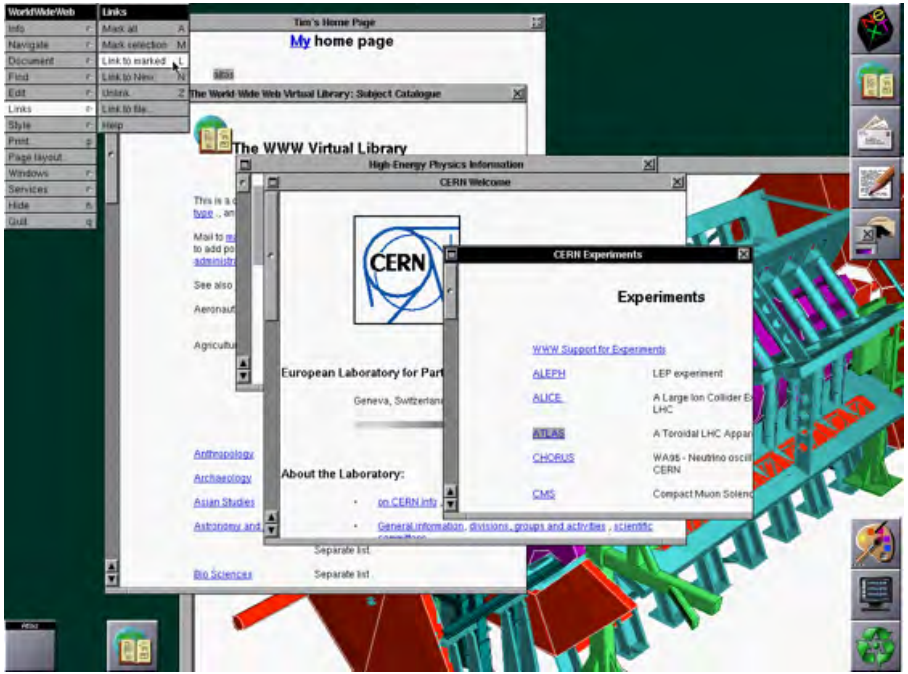
CS 419: Computer Security

Week 13: Web Security

Paul Krzyzanowski

First browsers

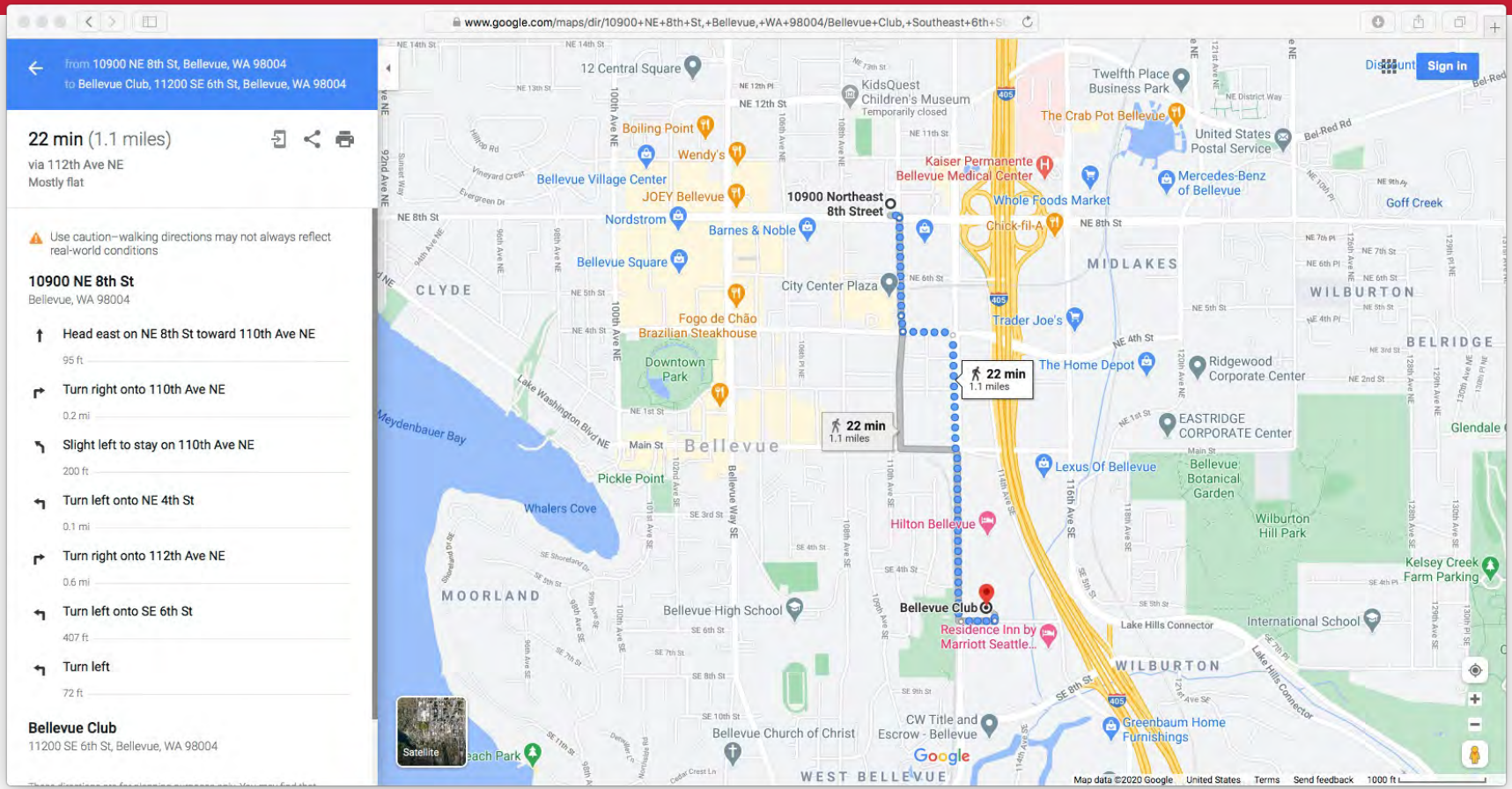
- **Static content**
- **Security attacks were focused on servers**
  - Malformed URLs, buffer overflows, root paths, Unicode attacks



