

PUPPETEER FOR EVIL MINDS



ANTISNATCHOR

OUTLINE

- The need of automation
- Puppeteer fundamentals
- Automating Recon and Phishing
- Puppeteer for WebSecurity?
- Puppeteer detection
- The future

THE NEED OF AUTOMATION

- Many different contexts benefits from automation:
 - Reconnaissance
 - Phishing
 - Advanced simulations
 - Functional and Web Security testing

THE NEED OF AUTOMATION: RECON

- Searching and scraping info from web portals without relying on APIs (no rate limits, flexibility)
- Perform programmatic actions via fake profiles in a very realistic way
- Monitor hourly for content changes on social networks and web portals in general

THE NEED OF AUTOMATION: PHISHING

- The more targets, the more sessions you collect: doing manual work on the hijacked session rarely make sense, even in spear phishing scenarios
- Setting the victim's Cookies on a Chrome headless instance that:
 - add your SSH key to the target repository;
 - or add an Application Password to the profile;
 - Or dumps the contact list and send a dropper to all the marketing team?

THE NEED OF AUTOMATION: ADVANCED SIMULATIONS

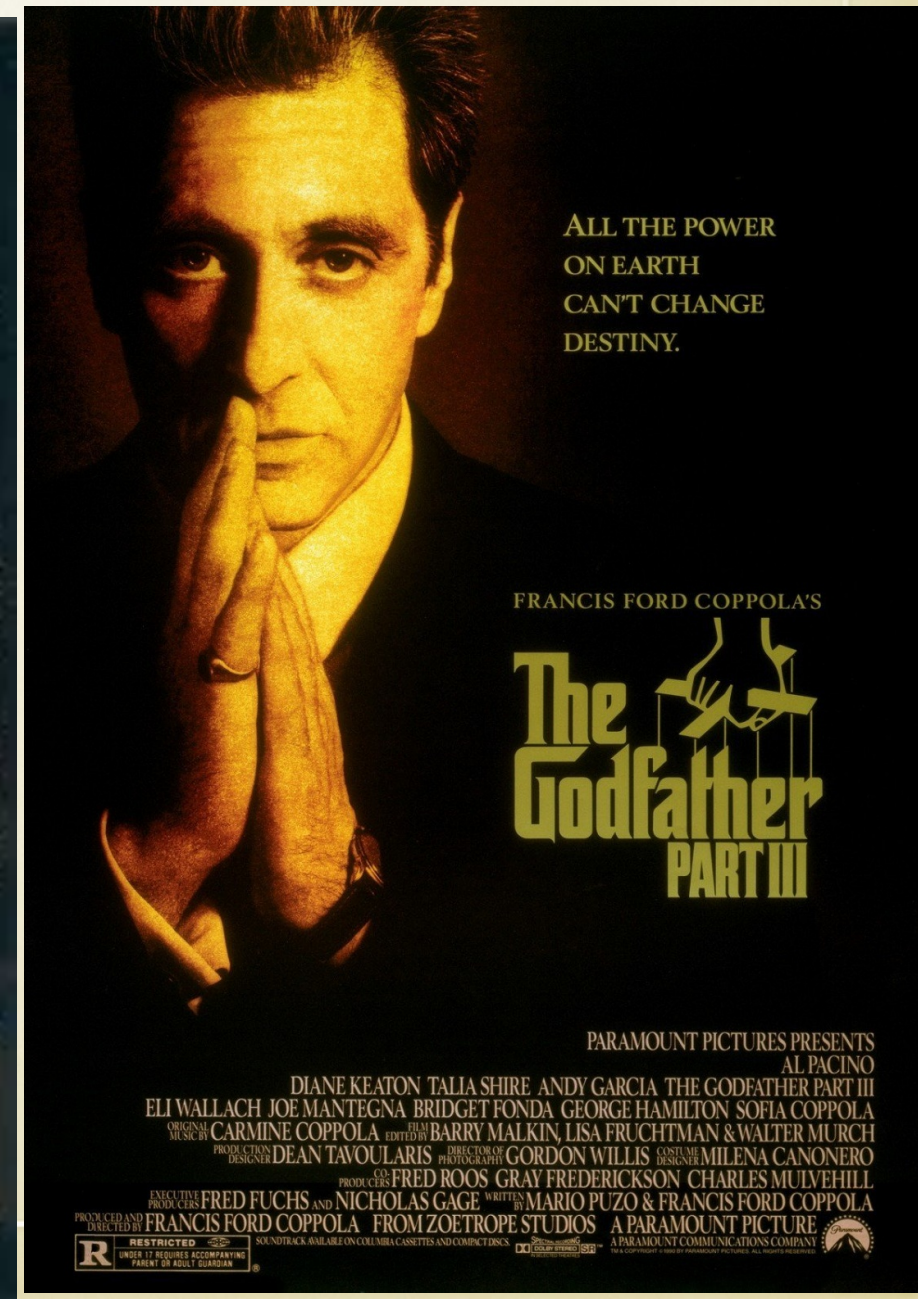
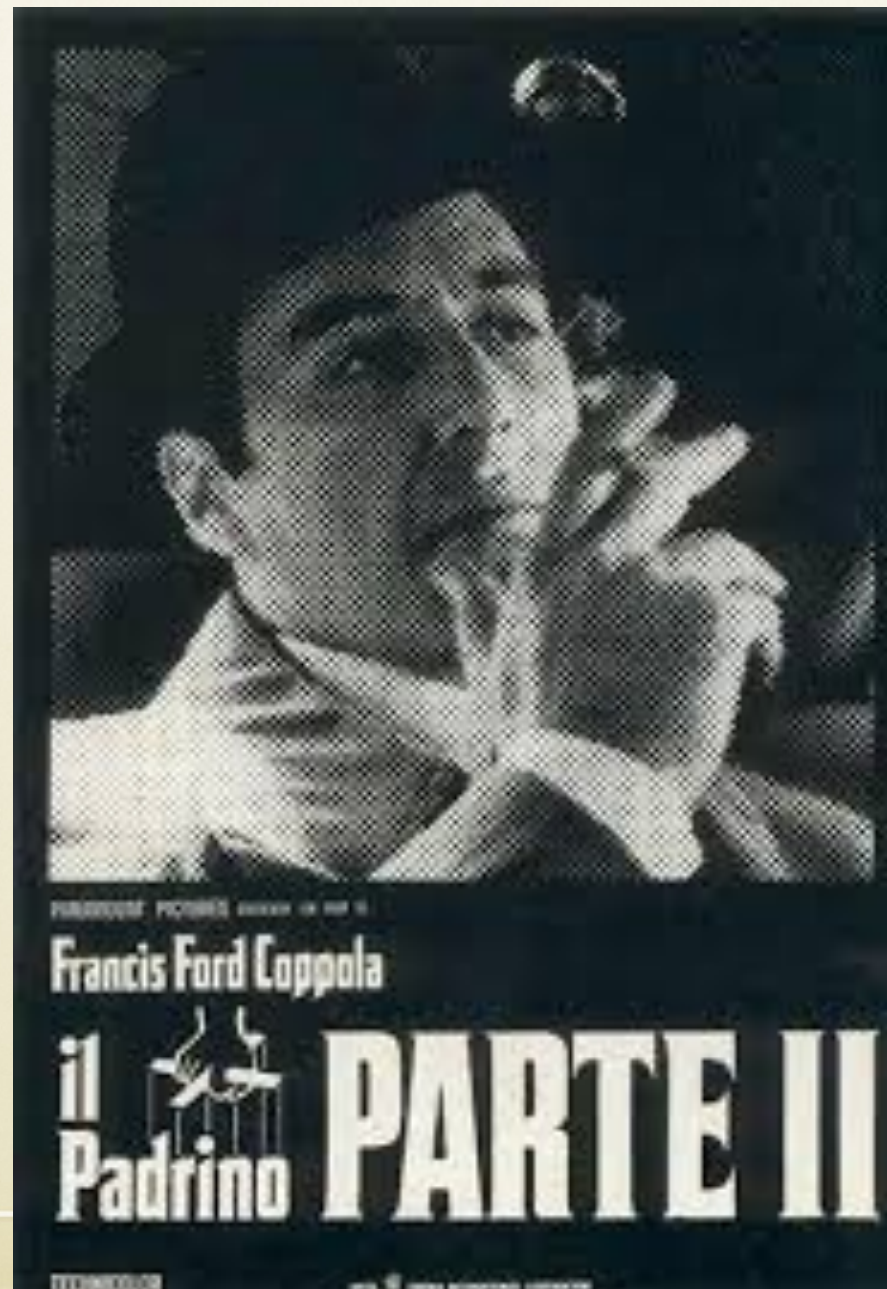
- Simulating N browsers that do N actions from N IPs, each of them with different fingerprint
- Automating complex web workflows that are harder to do without a browser

THE NEED OF AUTOMATION: TESTING & WEB SECURITY

- Where Burp Macros are not enough, browser automation comes to the rescue
 - Ex.: bugs or chains that need:
 - drag&drop or other mouse events
 - Weird JS apps and other browser detections



