the two factor page
  if (account.two_factor_secret) {
    return res.render('login', {
      uid,
      details: prompt.details,
      params: {
        ...params,
        ...defaultParams,
        gaPageTracker: urls.INTERACTION_LOGIN,
```

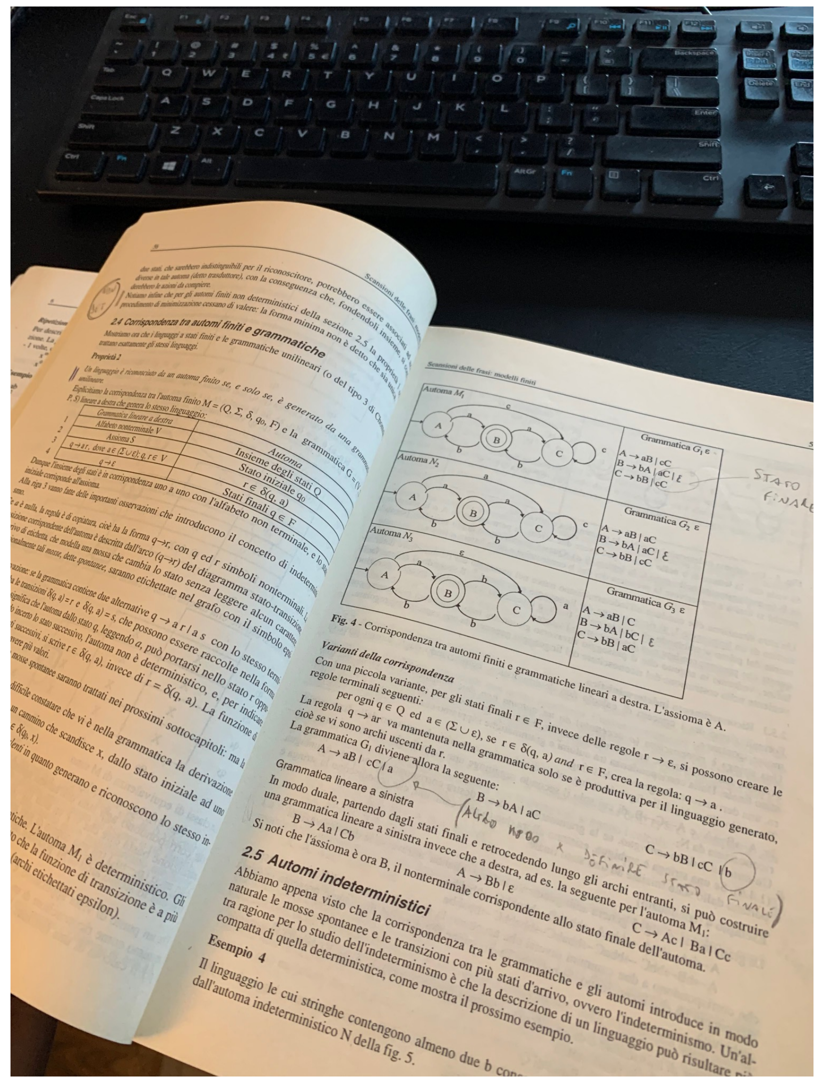
LOGIN (2FA OTP)

```
…
// verify two factor token if present in the POST request
      if (twoFactorToken) {
        // get user from db
        const account = await getUserByEmail(email);

        // verify two factor token
        const twoFactorService = new TwoFactorService();

        if (!twoFactorService.verify2faToken(account, twoFactorToken)) {
          // if invalid, return to login page to try again
         …
```

- No rate limiting
- Authentication bypass
  - Affects 2FA-enabled accounts only

- Who would have guessed?





2

"Given sufficient bug density, security design is irrelevant"

Ian Beer

Demo

…or backup video

CVE-2021-26437
VScode .ipynb XSS

- August 2021, Justin Steven releases https://github.com/justinsteven/advisories/blob/master/2021_vscode_ipynb_xss_arbitrary_file_read.md