First browsers

- **Static content**
- **Security attacks were focused on servers**
  - Malformed URLs, buffer overflows, root paths, Unicode attacks

Today's Browsers – Complexity Creeps In



Today's Browsers – Complexity Creeps In

- **JavaScript** – allows code execution

Today's Browsers – Complexity Creeps In

- **JavaScript** – allows code execution
- **Document Object Model** (DOM) & **Cascading Style Sheets** (CSS)
  – change appearance of page

Today's Browsers – Complexity Creeps In

- **JavaScript** – allows code execution
- **Document Object Model** (DOM) & **Cascading Style Sheets** (CSS)
  – change appearance of page
- **XMLHttpRequest** (AJAX) – asynchronously fetch content

Today's Browsers – Complexity Creeps In

- **JavaScript** – allows code execution
- **Document Object Model** (DOM) & **Cascading Style Sheets** (CSS)
  – change appearance of page
- **XMLHttpRequest** (AJAX) – asynchronously fetch content
- **WebSockets** –interactive communication between a browser and a server

Today's Browsers – Complexity Creeps In

- **JavaScript** – allows code execution
- **Document Object Model** (DOM) & **Cascading Style Sheets** (CSS)
  – change appearance of page
- **XMLHttpRequest** (AJAX) – asynchronously fetch content
- **WebSockets** –interactive communication between a browser and a server
- **Multimedia support** – <audio>, <video>, <track>

Today's Browsers – Complexity Creeps In

- **JavaScript** – allows code execution
- **Document Object Model** (DOM) & **Cascading Style Sheets** (CSS)
  – change appearance of page
- **XMLHttpRequest** (AJAX) – asynchronously fetch content
- **WebSockets** –interactive communication between a browser and a server
- **Multimedia** support – <audio>, <video>, <track>
- **Geolocation**

WebAssembly (Wasm) & NaCl

- **Google Native Client (NaCl)**
  - Download binary software and run it in your browser
  - Sandboxing and load-time code verification for safety

WebAssembly (Wasm) & NaCl

- **Google Native Client (NaCl)**
  - Download binary software and run it in your browser
  - Sandboxing and load-time code verification for safety
- **WebAssembly**
  - Execution of compiled code by a browser via a processor virtual machine
  - Simple, stack-based virtual machine
  - Harder to detect malware & more opportunities to disguise malware
  - Has been great for cryptominers

Complexity creates a huge threat surface

- More features → more bugs
- Browsers experienced a rapid introduction of features
- Browser vendors don’t necessarily conform to all specs
- Check out `quirksmode.org`