GOOGLE'S PUPPETEER
&&&
COPPOLA'S PADRINO





PUPPETEER

- Browser Automation Library bridging Chrome and NodeJS through CDP (Chrome DevTools Protocol)
- Created by Google to work on Google browser
- *Modern Web Testing and Automation with Puppeteer (Google I/O '19)* by Andrey Lushnikov & Joel Einbinder:
<https://www.youtube.com/watch?v=MbnATLCuKI4>



PUPPETEER

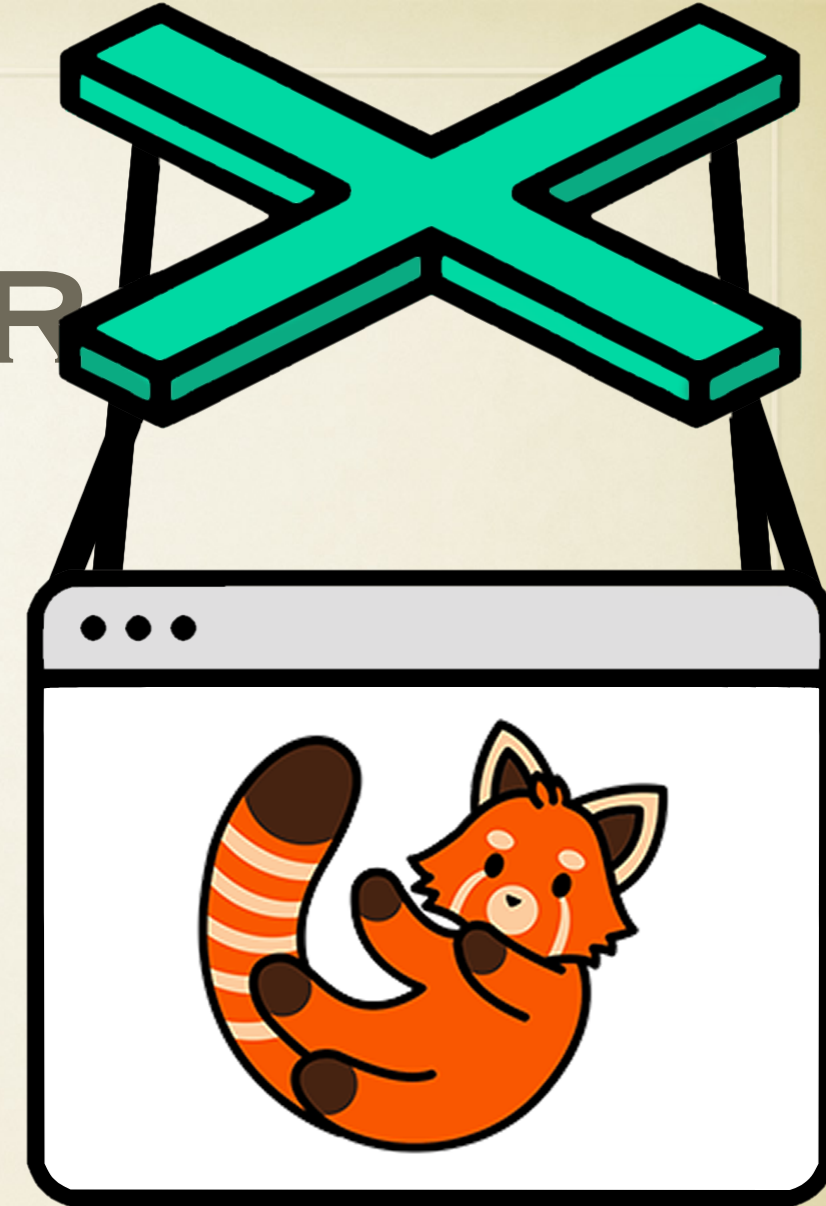
- Allows you to programmatically tell the browser, in GUI or headless mode, to do all the things
 - Set cookies, fill form inputs, click buttons
 - Reliably wait for elements and control the DOM
 - Achieve parallelism with multiple browser tabs



PUPPETEER

- Chrome DevTools Protocol is the key
 - Fast and reliable over WebSockets
 - Also used by Chrome Inspect View to inspect and control the DOM dynamically
- 1. `chrome --remote-debugging-port=9222`
- 2. `chrome --user-data-dir=<dir>`
- 3. browse to `http://localhost:9222`

PUPPETEER



- From a sane NodeJS environment: `npm install puppeteer`
- There is **experimental support** for **Firefox**: `npm install puppeteer-firefox`
- <https://aslushnikov.github.io/ispuppeteerfirefoxready/>

- Tests Passing: **82%** (517/632)
- Supported API: **89%** (222/249)
- Last updated: **71 days ago**