```
{
  "cells": [
    {
      "cell_type": "code",
      "execution_count": null,
      "source": [],
      "outputs": [
        {
          "output_type": "display_data",
          "data": {"text/markdown": "<img src=x onerror=‘console.log(1)’>"}
        }
      ]
    }
  ]
}
```

99% of ElectronJS EXPLOITs

1

Take control of the DOM

Hijack the navigation flow,
Cross-Site Scripting,
Protocol Handlers,
AuxClick,
Man-in-The-Middle,
Drag & Drop

2

Bypass isolation

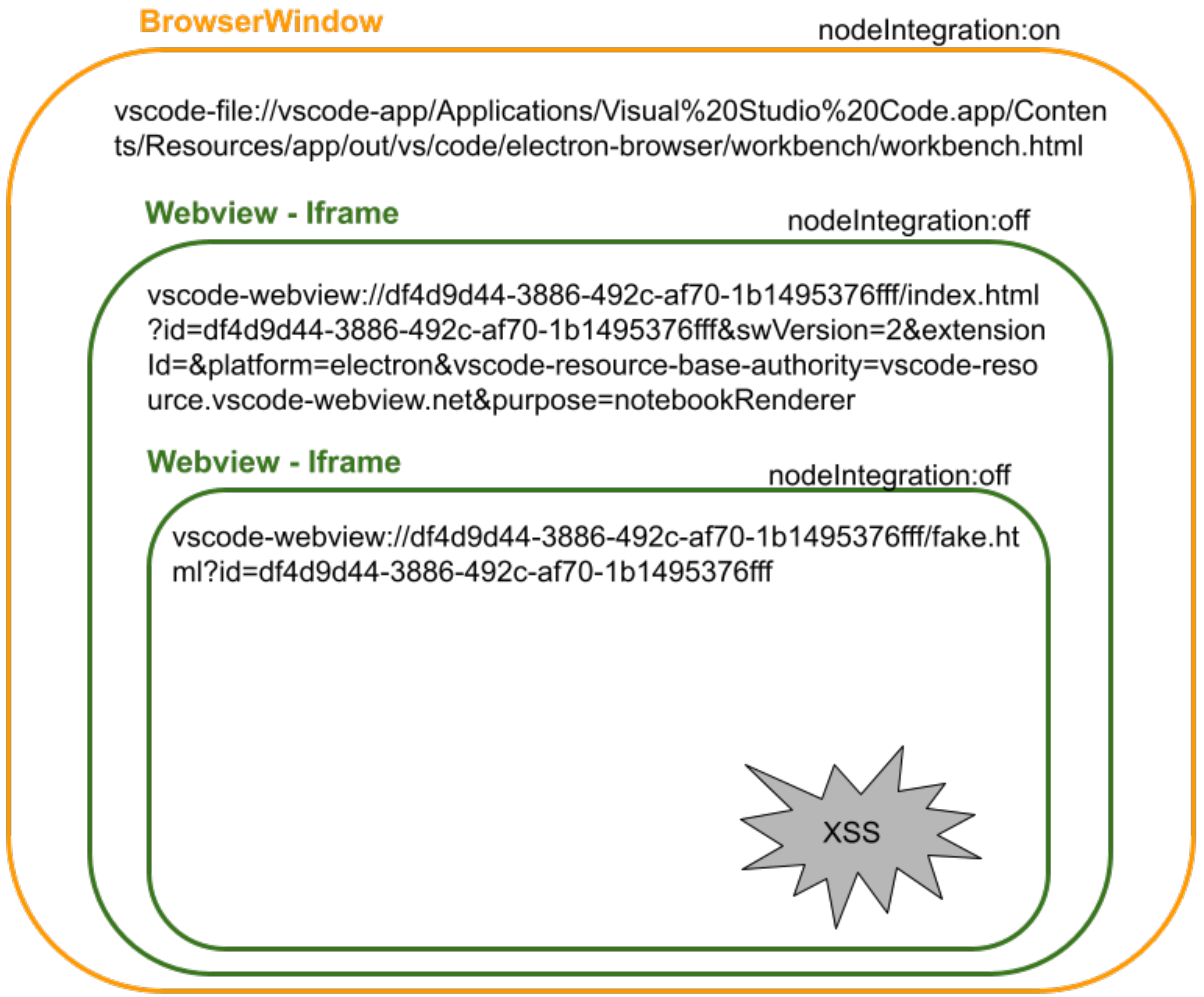
nodeIntegration bypasses,
webview tricks, …

3

Execute code

Leverage Node.js APIs

VScode DESIGN


BrowserWindow
nodeIntegration:on
vscode-file://vscode-app/Applications/Visual%20Studio%20Code.app/Contents/Resources/app/out/vs/code/electron-browser/workbench/workbench.html
Webview - Iframe
nodeIntegration:off
vscode-webview://df4d9d44-3886-492c-af70-1b1495376fff/index.html?id=df4d9d44-3886-492c-af70-1b1495376fff&swVersion=2&extensionId=&platform=electron&vscode-resource-base-authority=vscode-resource.vscode-webview.net&purpose=notebookRenderer
Webview - Iframe
nodeIntegration:off
vscode-webview://df4d9d44-3886-492c-af70-1b1495376fff/fake.html?id=df4d9d44-3886-492c-af70-1b1495376fff
XSS

IFRAME SANDBOX

- By default, `sandbox` makes the browser treat the `iframe` as if it was coming from another origin
- Thanks to the `allow-same-origin` attribute, this limitation is lifted
- Assuming content from the `vscode-file://vscode-app/` origin, we could execute something like:

