

# MURAENA

antisnatchor & ohpe present

THE  
UNEXPECTED  
PHISH



# OUTLINE

---

- ▶ Demystifying 2FA
- ▶ Reverse Proxy strikes back
  - ▶ Reversing anti-reverse proxy
- ▶ The art of instrumenting stolen web sessions
  - ▶ Bypassing anti-browser instrumentation
  - ▶ Automating post-phishing activities
- ▶ Full-chain demos on GSuite, GitHub, Dropbox
- ▶ Upcoming challenges

# DEMYSTIFYING 2FA: NON-U2F MYTHS

---

- ▶ You would immediately realise something dodgy is going on if you receive a 2FA challenge/token for an action you did not perform
- ▶ If attackers know your credentials, they can't login
- ▶ If you set up 2FA you are more secure (even via SMS)

# DEMYSTIFYING 2FA: NON-U2F MYTHS

---

- ▶ You would immediately realise something dodgy is going on if you receive a 2FA challenge/token for an action you did not perform
- ▶ Potentially true, but it doesn't consider scenarios where both credentials and tokens are phished in real-time
- ▶ If attackers know your credentials, they can't login
- ▶ It minimises only password spraying and other dictionary related attacks (Captcha would be enough)

# DEMYSTIFYING 2FA

- *If you set up 2FA you are more secure (even via SMS)*

MitiGator raises the bar...



...until it sees no more exploits

# DEMYSTIFYING 2FA: NON-U2F MYTHS

---

- ▶ Universal Two Factor (U2F) is the only 2FA solution that offers protection from phishing
  - ▶ Crypto challenge with the web origin
- ▶ Bypassed in early 2018 via WebUSB by @marver and @antisnatchor
  - ▶ OffensiveCon talk at: [youtube.com/watch?v=pUa6nWWTO4o](https://www.youtube.com/watch?v=pUa6nWWTO4o)

# REVERSE PROXY TO THE RESCUE

---

- ▶ Most 2FA solutions rely on token submission **via a web form**: SMS, Push/OTP based, Google/CompanyX authenticators
- ▶ After credentials are verified, the server triggers the 2FA action and waits for user token submission
- ▶ A smart **reverse proxy** can then be used to:
  - ▶ **intercept** all the traffic
  - ▶ **fulfil** requests that trigger the 2FA action
  - ▶ **pass** post-2FA login session cookies to an instrumented browser that hijacks the victim's session

# REVERSE PROXY TO THE RESCUE

---

- ▶ There is no magic here (and don't say you didn't know):



# REVERSE PROXY TO THE RESCUE

---

- ▶ Reverse proxies exist from over 20 years
- ▶ Technical Trends in Phishing Attacks (US CERT) - 2005

[https://www.us-cert.gov/sites/default/files/publications/phishing\\_trends0511.pdf](https://www.us-cert.gov/sites/default/files/publications/phishing_trends0511.pdf)

## 3.4.4 Man-in-the-Middle Attacks

Man-in-the-middle attacks define a broad class of potential attacks in which an attacker is able to intercept, read, and modify communications between two other parties without their knowledge. As related to phishing, a man-in-the-middle attack involves an attacker serving as a proxy between a user and an online commerce site. The attacker potentially has access to all authentication and account information, including an opportunity to hijack credentials used in two-factor authentication.

- ▶ So how did the situation evolved from 2005?

# REVERSE PROXY TO THE RESCUE

---

- ▶ In the last 4/5 years they have been hot again and used for offensive purposes:
  - ▶ (2015) <https://labs.portcullis.co.uk/blog/blood-in-the-water-phishing-with-beef/>
  - ▶ (2017) <http://www.chokepoint.net/2017/03/reverse-proxy-phishing-with-valid.html>
  - ▶ (2017) <https://breakdev.org/evilginx-advanced-phishing-with-two-factor-authentication-bypass/>

# REVERSE PROXY TO THE RESCUE

- ▶ And as expected on APTs:
  - ▶ (2017) <https://citizenlab.ca/2017/02/nilephish-report/>
  - ▶ (2018) <https://www.amnesty.org/en/latest/research/2018/12/when-best-practice-is-not-good-enough/>



- ▶ How did the community react to this?

# REVERSE PROXYING HEADERS

ALICE  
(PHISHING CLIENT)

EVILCORP  
(PHISHING ENGINE)  
HTTPS://PORTAL.BOB.CO

BOB  
(PHISHING TARGET)  
HTTPS://PORTAL.BOB.COM

GET /

REQUEST HEADERS  
- QUERY STRING  
- HOST  
- ORIGIN  
- REFERER  
- COOKIE  
- X-HEADERS

PATCH  
TLD

MODIFIED REQUEST  
PROCESSED

ORIGINAL  
HTTP RESPONSE

HTTP/1.1 200 OK

- CORS (ACAC/ACAC)

- ORIGIN

- SECURITY (CSP, FRAMING, MSTS, ANTI-SNIFFING)

- WWW\*

- COOKIE

- LOCATION

RESPONSE  
HEADERS

# REVERSE PROXYING BODY

ALICE  
(PHISHING CLIENT)

GET /

EVILCORP  
(PHISHING ENGINE)  
HTTPS://PORTAL.BOB.CO

HTTP REQUEST BODY

- MATCH/REPLACE  
FQDN

- UPDATE CONTENT LENGTH

HTTP RESPONSE BODY

- DECOMPRESSOR (GZIP  
DEFLATE  
BR)

- CONTENT-TYPE  
WHITELIST (PREVENTS  
PROXYING  
VIDEO/AUDIO  
IMAGE/CSS/  
BINARY)