PUPPETEER API

- class: Page
 - event: 'close'
 - event: 'console'
 - event: 'dialog'
 - event: 'domcontentloaded'
 - event: 'error'
 - event: 'frameattached'
 - event: 'framedetached'
 - event: 'framenavigated'
 - event: 'load'
 - event: 'metrics'
 - event: 'pageerror'
 - event: 'popup'
 - event: 'request'
 - event: 'requestfailed'
 - event: 'requestfinished'
 - event: 'response'
 - event: 'workercreated'
 - event: 'workerdestroyed'
 - page.\$(selector)
 - page.\$\$\$(selector)
 - page.\$\$eval(selector, pageFunction[, ...args])
 - page.\$eval(selector, pageFunction[, ...args])
 - page.\$x(expression)
 - page.accessibility
 - page.addScriptTag(options)
 - page.addStyleTag(options)
 - page.authenticate(credentials)
 - page.bringToFront()
 - page.browser()
 - page.browserContext()
 - page.click(selector[, options])
 - page.close([options])
 - page.content()
 - page.cookies([...urls])
 - page.coverage
 - page.deleteCookie(...cookies)
 - page.emulate(options)
 - page.emulateMedia(type)
 - page.emulateMediaFeatures(features)
 - page.emulateMediaType(type)
 - page.emulateTimezone(timezoneId)
 - page.evaluate(pageFunction[, ...args])
 - page.evaluateHandle(pageFunction[, ...args])
 - page.evaluateOnNewDocument(pageFunction[, ...args])
 - page.exposeFunction(name, puppeteerFunction)
 - page.focus(selector)
 - page.frames()
 - page.goBack([options])
 - page.goForward([options])
 - page.goto(url[, options])
 - page.hover(selector)
 - page.isClosed()
 - page.keyboard
 - page.mainFrame()
 - page.metrics()
 - page.mouse
 - page.pdf([options])
 - page.queryObjects(prototypeHandle)
 - page.reload([options])
 - page.screenshot([options])
 - page.select(selector, ...values)
 - page.setBypassCSP(enabled)
 - page.setCacheEnabled([enabled])
 - page.setContent(html[, options])
 - page.setCookie(...cookies)
 - page.setDefaultNavigationTimeout(timeout)
 - page.setDefaultTimeout(timeout)
- class: Accessibility
 - accessibility.snapshot([options])
- class: Keyboard
 - keyboard.down(key[, options])
 - keyboard.press(key[, options])
 - keyboard.sendCharacter(char)
 - keyboard.type(text[, options])
 - keyboard.up(key)
- class: Mouse
 - mouse.click(x, y[, options])
 - mouse.down([options])
 - mouse.move(x, y[, options])
 - mouse.up([options])
- class: Touchscreen
 - touchscreen.tap(x, y)
- class: Tracing
 - tracing.start([options])
 - tracing.stop()
- class: FileChooser
 - fileChooser.accept(filePaths)
 - fileChooser.cancel()
 - fileChooser.isMultiple()
- class: Dialog
 - dialog.accept([promptText])
 - dialog.defaultValue()
 - dialog.dismiss()
 - dialog.message()
 - dialog.type()
- class: ConsoleMessage
 - consoleMessage.args()
 - consoleMessage.location()



PUPPETEER API

```
page.$(selector)
```

```
page.$$ (selector)
```

```
page.$$eval(selector, pageFunction[, ...args])
```

```
page.$eval(selector, pageFunction[, ...args])
```

```
page.$x(expression)
```

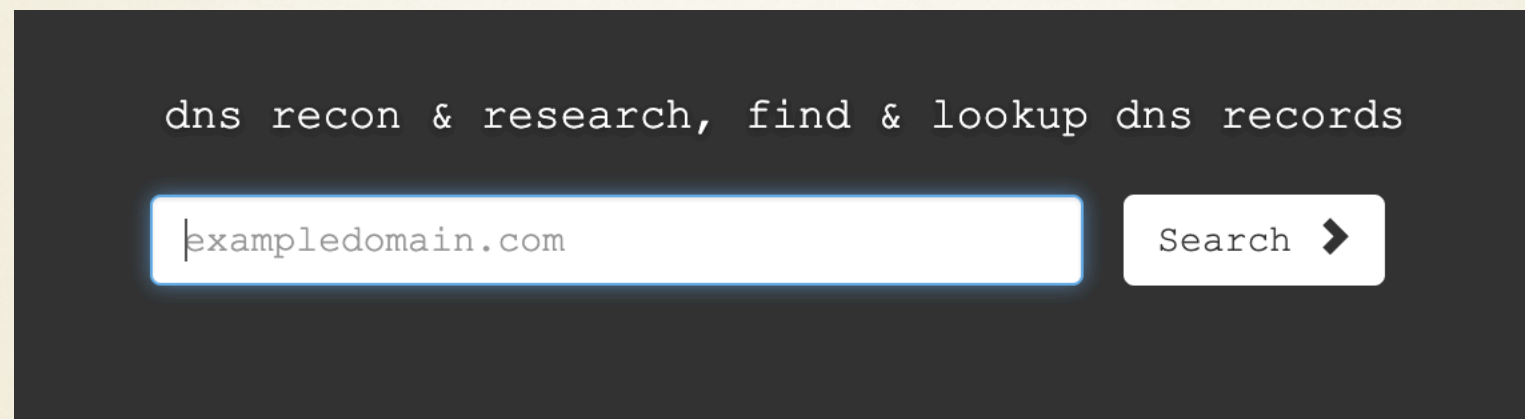
- Selector is the CSS selector
- `$ == document.querySelector`
- `$$ == document.querySelectorAll`
- `$eval/$$eval ==` as above but passing the result to the pageFunction
- `$x == Xpath expression`