```
top.require('child_process').exec('id');
```

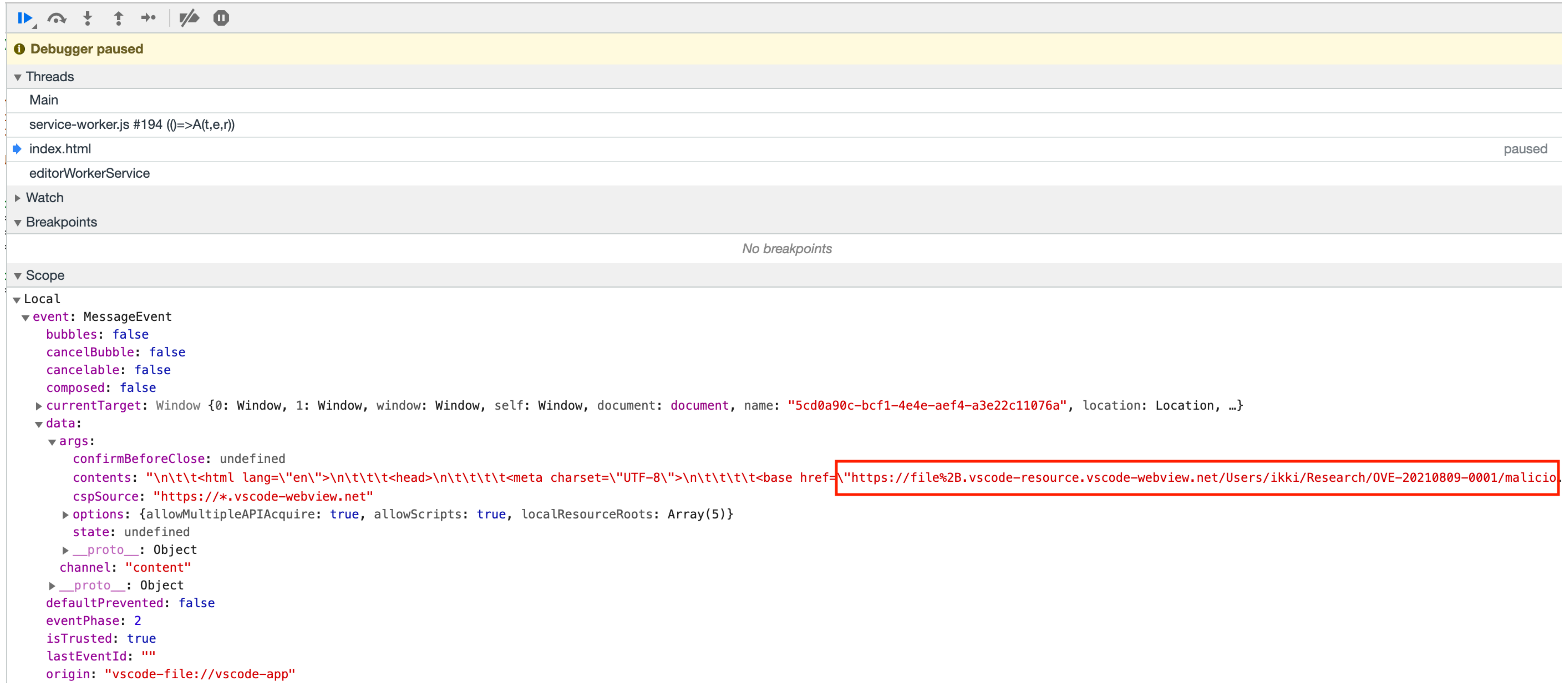
CVE-2021-43908

- Details disclosed at the latest BlackHat USA 2022
  - https://i.blackhat.com/USA-22/Thursday/US-22-Purani-ElectroVolt-Pwning-Popular-Desktop-Apps.pdf

  - `vscode-file://vscode-app/Applications/Visual Studio Code.app/Contents/Resources/app/..%2F..%2F..%2F..%2F..%2F..%2F..%2F..%2F..%2F..%2F..%2F..%2F/file.html`

POSTMESSAGE EVENTS

- Similarly to CVE-2021-43908, we can leverage a postMessage's reply to leak the path of the image files loaded