- MATCH/REPLACE FQDN

BOB  
(PHISHING TARGET)  
HTTPS://PORTAL.BOB.COM

MODIFIED REQUEST  
PROCESSED

ORIGINAL  
HTTP RESPONSE

STATIC REPLACE

RULE-BASED REPLACE

- DEOBFUSCATE/MODIFY  
PACKED JAVASCRIPT

- CHANGE STRINGS  
VIA REGEXES

HTTP/1.1 200 OK

# REVERSE PROXY: MURAENA GOLANG IMPLEMENTATION

---

- ▶ We choose Golang for:
  - ▶ High performance, great syntax
  - ▶ Stable core and good library ecosystem
  - ▶ Cross-compilation
- ▶ `cloc . --exclude-dir vendor: ~2300 LoC`



# REVERSE PROXY: MURAENA GOLANG IMPLEMENTATION

---

- ▶ Muraena comes with a number of unique features
  - ▶ Web Crawler
  - ▶ Customisable Tracking
  - ▶ Static Content Server
  - ▶ Wildcard domain support
  - ▶ Browser Instrumentation Integration
- ▶ The proxy core is `net/http SingleHostReverseProxy` with a custom transformer to grep & replace strings in HTTP request/response

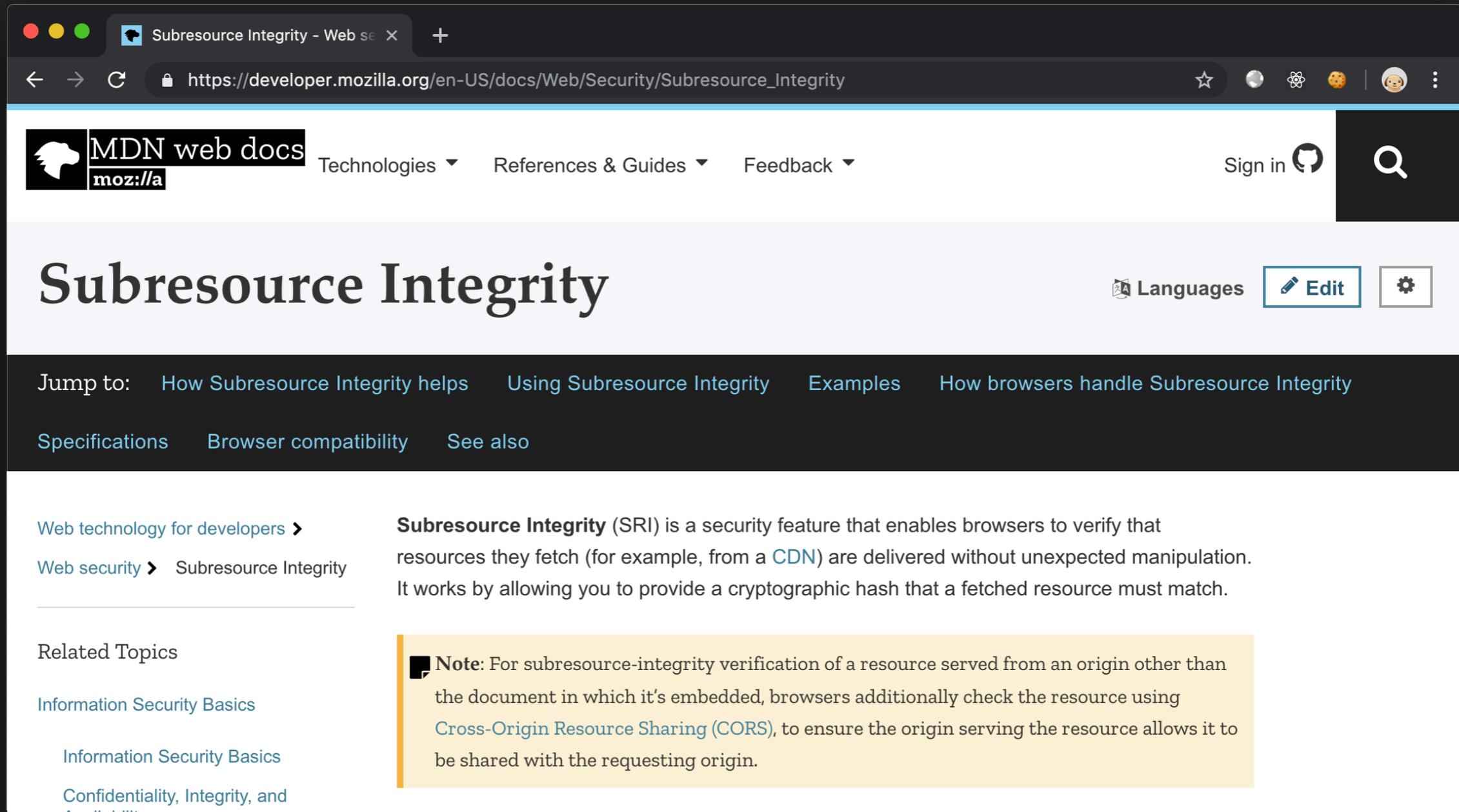
# REVERSING ANTI-REVERSE PROXY

---

- ▶ Some origins implement additional checks aimed at preventing proxying or framing
- ▶ Mostly easy checks, but reversing could get complicated with polymorphic heavily-obfuscated Javascript
- ▶ Chrome **DevTools** and **Burp Proxy** are your friends ;-)

# REVERSING ANTI-REVERSE PROXY

## ► Universal manipulations



The screenshot shows a web browser window displaying the MDN web docs page for Subresource Integrity. The browser's address bar shows the URL [https://developer.mozilla.org/en-US/docs/Web/Security/Subresource\\_Integrity](https://developer.mozilla.org/en-US/docs/Web/Security/Subresource_Integrity). The page header includes the MDN logo, navigation links for Technologies, References & Guides, and Feedback, and a search icon. The main heading is "Subresource Integrity". Below the heading, there are links for "Jump to:" including "How Subresource Integrity helps", "Using Subresource Integrity", "Examples", and "How browsers handle Subresource Integrity". There are also links for "Specifications", "Browser compatibility", and "See also". The main content area starts with a breadcrumb trail: "Web technology for developers > Web security > Subresource Integrity". The main text defines Subresource Integrity (SRI) as a security feature that enables browsers to verify that resources they fetch (for example, from a CDN) are delivered without unexpected manipulation. It works by allowing you to provide a cryptographic hash that a fetched resource must match. A yellow callout box contains a note: "Note: For subresource-integrity verification of a resource served from an origin other than the document in which it's embedded, browsers additionally check the resource using Cross-Origin Resource Sharing (CORS), to ensure the origin serving the resource allows it to be shared with the requesting origin." The left sidebar shows "Related Topics" with links to "Information Security Basics" and "Confidentiality, Integrity, and Availability".