Web Security Model

Page content = multiple sources

www.cnn.com

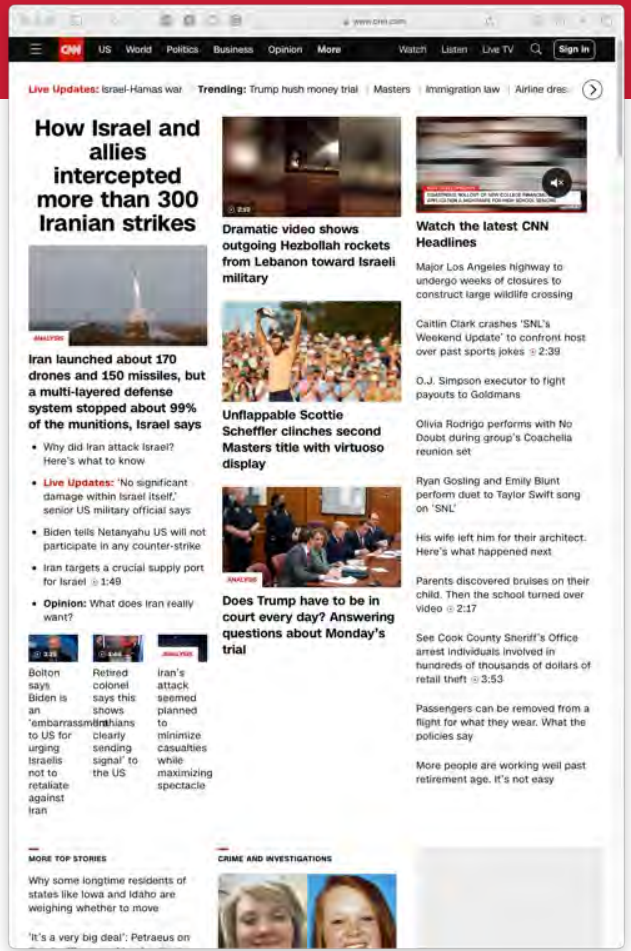
- **Beacons (spy pixels)**
  czion-telemetry.api.cnn.io, bidder.criteo.com, connect-metrics-collector.s-onetag.com, log.outbrainimg.com, logs.browser-intake-datadoghq.com, receive.wmcdp.io, signal-dynamic-pricing-analysis.s-onetag.com, www.google-analytics.com

- **Images**
  www.cnn.com, 1x1.a-mo.net, a..jsrdn..com, ad-delivery.net, ad.doubleclick.net, bea4.v.fwmrm.net, cdn.cnn.com, cdn.cookielaw.org, dt.adsafeprotected.com, events.bouncex.net, i.jsrdn.com, image8.pubmatic.com, media.cnn.com, ping.chartbeat.net, px.moatads.com, saambaa-static.azureedge.net, …

- **Scripts**
  www.cnn.com, a.jsrdn.com, amplify.outbrain.com, api.saambaa.com, assets.bounceexchange.com, btloader.com, c.amazon-adsystem.com, c.jsrdn.com, cadmus.script.ac, cdn.adsafeprotected.com, cdn.boomtrain.com, cdn.cookielaw.com, …

- **Style Sheets**
  db.onlinewebfonts.com, fonts.googleapis.com, registry.api.cnn.io, saambaa.com, turnip.cdn.turner.com …

- **XMLHttpRequest**
  cdn.cookielaw.org, mab.chartbeat.com, logx.optimizely.com, c.amazon-adsystem.com, aax.amazon-adsystem.com, collector.cdp.cnn.com, i.clean.gg, pixel.adsafeprotected.com, atlas.ngtv.io, wmff.warnermediacdn.com, api.btloader.com, …



What should code on a page have access to?