```
Debugger paused
▼ Threads
  Main
  service-worker.js #194 (()=>A(t,e,r))
  index.html                                                          paused
  editorWorkerService
▶ Watch
▼ Breakpoints
                              No breakpoints
▼ Scope
▼ Local
  ▼ event: MessageEvent
      bubbles: false
      cancelBubble: false
      cancelable: false
      composed: false
    ▶ currentTarget: Window {0: Window, 1: Window, window: Window, self: Window, document: document, name: "5cd0a90c-bcf1-4e4e-aef4-a3e22c11076a", location: Location, …}
    ▼ data:
      ▼ args:
          confirmBeforeClose: undefined
          contents: "\n\t\t<html lang=\"en\">\n\t\t\t<head>\n\t\t\t\t<meta charset=\"UTF-8\">\n\t\t\t\t<base href=\"https://file%2B.vscode-resource.vscode-webview.net/Users/ikki/Research/OVE-20210809-0001/malicio
          cspSource: "https://*.vscode-webview.net"
        ▶ options: {allowMultipleAPIAcquire: true, allowScripts: true, localResourceRoots: Array(5)}
          state: undefined
        ▶ __proto__: Object
        channel: "content"
      ▶ __proto__: Object
      defaultPrevented: false
      eventPhase: 2
      isTrusted: true
      lastEventId: ""
      origin: "vscode-file://vscode-app"
```

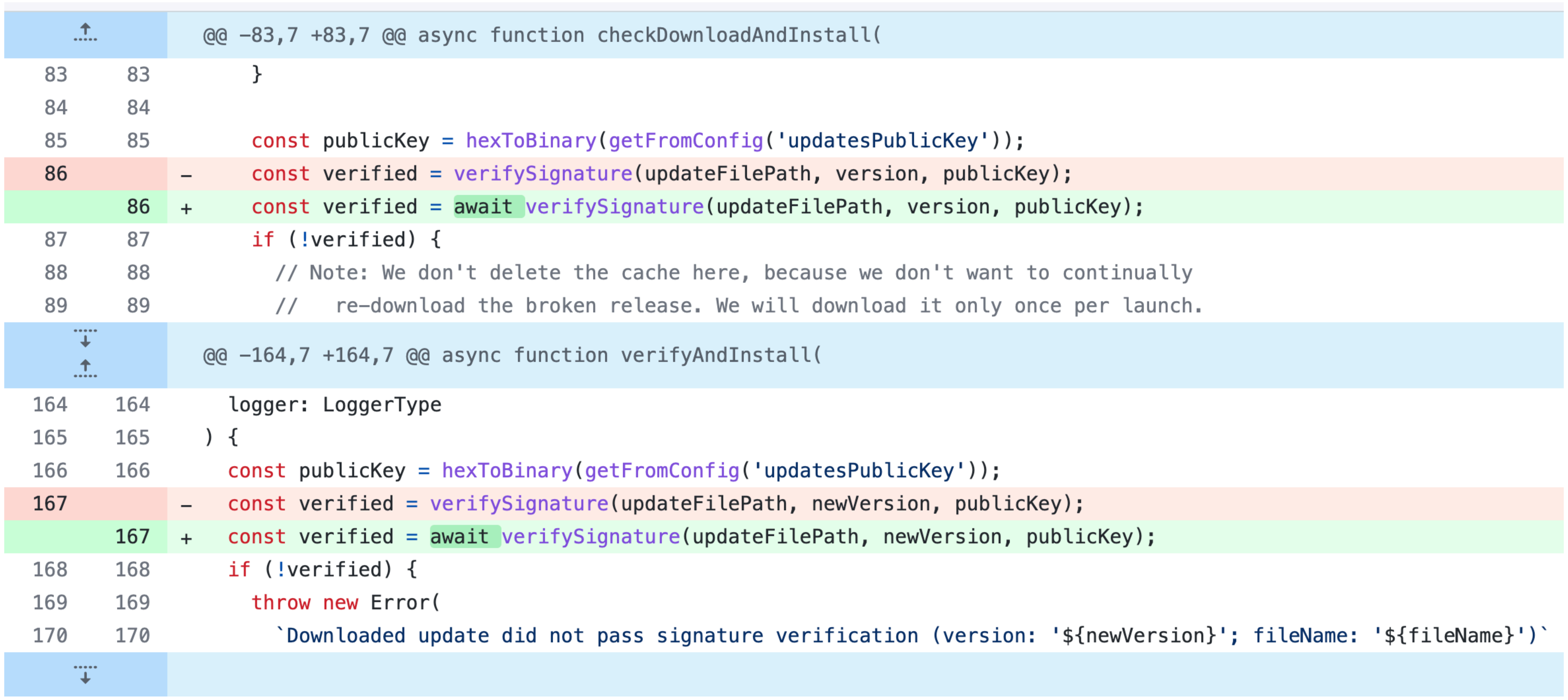
3

Not Keeping a “Promise” is the same as lying

Eric J. Dickey

LET'S START FROM THE END

https://github.com/signalapp/Signal-Desktop/commit/9d88abdb9006527bd7d1e3dea5443646af954875 (Aug 6, 2019)