Subresource Integrity

Jump to: [How Subresource Integrity helps](#) [Using Subresource Integrity](#) [Examples](#) [How browsers handle Subresource Integrity](#)

[Specifications](#) [Browser compatibility](#) [See also](#)

Web technology for developers >  
Web security > Subresource Integrity

**Subresource Integrity** (SRI) is a security feature that enables browsers to verify that resources they fetch (for example, from a [CDN](#)) are delivered without unexpected manipulation. It works by allowing you to provide a cryptographic hash that a fetched resource must match.

**Note:** For subresource-integrity verification of a resource served from an origin other than the document in which it's embedded, browsers additionally check the resource using [Cross-Origin Resource Sharing \(CORS\)](#), to ensure the origin serving the resource allows it to be shared with the requesting origin.

Related Topics

[Information Security Basics](#)

[Information Security Basics](#)

[Confidentiality, Integrity, and Availability](#)

# REVERSING ANTI-REVERSE PROXY

---

## ▶ Universal manipulations

```
<script src="https://example.com/example-framework.js"  
    integrity="sha384-oqVuAfXRKap7fdgcy5uykM6+R9GqQ8K/  
uxy9rx7HNQlGYl1kPzQho1wx4JwY8wC"  
    crossorigin="anonymous"></script>
```

becomes

```
<script src="https://example.com/example-framework.js"  
    no-integrity="sha384-oqVuAfXRKap7fdgcy5uykM6+R9GqQ8K/  
uxy9rx7HNQlGYl1kPzQho1wx4JwY8wC"  
    crossorigin="anonymous"></script>
```

# REVERSING ANTI-REVERSE PROXY

---

- ▶ Universal manipulations
- ▶ Content-Security Policy annihilation!

```
<script src="https://example.com/example-framework.js"  
  nonce="oqVuAfXRKap7fdgc"></script>
```

becomes

```
<script src="https://example.com/example-framework.js"  
  no-more-nonce="oqVuAfXRKap7fdgc"></script>
```

# REVERSING ANTI-REVERSE PROXY

---

- ▶ Universal manipulations
- ▶ Content-Security Policy annihilation!

```
<meta http-equiv="Content-Security-Policy" content="...">
```

becomes

```
<meta no-more-CSP content="...">
```

# REVERSING ANTI-REVERSE PROXY

---

- ▶ Universal manipulations
- ▶ Content-Security Policy annihilation!

```
"remove": {  
  ...,  
  "response": {  
    "header": [  
      "Content-Security-Policy",  
      "Content-Security-Policy-Report-Only",  
      ...  
    ]  
  }  
},
```

# REVERSING ANTI-REVERSE PROXY: GITHUB



## ► An easy one: GitHub

The screenshot shows a browser window with the address bar displaying `view-source:https://github.com`. The developer tools are open, showing the source code of the page. The code includes several meta tags and a JavaScript function. The console shows the output of the `atob` function, which decodes a Base64-encoded string into a JSON object.

```
92
93
94     <meta name="hostname" content="github.com">
95 <meta name="user-login" content="">
96
97     <meta name="expected-hostname" content="github.com">
98 <meta name="js-proxy-site-detection-payload"
content="MDI1YzE2ZmE0YzRlMDlmZmY4YzZmNTk5ODMiLCJ0aW1lc3RhbXAiOjE1NTcyNDAxOTUsImhvc3QiOiJnaXRodWIuY29tIn0=">
99
100 <meta name="enabled-features"
content="UNIVERSE_BANNER,MARKETPLACE_INVOICED_BILLING,MARKETPLACE_SOCIAL_PROOF_CUSTOMERS,MARKETPLACE_TRENDING_SOCIAL_PROOF,MARKETPLACE_RECOMMENDATIONS">
101
102 <meta name="html-safe-nonce" content="37f72641ec81fb48d10f5a10bb226d9024a84c4f">
```

Elements Console Sources Network Performance Memory Application Security Audits

top Filter All levels

```
> atob('MDI1YzE2ZmE0YzRlMDlmZmY4YzZmNTk5ODMiLCJ0aW1lc3RhbXAiOjE1NTcyNDAxOTUsImhvc3QiOiJnaXRodWIuY29tIn0=')
< "025c16fa4c4e09fff8c6f521104c234c76b31ed587605cb492524bcbbc24105e|{"remote_address":"[REDACTED]","request_id":"F428:7691:1FC82DE:356AF29:5CD19983","timestamp":1557240195,"host":"github.com"}"
```

# REVERSING ANTI-REVERSE PROXY: GITHUB



## ► An easy one: GitHub

The screenshot shows a browser window with a debugger. The address bar shows `view-source:https://github.com`. The page content is partially visible, showing the GitHub logo and navigation links. A debugger overlay is active, showing the source code of `frameworks-a0f1f08f.js`. The code is paused at line 9732, which contains a `const qc = document.querySelector("meta[name=js-proxy-site-detection-payload]")` statement. The debugger interface includes a console, a call stack, and a watch panel. The console shows the output of a function call: `> vt` and `< f Ae(e){const t=xe(e,"expected-hostname");return!!t&&t.replace(/\.\/, "").split(".").slice(-2).join(".")!==e.location.hostname.replace(/\.\/, "").split(".").slice(-2).join(".")}`. The call stack shows the function `execute` at line 9732 of `github-bootstra...formatted`.