PUPPETEER API

- Most of the API calls expect CSS selectors
- Convenient calls to emulate:
 - Mobile devices (see <https://github.com/GoogleChrome/puppeteer/blob/master/lib/DeviceDescriptors.js>)
 - Media features, types
 - Timezone and Geolocation

PUPPETEER FOR RECON



- DnsDumpster.com is a great resource, but no API
- Puppeteer to the rescue to scrape subdomains, and screenshot those that are reachable via HTTP(S)
- Parallelism achieved opening each FQDN in its own tab


```
const puppeteer = require('puppeteer');
const target = "alitalia.it";
const headless = true;
const pageTimeout = 60000;
```

```
(async () => {
  const browser = await puppeteer.launch({
    headless: headless,
  });
  const page = await browser.newPage()

  const navigationPromise = page.waitForNavigation()
```

```
  await page.goto('https://dnsdumpster.com/')
```

```
  await page.setViewport({ width: 1920, height: 900 })
```

```
  await page.waitForSelector('#hideform #regularInput')
  await page.click('#hideform #regularInput')
```

```
  await page.keyboard.type(target, {delay: 100}); // Types slower, like a user
```

```
  await page.waitForSelector('.inner > #hideform > form > #formssubmit > .btn')
  await page.click('.inner > #hideform > form > #formssubmit > .btn')
```

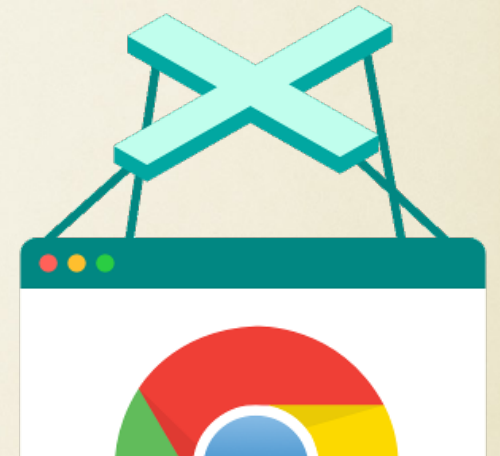
```
  // wait for results
```

```
  console.log("Waiting for results to come up...")
```

```
  await page.waitForSelector('#intro > div:nth-child(1) > div.row > div > h4')
```

```
  await navigationPromise
```

PUPPETEER FOR RECON




```
// gets all tables in results
const tables = await page.$$('#intro > div:nth-child(1) > div.row > div table');
```

```
const txtRecords = await tables[2].$$eval('tr td', tds => tds.map((td) => {
  return td.innerText + "\n";
}));
```

```
let hosts = await tables[3].$$eval('tr td:nth-child(1)', tds => tds.map((td) => {
  let fqdn = td.innerHTML.split("<br>")[0]
  return fqdn;
}));
```

```
hosts = hosts.sort();
hosts.forEach(function(host){
  console.log(host);
});
```

```
const promises=[];
hosts.forEach(function(host){
  // open each host in its own tab
  promises.push(browser.newPage().then(async page => {
    try {
      await page.goto("https://" + host, {
        waitFor: 'networkidle2', timeout: pageTimeout,
        ignoreHTTPSErrors: true
      });
      await page.screenshot({path: `screenshots/https--${host}.png`});
    }catch(e){}
  }));
});
```