```
@@ -83,7 +83,7 @@ async function checkDownloadAndInstall(
      }

      const publicKey = hexToBinary(getFromConfig('updatesPublicKey'));
-     const verified = verifySignature(updateFilePath, version, publicKey);
+     const verified = await verifySignature(updateFilePath, version, publicKey);
      if (!verified) {
        // Note: We don't delete the cache here, because we don't want to continually
        //   re-download the broken release. We will download it only once per launch.
@@ -164,7 +164,7 @@ async function verifyAndInstall(
    logger: LoggerType
  ) {
    const publicKey = hexToBinary(getFromConfig('updatesPublicKey'));
-   const verified = verifySignature(updateFilePath, newVersion, publicKey);
+   const verified = await verifySignature(updateFilePath, newVersion, publicKey);
    if (!verified) {
      throw new Error(
        `Downloaded update did not pass signature verification (version: '${newVersion}'; fileName: '${fileName}')`
```

THEN, WHAT?

- Verification mechanism for software updates is based on a lightweight Ed25519 public-key signature verification
- The function in use is defined as
  ```
  export async function verifySignature(...)
  ```
- The code does not wait for the promise’s return value

Definitely not something you expect in a signature verification routine

4

“Cloud is about how you do computing, not where you do computing”

Paul Maritz

AWS SDK CREDENTIALS

- When the AWS client is initialized without directly providing the credential’s source, a credential provider chain is used

- For Golang:
  1. Environment variables
  2. Shared credentials file
  3. If the application uses ECS task definition or RunTask API operation, IAM role for tasks
  4. If the application is running on an Amazon EC2 instance, IAM role for Amazon EC2

“Import Data From S3”