# REVERSING ANTI-REVERSE PROXY: GITHUB



- ▶ An easy one: GitHub
- ▶ The fix: nullify the keywords:

`"js-proxy-site-detection-payload" -> ""`

`"expected-hostname" -> ""`

# REVERSING ANTI-REVERSE PROXY: DROPBOX



## ▶ The reCAPTCHA challenge

The screenshot shows a web browser window with the URL <https://www.dropbox.com>. The page features the Dropbox logo and navigation links for "For teams" and "For individuals". The main heading reads "Put your creative energy to work with Dropbox". Below this, a sub-heading states: "Dropbox is a modern workspace designed to reduce busywork – so you can focus on the things that matter."

On the right side of the page, there is a "Sign in" section with the following elements:

- Links for "Sign up" and "Download".
- Text: "Sign in" or [create an account](#).
- Input fields for "Email" and "Password".
- A red-bordered box containing the text: "This page is protected by reCAPTCHA and is subject to the Google [Privacy Policy](#) and [Terms of Service](#)."
- A checked checkbox labeled "Remember me".
- A blue "Sign in" button.
- A "Sign in with Google" button.
- A link for "Forgotten your password?".

# REVERSING ANTI-REVERSE PROXY: DROPBOX



## ▶ The reCAPTCHA challenge

The screenshot shows a browser window with two tabs labeled 'Dropbox'. The address bar displays `https://www.phishing.anti/#`. The page content is a dark red header with the Dropbox logo and navigation links 'For teams' and 'For individuals'. The main heading reads 'Put your creative energy to work with Dropbox'. On the right, there are 'Sign up' and 'Download' links, followed by input fields for 'Email' and 'Password'. Below these fields, a message states: 'This page is protected by reCAPTCHA and is subject to the Google Privacy Policy and Terms of Service.' A red error message from reCAPTCHA is visible: 'ERROR for site owner: Invalid domain for site key'. The browser's developer tools are open at the bottom, showing the DOM tree with a script element selected. The script's content is: `recaptcha.anchor.ErrorMain.init(["\x22ainput\x22,null,null,null,null,[1,1,1]\n,\x22Invalid domain for site key\x22,6,null,null,null,["\x22https://drb-17.phishing.anti/intl/en-GB/policies/privacy/\x22,\x22https://drb-17.phishing.anti/intl/en-GB/policies/terms/\x22]\n]\n");`. The error message in the DOM tree is: `Invalid domain for site key.`

# REVERSING ANTI-REVERSE PROXY: DROPBOX



## ▶ The reCAPTCHA challenge

The screenshot shows a browser window with the URL `https://www.phishing.anti/#`. The page displays the Dropbox logo and navigation links for "For teams" and "For individuals". A cookie consent banner is visible at the top. The developer tools network tab is open, showing a list of requests. The selected request is for `recaptcha_challenge-vfircf67y...` with the following details:

- Request URL:** `https://drb-17.phishing.anti/recaptcha/ap12/anchor?ar=1&k=6LdnLyIUAAAAA0iGptddh-g3KiJRoDGGPD-6dqXo&co=aHR0cHM6Ly93d3cucGhpc2hpbmCuYW50aTo0NDM'.&hl=en-GB&v=v1555968629716&size=invisible&cb=aqi3mkssrpd`
- Request Method:** GET
- Status Code:** 200 OK
- Remote Address:** 127.0.0.1:31337
- Referrer Policy:** origin-when-cross-origin
- Response Headers:**
  - Alt-Svc: quic=":443"; ma=2592000; v="46,44,43,39"
  - Cache-Control: private, max-age=0
  - Connection: close

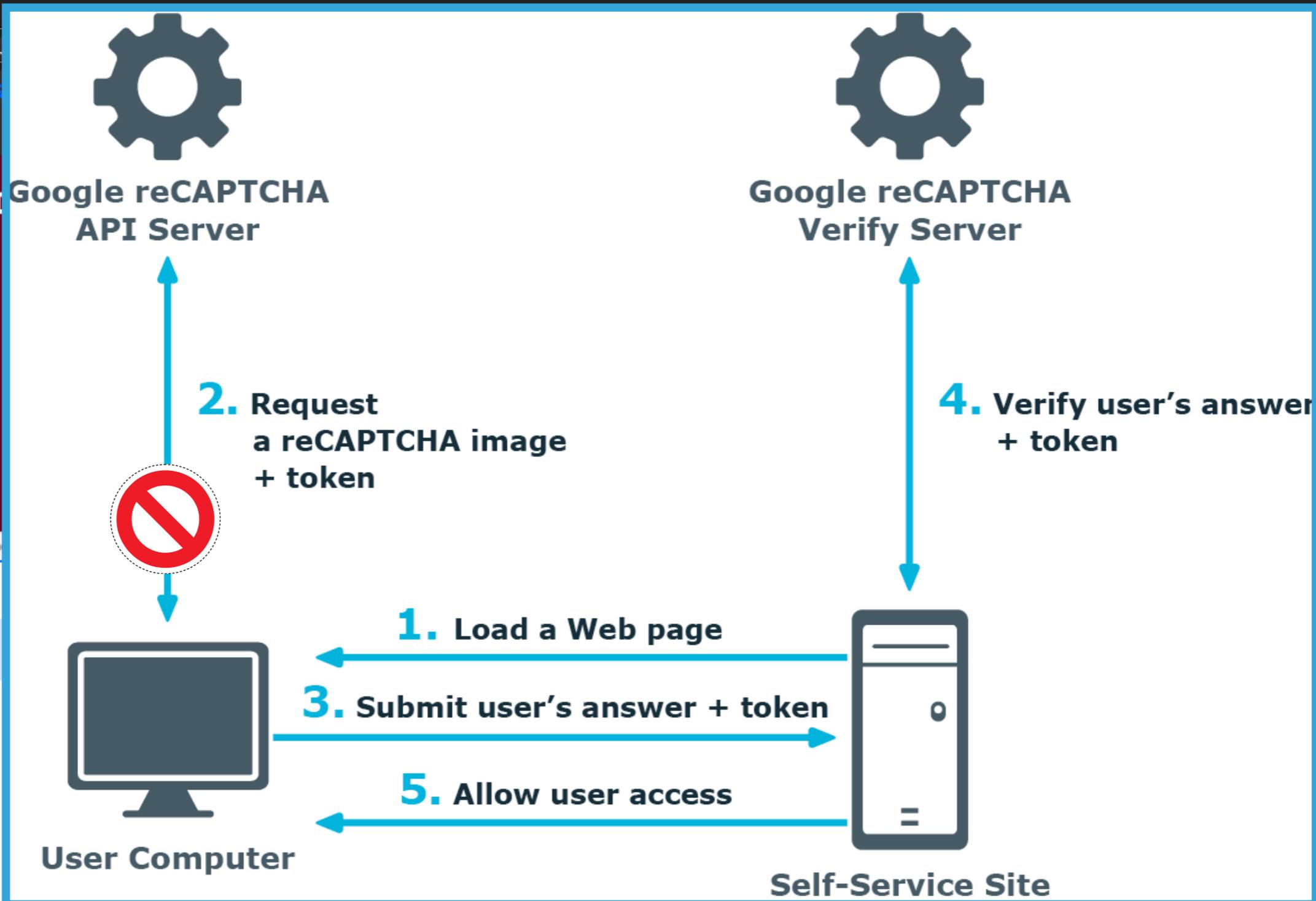
The console shows the following JavaScript code being executed:

```
atob('aHR0cHM6Ly93d3cucGhpc2hpbmCuYW50aTo0NDM')  
"https://www.phishing.anti:443"
```

