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)
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).
• If you set up 2FA you are more secure (even via SMS)

MitiGator raises the bar...

...until it sees no more exploits

Credit @halvarflake
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
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.
There is no magic here (and don’t say you didn’t know):
Reverse proxies exist from over 20 years

Technical Trends in Phishing Attacks (US CERT) - 2005

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?
In the last 4/5 years they have been hot again and used for offensive purposes:

REVERSE PROXY TO THE RESCUE

- And as expected on APTs:
  - (2017) [https://citizenlab.ca/2017/02/nilephish-report/](https://citizenlab.ca/2017/02/nilephish-report/)

How did the community react to this?
REVERSE PROXYING HEADERS

**ALICE** (phishing client)

GET /

HTTP/1.1 200 OK

**EVILCORP** (phishing engine)

HTTPS://PORTAL.BOB.CO

REQUEST HEADERS

- Query string
- Host
- Origin
- Referer
- Cookie
- X-Headers

-CORS(ACAC/ACAC)

- Origin
- Security (CSP, framing, HSTS, anti-sniffing)

- WWW*
- Cookie
- Location

**BOB** (phishing target)

HTTPS://PORTAL.BOB.COM

MODIFIED REQUEST PROCESSED

ORIGINAL HTTP RESPONSE

RESPONSE HEADERS
**REVERSE PROXYING BODY**

**ALICE**  
(Phishing Client)

```
GET /
```

**EVILCORP**  
(Phishing Engine)

```
HTTPS://PORTAL.BOBCO
```

- HTTP Request Body
  - MATCH/REPLACE FQDN
  - UPDATE CONTENT LENGTH

- HTTP Response Body
  - DECOMPRESSOR (GZIP/DEFLATE/BR)
  - CONTENT-TYPE WHITELIST (PREVENTING PROXYING VIDEO/AUDIO/Image/CSS/BINARY)
  - MATCH/REPLACE FQDN

**BOB**  
(Phishing Target)

```
HTTPS://PORTAL.BOBCOM
```

- MODIFIED REQUEST PROCESSED
- ORIGINAL HTTP RESPONSE
- STATIC REPLACE
- RULE-BASED REPLACE
  - DEOBfuscate/modify packed JavaScript
  - CHANGE STRINGS VIA REGEXES
We choose Golang for:

- High performance, great syntax
- Stable core and good library ecosystem
- Cross-compilation

`cloc . --exclude-dir vendor: ~2300 LoC`
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

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.
REVERSING ANTI-REVERSE PROXY

- Universal manipulations

```html
<script src="https://example.com/example-framework.js" integrity="sha384-oqVuAfXRXKap7fdgcCY5uykM6+R9GqQ8K/uxy9rx7HNQlGYl1kPzQho1wx4JwY8wC" crossorigin="anonymous"></script>

becomes

```html
<script src="https://example.com/example-framework.js" no-integrity="sha384-oqVuAfXRXKap7fdgcCY5uykM6+R9GqQ8K/uxy9rx7HNQlGYl1kPzQho1wx4JwY8wC" crossorigin="anonymous"></script>
```
REVERSING ANTI-REVERSE PROXY

- Universal manipulations
- Content-Security Policy annihilation!

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

becomes

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

- Universal manipulations
- Content-Security Policy annihilation!

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

becomes

```html
<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
REVERSING ANTI-REVERSE PROXY: GITHUB

- An easy one: GitHub
REVERSING ANTI-REVERSE PROXY: GITHUB

- An easy one: GitHub

- The fix: nullify the keywords:

  "js-proxy-site-detection-payload" -> ""
  "expected-hostname" -> ""
The reCAPTCHA challenge
REVERSING ANTI-REVERSE PROXY: DROPBOX

- The reCAPTCHA challenge

```
<script type="text/javascript" nonce="pF3HjXiFbOzXiMGBj4Fw">
recaptcha.anchor.ErrorMain.init("\x22ainput\x22, mult, null, null, null, [1,1,1]\x22);\x22Invalid domain for site key\x22, mult, null, null, [\x22https://dbb-17.phishing.anti/int/en-GB/policies/privacy/\x22, \x22https://dbb-17.phishing.anti/int/en-GB/policies/terms/\x22]);

</script>
```
REVERSING ANTI-REVERSE PROXY: DROPBOX

- The reCAPTCHA challenge

![Image of Dropbox webpage and developer tools]

We use cookies so that Dropbox works for you. By using our website, you agree to our use of cookies. Learn more.
REVERSING ANTI-REVERSE PROXY: DROPBOX

» The reCAPTCHA challenge

1. Load a Web page
2. Request a reCAPTCHA image + token
3. Submit user’s answer + token
4. Verify user’s answer + token
5. Allow user access
REVERSING ANTI-REVERSE PROXY: DROPBOX

- The reCAPTCHA challenge

- The fix: base64 transformation support:

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

```json
[
  "|\(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)",
  "LCJwcHUiOiJodHRwczovL21haWwuZ29vZ2xlLmNvbS9yb2JvdHMudHh0IiwibHB1IjoiaHR0cHM6Ly
  9oYW5nb3V0cy5nb29nbGUuY29tL3JvYm90cy50eHQifQ",
  "LCJwcHUiOiJodHRwczovL21haWwucGhpc2hpbmcuYW50aS9yb2JvdHMudHh0IiwibHB1IjoiaHR0cHM6
  Ly9oYW5nb3V0cy5waGlzaGlzuZy5hbnaRpl3JvYm90cy50eHQifQ==",
  "LCJwcHUiOiJodHRwczovL2hhbmdvdXRzLmdvb2dsZS5jb20vcm9ib3RzMnR4dCI6ImxwdSI6Imh0dHBz
  0i8vaGFuZ291dHMuZ29vZ2xlLmNvbS9yb2JvdHMudHh0In0",
  "LCJwcHUiOiJodHRwczovL2hhbmdvdXRzLnBoaXNoaW5nLmFudGkvcm9ib3RzLnR4dCI6ImxwdSI6Imh0
  dHBz0i8vaGFuZ291dHMucGhpc2hpbmcyW50aS9yb2JvdHMudHh0In0=="
]
```
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?
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

HTTP REQ → PORTAL.BOB.CO

MURAENA

TRANSFORMED HTTP REQ → PORTAL.BOB.COM

TARGET

HTTP RESP

TRANSFORMED HTTP RESP

ALICE CLIENT
MURAENA WITH NECROBROWSER

ALICE

CLIENT

HARVEST AUTH COOKIES

PORTAL.BOB.CO

MURAENA

LOGIN

HARVEST CRED

TARGET

REDIRECT TO 2FA CHALLENGE
MURAENA WITH NECROBROWSER

- **Portal.bob.co**
- **Login**
- **Harvest creds**
- **Alice client**
- **Harvest auth cookies**
- **Pass auth cookies**
- **Necrobrowser**
- **Target**
- **Sure!**

DIAGRAM: Alice client authenticates with Portal.bob.co using Muraena, which harvests credentials. These credentials are passed to Necrobrowser, which then accesses the target. The process is illustrated with various symbols and labels.
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


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

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