```
...
if err != nil {
    if err, awsError := err.(awserr.Error); awsError {
      aws_config.credentials = nil
      getObjectsList(session_init, aws_config, bucket_name)
    }
}
```

- More details in https://blog.doyensec.com/2022/10/18/cloudsectidbit-dataimport.html
- Credits to Mohamed Ouad, Francesco Lacerenza

Demo

…or backup video

5

“There’s so much pollution in ~~the air~~ Javascript now that if it weren’t for our ~~lungs~~ apps there’d be no place to put it all”

Robert Orben (not really)

Prototype Pollution

- JavaScript is prototype-based
- Object inheritance gives flexibility, but it’s dangerous

```
let user = {name: "luca"}
console.log(user.toString())

user.__proto__.toString = ()=>{alert(1)}
console.log(user.toString())
```

Prototype Pollution in TypeORM 0.2.35 - 0.3.9

- TypeORM is a JS/TS ORM
- Deep `Object.assign` is implemented in `mergeDeep()` https://github.com/typeorm/typeorm/blob/e92c743fb54fc404658fcaf2254861b6aa63bd98/src/util/OrmUtils.ts#L66
- A SQL injection can be triggered with the following payload

```
const post = JSON.parse(`{"text":"a","title":{"__proto__":
{"where":{"name":"foobar","where":null}}}}`)
```

Prototype Pollution in TypeORM 0.2.35 - 0.3.9

- More details in https://doyensec.com/resources/Doyensec_Advisory_TypeORM_Q32022.pdf
- Credits to Norbert Szetei, Viktor Chuchurski
- Original discovery: Francesco Soncina (phra)

Demo

…or backup video

6

“It’s all about parsing parsing parsing…”

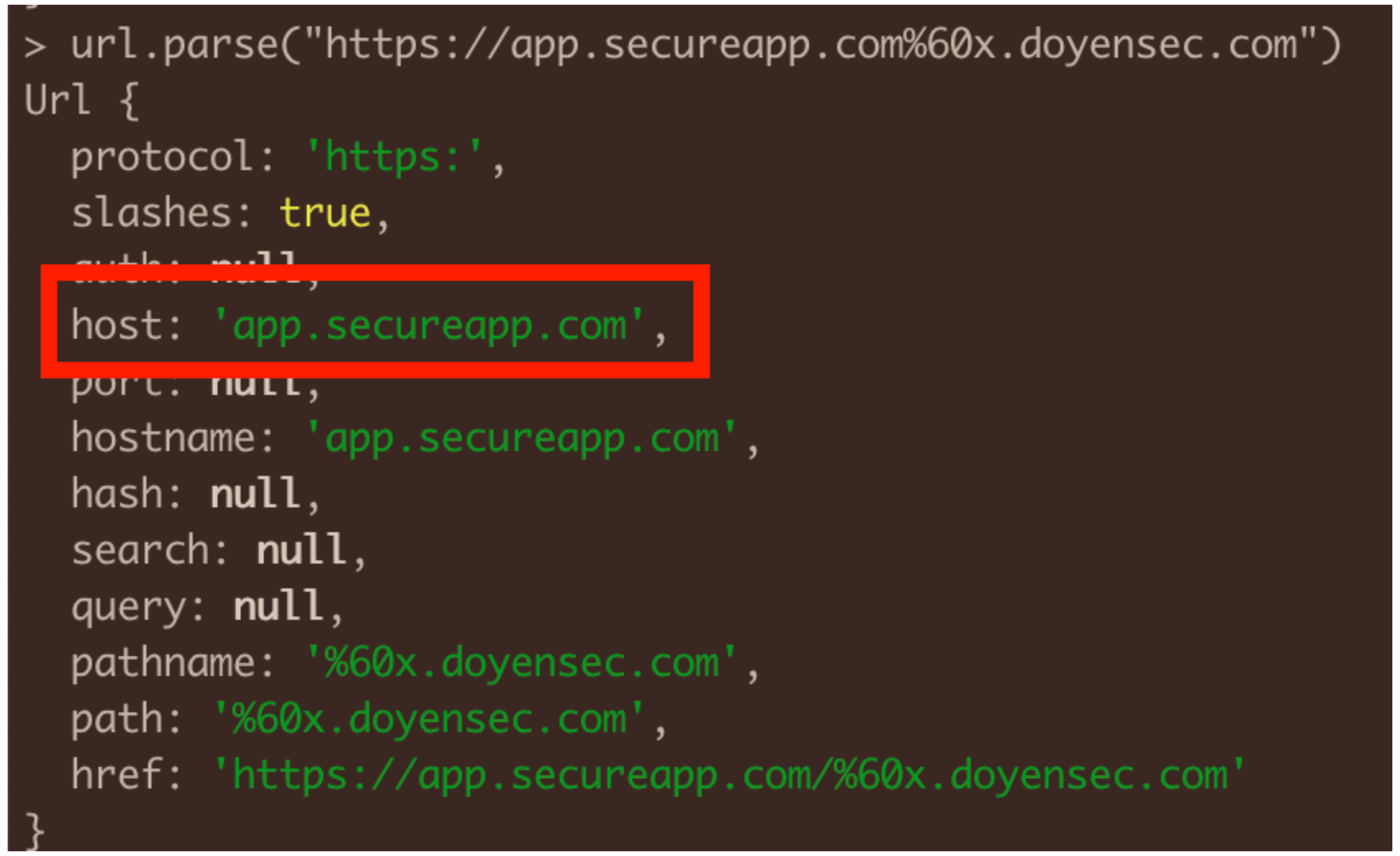
Meja

- The application implements validation to prevent open redirects