# REVERSING ANTI-REVERSE PROXY: DROPBOX



## ▶ The reCAPTCHA challenge





- ▶ The reCAPTCHA challenge
- ▶ The fix: base64 transformation support:

```
"transform": {  
  "base64": {  
    "enabled": true,  
    "padding": [  
      "=",  
      "."  
    ]  
  },  
}
```

# REVERSING ANTI-REVERSE PROXY: GSUITE



- ▶ If it works for Google works for all



# REVERSING ANTI-REVERSE PROXY: GSUITE



- ▶ If it works for Google works for all
- ▶ There are several regexes to patch:

```
[ "(google)\\.com", "(google)|(phishing)(\\.anti|\\.com)" ],  
[".google\\.((co|com)", "(google|phishing)\\.((co|com|anti)" ],  
[".google\\.com", ".phishing\\.anti"],  
["\\.google(rs)?\\.com", "\\. (google(rs)|phishing)?\\. (com|anti)" ],  
["LCJwcHUi0iJodHRwczovL21haWwuZ29vZ2x1LmNvbS9yb2JvdHMudHh0IiwibHB1IjoiaHR0cHM6Ly9oYW5nb3V0cy5nb29nbGUuY29tL3JvYm90cy50eHQifQ",  
"LCJwcHUi0iJodHRwczovL21haWwucGhpc2hpbmcuYW50aS9yb2JvdHMudHh0IiwibHB1IjoiaHR0cHM6Ly9oYW5nb3V0cy5waGlzaGlzZy5hbnRpL3JvYm90cy50eHQifQ=="],  
],  
["LCJwcHUi0iJodHRwczovL2hhbmdvdXRzLmdvb2dsZS5jb20vcml9ib3RzLnR4dCI6ImxwdSI6Imh0dHBz0i8vaGFuZ291dHMucGhpc2hpbmcuYW50aS9yb2JvdHMudHh0In0",  
"LCJwcHUi0iJodHRwczovL2hhbmdvdXRzLnBoaXNoaW5nLmFudGkvcm9ib3RzLnR4dCI6ImxwdSI6Imh0dHBz0i8vaGFuZ291dHMucGhpc2hpbmcuYW50aS9yb2JvdHMudHh0In0="]
```

# THE ART OF INSTRUMENTING STOLEN WEB SESSIONS

---

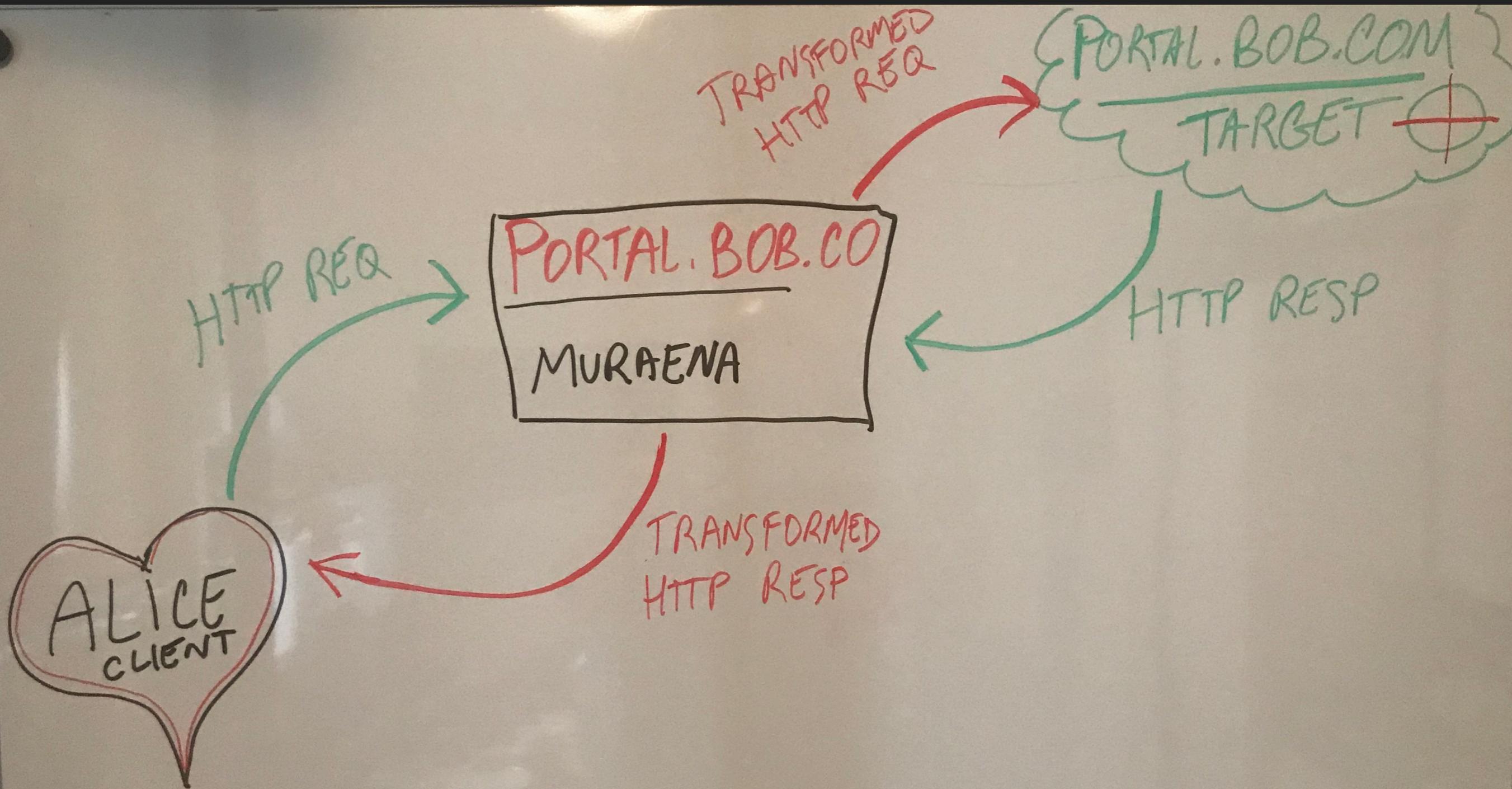
- ▶ Since all the traffic is passing through Muraena, credentials and session cookies are captured
- ▶ Is the targeted origin able to spot if we hijack the authenticated session passing it to an instrumented browser?