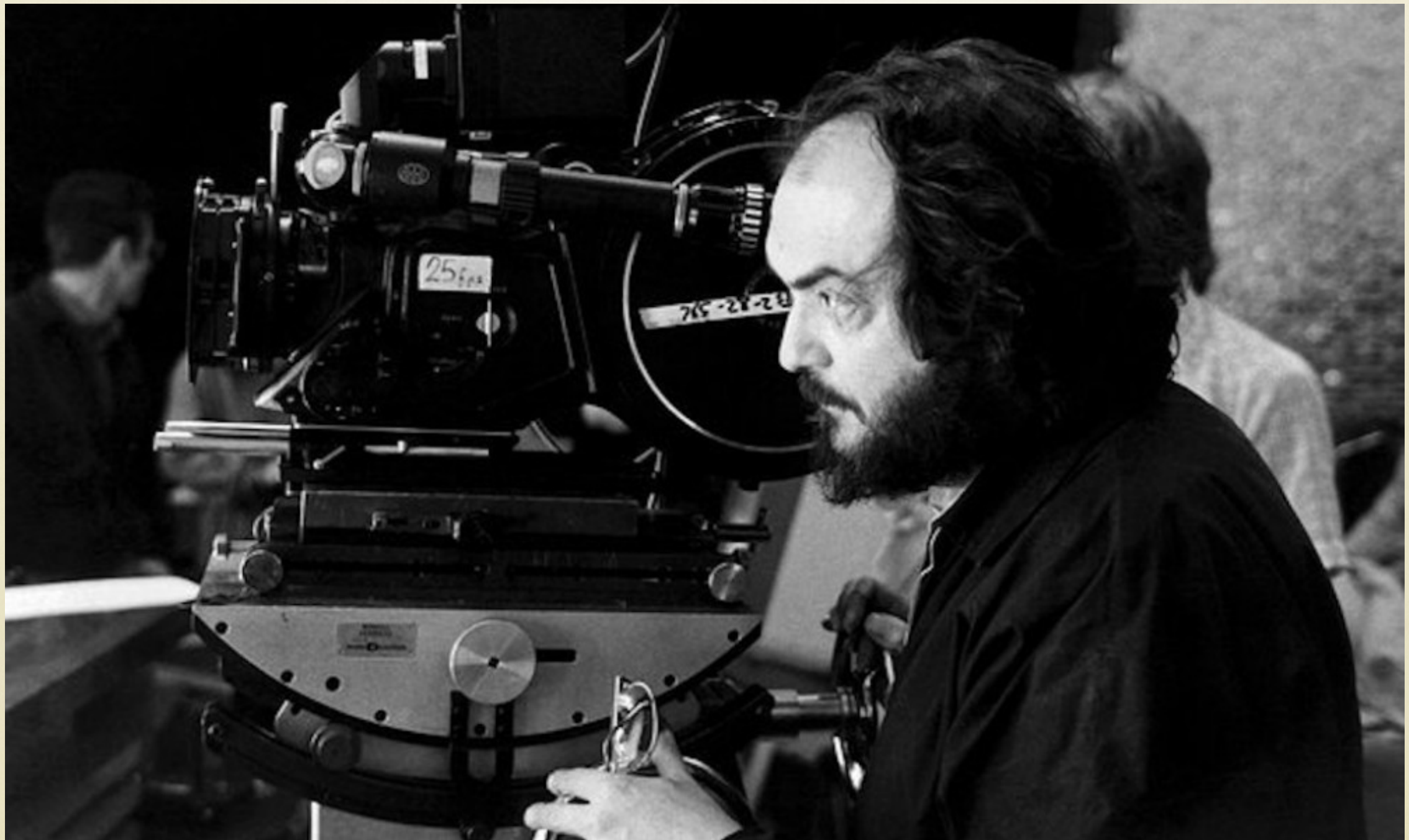
```
// wait for all tabs to close
await Promise.all(promises);
await browser.close()
})()
```

PUPPETEER FOR RECON



Let's see it in action!