```
const sanitizeReturnTo = (returnTo: string) => {
 if (!returnTo) return;

 const { protocol, host } = url.parse(returnTo);
 if (protocol !== "https:" || host !== “app.secureapp.com”) return;

 return returnTo;
};
```

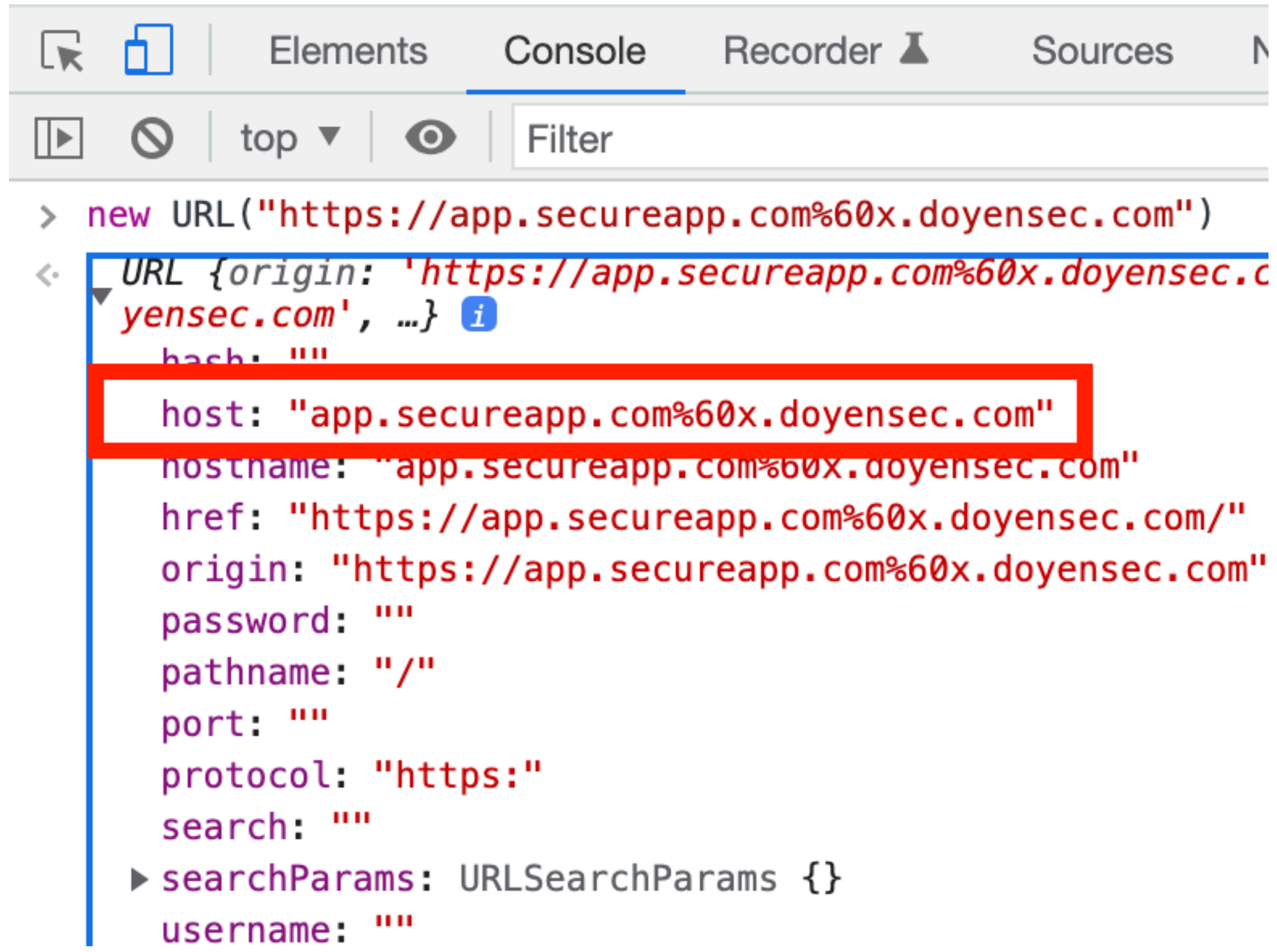
FE / BE Mismatch

NodeJS