# THE ART OF INSTRUMENTING STOLEN WEB SESSIONS

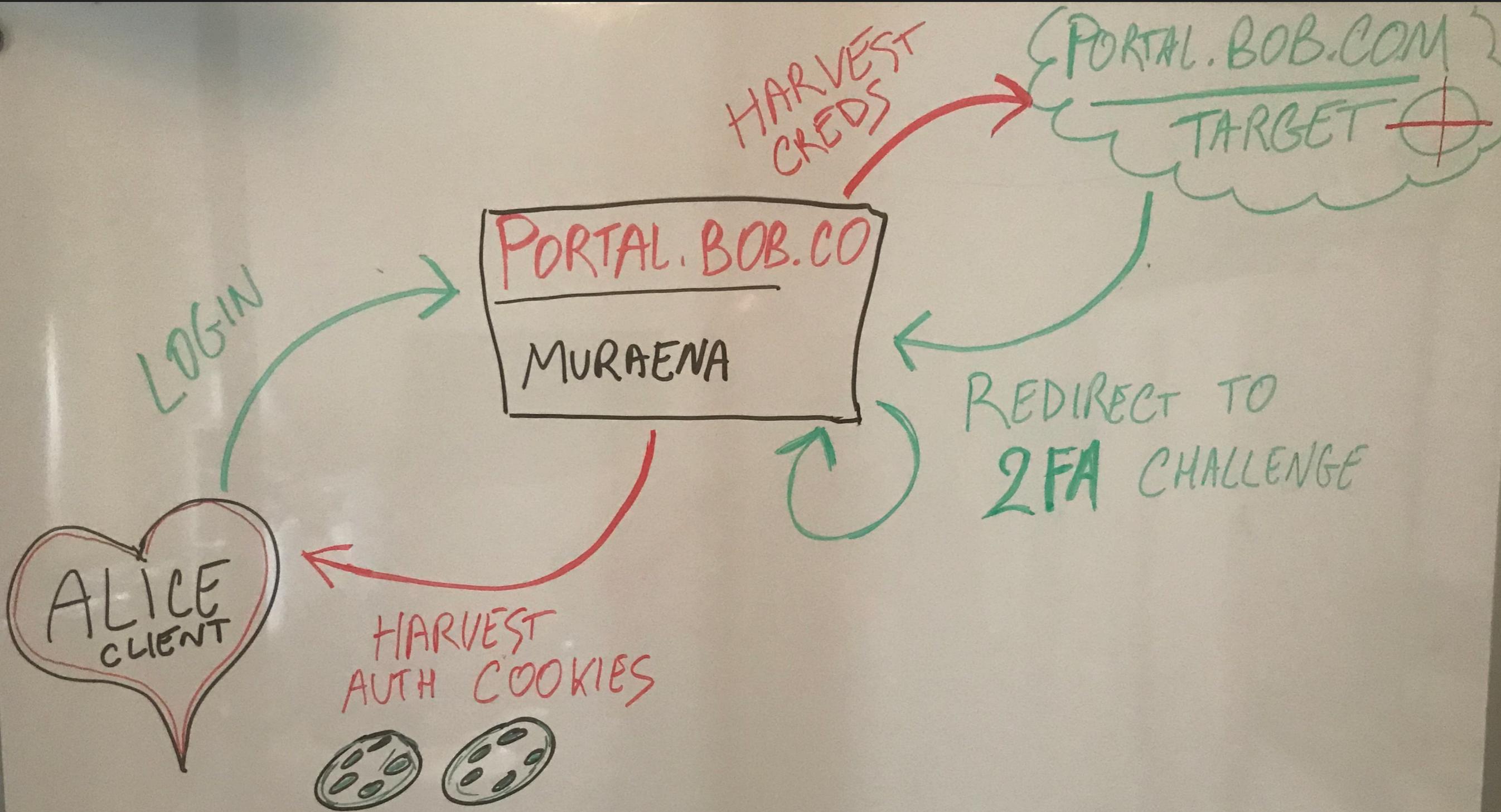
---

- ▶ Since all the traffic is passing through Muraena, credentials and session cookies are captured
- ▶ Is the targeted origin able to spot if we hijack the authenticated session passing it to an instrumented browser?
  - ▶ Usually **NO**
  - ▶ Additionally: the instrumented browser connection goes out via the same IP of Muraena, and the UA is changed to reflect the victim one.