PUPPETEER FOR RECON



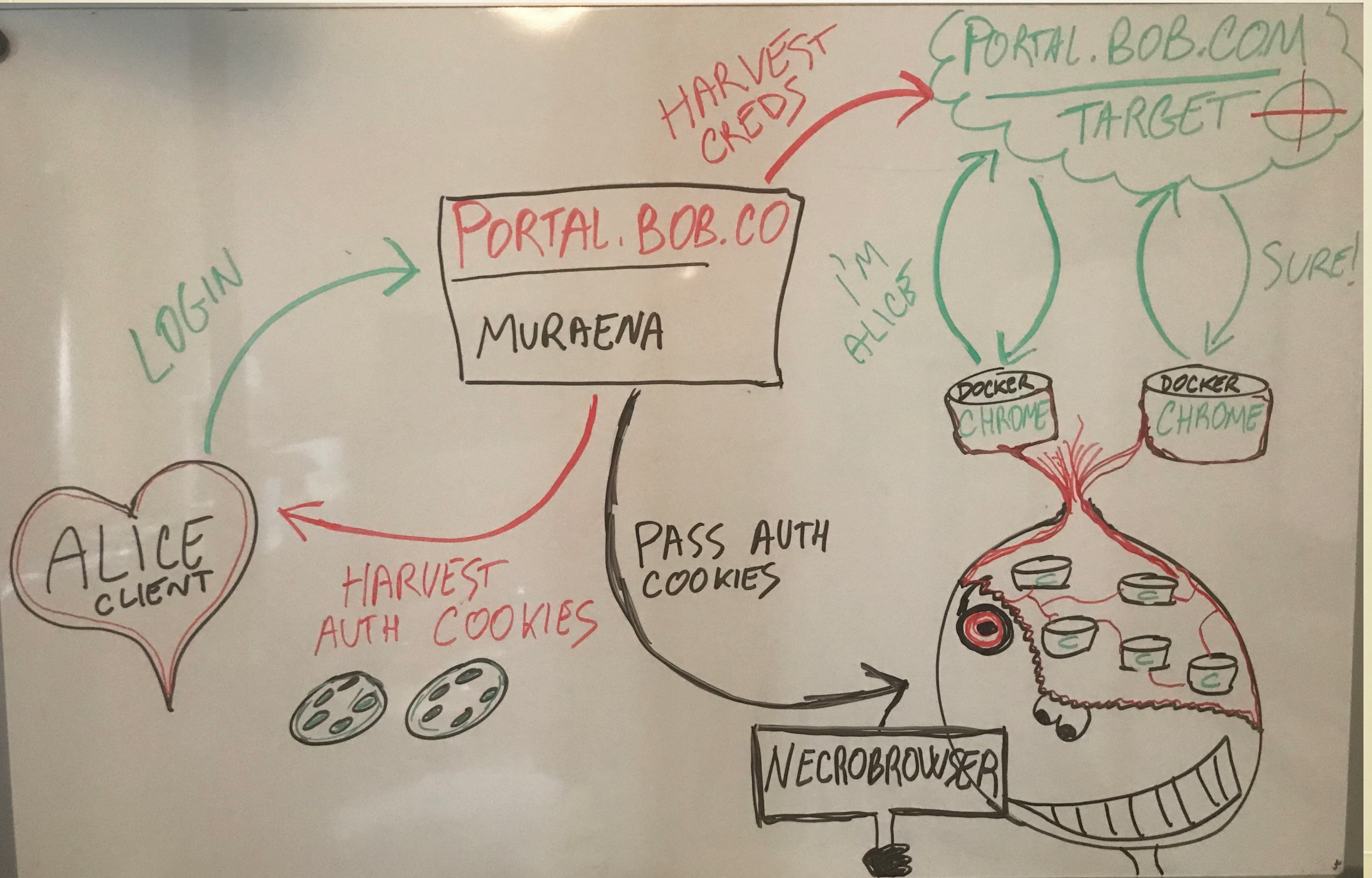
PUPPETEER FOR PHISHING

- Modern Phishing involves a reverse proxy solution (hint: Muraena)
- A smart reverse proxy can then be used to:
 - **intercept** all the traffic
 - **fulfil the 2FA requests flow**
 - pass post-2FA login session cookies to an **instrumented browser that hijacks** the victim's session

PUPPETEER FOR PHISHING

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

PUPPETEER FOR PHISHING



PUPPETEER FOR PHISHING

- **NecroBrowser** is a Go wrapper around **chromedp** (<https://github.com/chromedp/chromedp>)
 - Programmatically drive Chrome via Chrome DevTools Protocol (CDP), like Puppeteer
 - Exposed as a micro service that spawns dedicated Docker containers with Chrome headless
 - Allows to keep alive as many session as your Docker server/cluster can support