```
> url.parse("https://app.secureapp.com%60x.doyensec.com")
Url {
  protocol: 'https:',
  slashes: true,
  auth: null,
  host: 'app.secureapp.com',
  port: null,
  hostname: 'app.secureapp.com',
  hash: null,
  search: null,
  query: null,
  pathname: '%60x.doyensec.com',
  path: '%60x.doyensec.com',
  href: 'https://app.secureapp.com/%60x.doyensec.com'
}
```

JavaScript



```
> new URL("https://app.secureapp.com%60x.doyensec.com")
< ▼URL {origin: 'https://app.secureapp.com%60x.doyensec.c
  yensec.com', …}
    hash: ""
    host: "app.secureapp.com%60x.doyensec.com"
    hostname: "app.secureapp.com%60x.doyensec.com"
    href: "https://app.secureapp.com%60x.doyensec.com/"
    origin: "https://app.secureapp.com%60x.doyensec.com"
    password: ""
    pathname: "/"
    port: ""
    protocol: "https:"
    search: ""
  ▸ searchParams: URLSearchParams {}
    username: ""
```

Conclusions

+

Tips&Tricks

Log4Shell
ProxyLogon
Pwn2Own Targets

...

Web security is no longer a 2nd class citizen

Trends

A Safe Internet

- CSRF is almost dead
- Traditional XSS is slowly disappearing
- Injection bugs are getting rare
- Secure by default frameworks
- A lot more investments

Job Stability

- HTTP Splitting
- HTTP Caching
- SSRF
- Prototype Pollution
- Parsing mismatch
- API Path Traversal
- Incorrect use of APIs, Functions, Cloud Services
- Business logic bugs
- Vulns Chaining

For Auditors

READ THE MANUAL

You can find bugs, even before you open Burp Suite

NEW STUFF

Look out for new technologies and trends. Never stop learning

SPARSE or DENSE

Look for the same bug in different places. Look for different bugs in the same place.

COMPLEXITY

Complexity is the enemy of security. Laser focus on large and complex code and systems

INTERCONNECTION

Look at how systems interconnect. The boundaries are the attack surface

MISMATCH

Parsing (and other mismatch-prone functionalities) have always been a good source of bugs

For Developers

READ THE MANUAL

Secure by default.
Yet, secure coding practices are still required

NEW STUFF

New doesn't always mean better. Also, different paradigms

SPARSE or DENSE

Don't allow the intern to push production code

COMPLEXITY

Complexity is the enemy of security. KISS

INTERCONNECTION

Integration tests anyone?!

MISMATCH

Whenever possible minimize technologies and implementation of the same business logic

THANKS!

Any questions?

You can find me at

luca@doyensec.com

@lucacarettoni