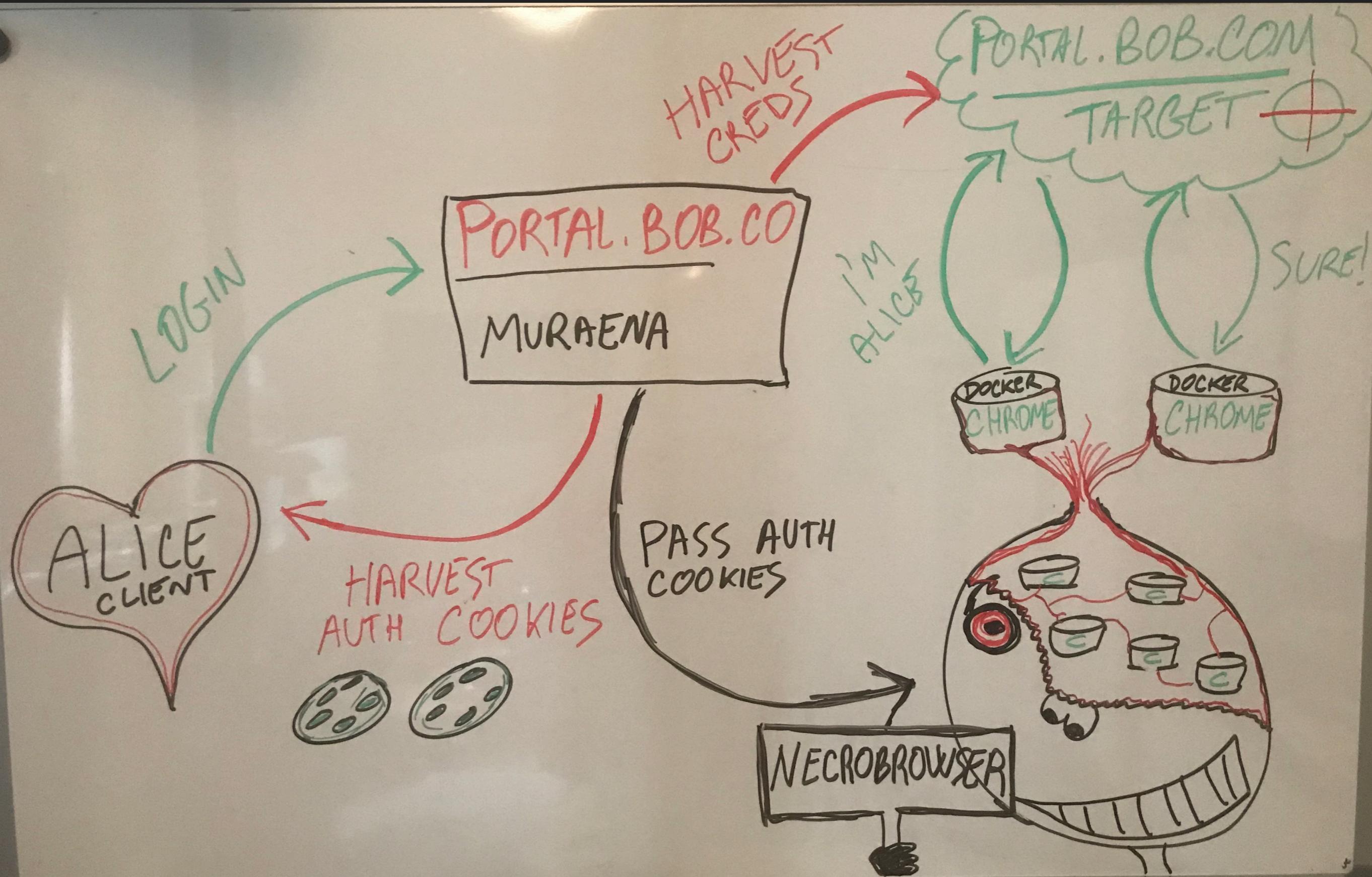
# MURAENA WITH NECROBROWSER



# MURAENA WITH NECROBROWSER



# MURAENA WITH NECROBROWSER



# NECROBROWSER

---

- ▶ NecroBrowser is a wrapper around ChromeDP (<https://github.com/chromedp/chromedp>)
- ▶ Programmatically drive Chrome via Chrome DevTools Protocol (CDP)
- ▶ Exposed as a micro service that spawns dedicated Docker containers with Chrome
- ▶ Allows to keep alive as many session as your Docker server/cluster can support
- ▶ Can be scheduled to do repeated actions (dump emails every hour, read Slack messages every minute)

# HEADLESS CHROME DETECTION

---



[http://antoinevastel.github.io/  
bot%20detection/2018/01/17/  
detect-chrome-headless-v2.html](http://antoinevastel.github.io/bot%20detection/2018/01/17/detect-chrome-headless-v2.html)

[https://intoli.com/blog/  
not-possible-to-block-  
chrome-headless/](https://intoli.com/blog/not-possible-to-block-chrome-headless/)

**Current status:**

Headless detection *\*failed\**.  
😏 Evaders are winning!

<https://github.com/paulirish/headless-cat-n-mouse>

# HEADLESS CHROME DETECTION

---

- ▶ <https://intoli.com/blog/making-chrome-headless-undetectable/>
- ▶ Simply, it's not easy to detect a non-human driven browser

Test Name	Result
User Agent (Old)	Mozilla/5.0 (X11; Linux x86_64)AppleWebKit/537.36 (KHTML, like Gecko) Chrome/64.0.3282.39 Safari/537.36
WebDriver (New)	missing (passed)
Chrome (New)	present (passed)
Permissions (New)	pending
Plugins Length (Old)	5
Languages (Old)	en-US,en

# AUTOMATING POST-PHISHING ACTIVITIES

---

- ▶ Examples could be endless, but the following are already implemented:
  - ▶ Disable GitHub Notifications, add SSH Key, download all repositories code
  - ▶ Disable GSuite Notifications, add Application Password, Dump Email
  - ▶ Upload arbitrary files to Dropbox, GDrive, Confluence
  - ▶ Screenshot and HTML dump all the things

# FULL-CHAIN DEMOS

---

- ▶ Come se fosse Antani col video registrato anche per lei, senno' son moccoli anche se non partissero come ieri?