PUPPETEER FOR PHISHING

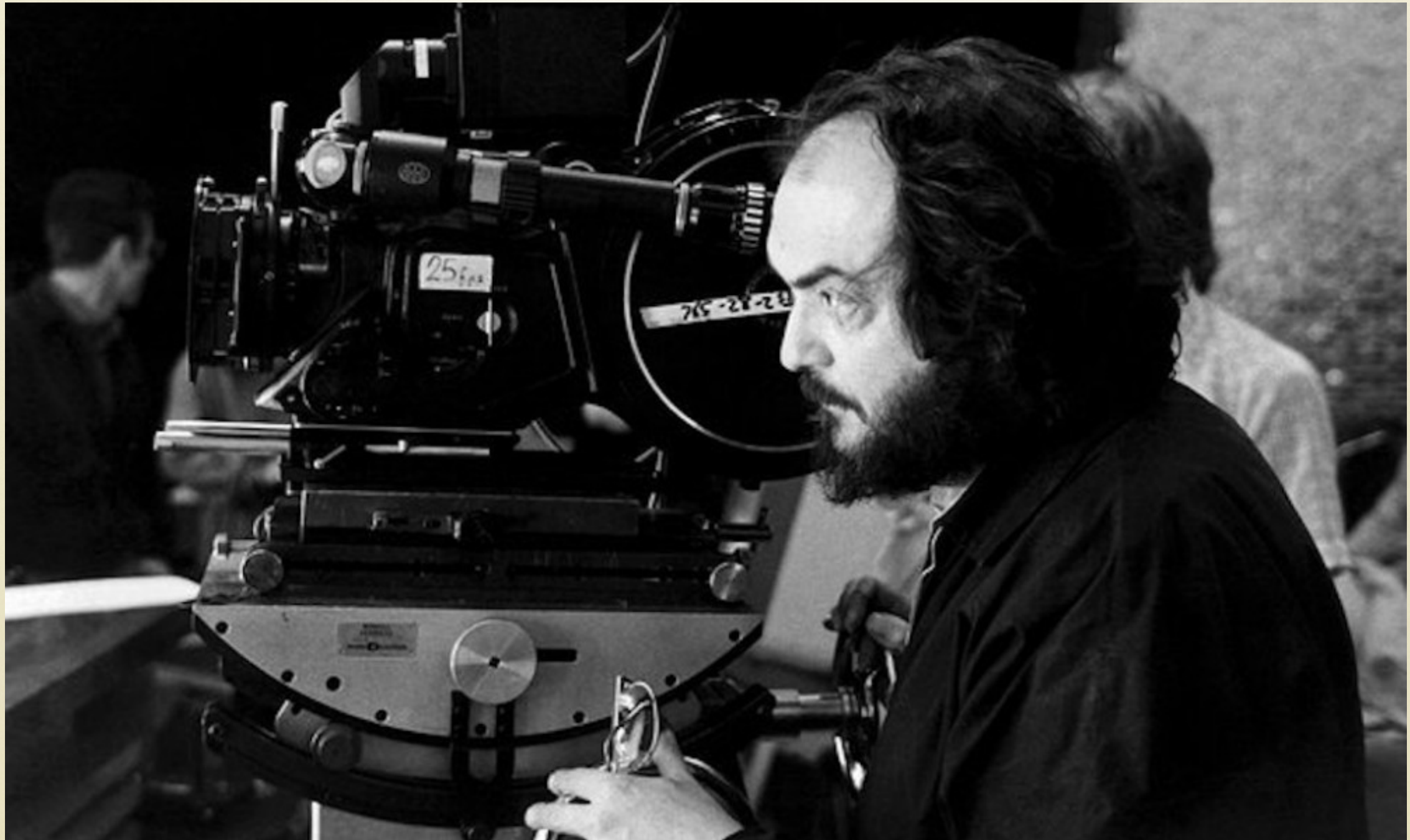
- The problems of **chromedp**:
 - Unreliable on certain complex pages, especially in headless mode (GSuite, Office365)
 - Sometimes events are not triggered, plus other subtle bugs hard to debug
- Not updated/maintained like Puppeteer

PUPPETEER FOR PHISHING

- Plan is to **replace chromedp with Puppeteer** in **NecroBrowser**
 - No need for Docker containers anymore
 - Faster and more reliable
 - ETA Christmas 2019

MURAENA AND NECROBROWSER

Let's see it in action!



PUPPETEER FOR PHISHING



Get Muraena and NecroBrowser here:

<https://github.com/muraenateam>

PUPPETEER FOR WEB SECURITY

- Automatically test to check if an application is still vulnerable to a certain bug
 - XSS: trigger and grep DOM or wait for callback
 - SQLi: trigger and grep errors/status codes/timing
 - RCE: trigger and check
 - SSRF & al.: trigger and check

PUPPETEER FOR WEB SECURITY

- Integrate Puppeteer in Continuous Integration Security Tests
 - Port the attack vectors to Puppeteer scripts
 - Use them in your Functional tests, simulating different devices
 - ...
- Not much websecurity ideas here sorry OWASP!

PUPPETEER DETECTION?



Current status:

Headless detection *failed*.

😎 Evaders are winning!

PUPPETEER DETECTION

- <https://intoli.com/blog/making-chrome-headless-undetectable/>
- <https://intoli.com/blog/not-possible-to-block-chrome-headless/>
- 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

THE FUTURE

- Integrate Puppeteer in NecroBrowser
 - ETA Christmas 2019