# UPCOMING CHALLENGES

## ▶ Google:

### Better protection against Man in the Middle phishing attacks

April 18, 2019

However, one form of phishing, known as “[man in the middle](#)” (MITM), is hard to detect when an embedded browser framework (e.g., [Chromium Embedded Framework](#) - CEF) or another automation platform is being used for authentication. MITM intercepts the communications between a user and Google in real-time to gather the user’s credentials (including the second factor in some cases) and sign in. Because we can’t differentiate between a legitimate sign in and a MITM attack on these platforms, we will be blocking sign-ins from embedded browser frameworks starting in June. This is similar to the [restriction on webview](#) sign-ins announced in April 2016.

#### What developers need to know

The solution for developers currently using CEF for authentication is the same: [browser-based OAuth authentication](#). Aside from being secure, it also enables users to see the full URL of the page where they are entering their credentials, reinforcing good anti-phishing practices. If you are a developer with an app that requires access to Google Account data, switch to using browser-based OAuth authentication today.

# FUTURE WORK

---



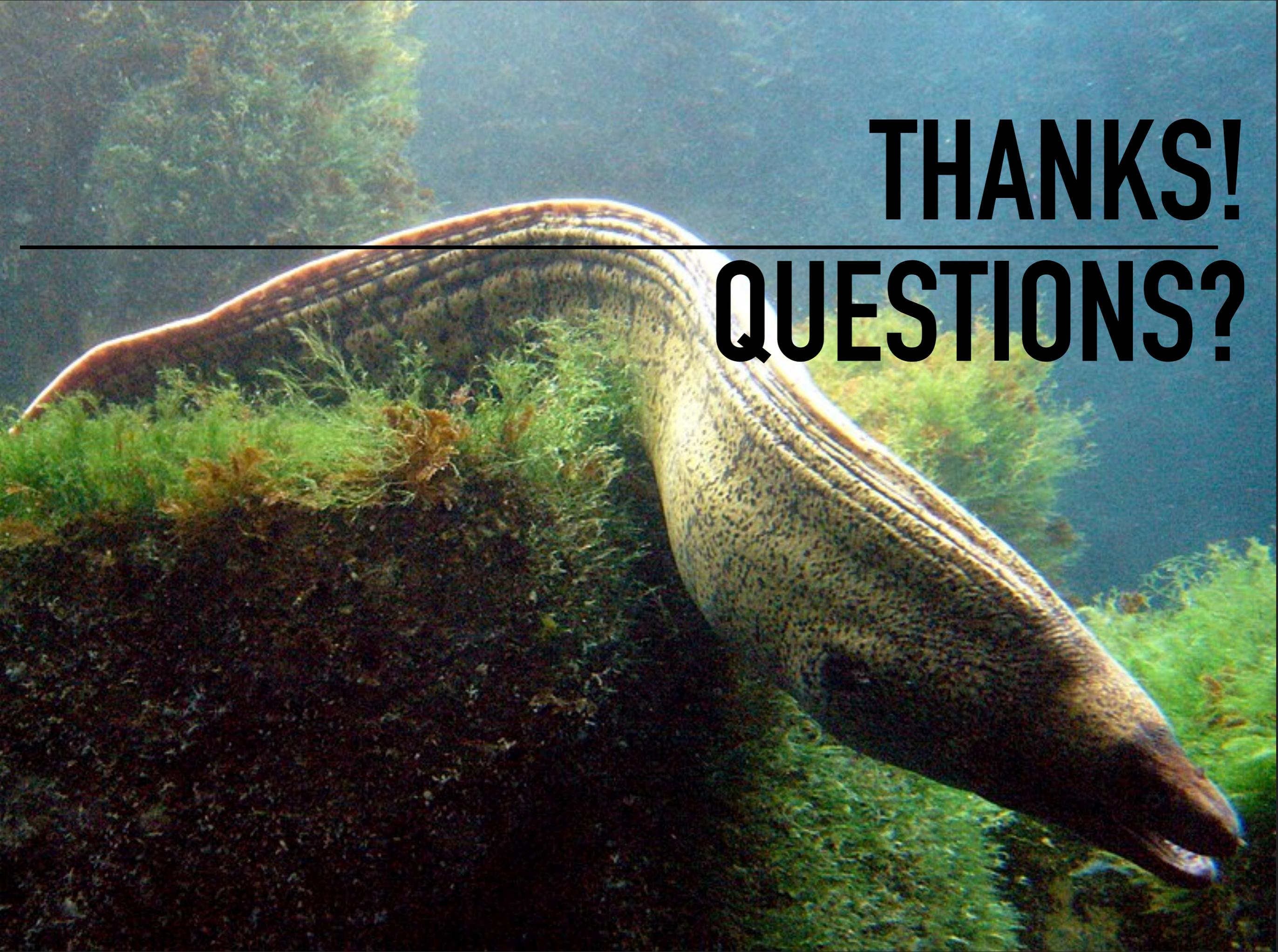
- ▶ Bettercap integration
- ▶ More browser automation fun:
  - ▶ Support for more commonly used web portals
  - ▶ Scaling tests, queuing instrumentation jobs
- ▶ Use browser instrumentation also for RECON/OSINT pre-phishing:
  - ▶ Scrape company X profile from LinkedIn/SocialNetworks using a real browser with a fake account

# WHERE TO FIND THE CODE

---



<https://github.com/muraenateam>



**THANKS!**

---

**QUESTIONS?**