- **Can analytics code access JavaScript variables from a script loaded from jQuery.com on the same page?**

  Scripts are from different places
  … *but the page author selected them so shouldn't that be OK?*

- **Can analytics scripts interact with event handlers?**
- **How about embedded frames?**

Background: Frames and iFrames

- **Browser window may contain embedded frames**
  - Each frame contains *independent* web or media content that may come from *different sources*
  - **Frame** = rigid division as part of frameset (no longer used)
  - **iFrame** = floating inline frame

Background: Frames and iFrames

- **Browser window may contain embedded frames**
  - Each frame contains *independent* web or media content that may come from *different sources*
  - **Frame** = rigid division as part of frameset (no longer used)
  - **iFrame** = floating inline frame
- **Why use them?**
  - Delegate screen area to content from another source
  - Browser provides isolation based on frames
  - Parent can continue to function even if a frame is broken

Web application security policy goals

- **Safe to visit a web site**

Web application security policy goals

- **Safe to visit a web site**

- **Safe to visit two pages at one time**
  - Address bar distinguishes them

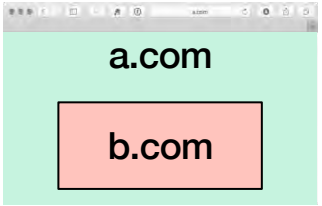
Web application security policy goals

- **Safe to visit a web site**

- **Safe to visit two pages at one time**
  - Address bar distinguishes them

- **iFrame inside a parent frame?**
  - We want to allow safe delegation
  - Each frame = **origin** of the HTML content within it

    Same-origin policy: `a.com` cannot access `b.com`'s content
    `b.com` cannot access `a.com`'s content
    if ***`a.com`*** *and* ***`b.com`*** *have different origins*

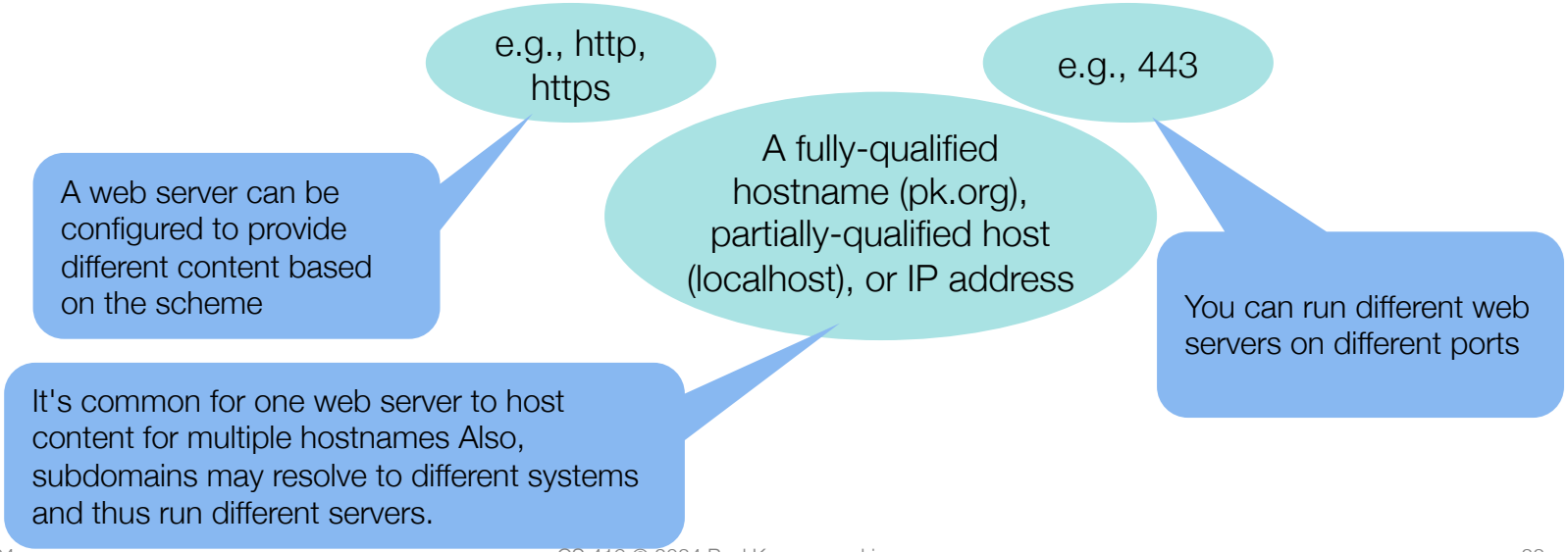


Same-origin policy

Web application security model: same-origin policy

A browser permits scripts in one page to access data in a second page *only if* both pages have the same origin

Origin = { URI scheme, hostname, port number }



Same-origin policy

Web application security model: same-origin policy

A browser permits scripts in one page to access data in a second page *only if* both pages have the same origin

Origin = { URI scheme, hostname, port number }

- **Same origin**
  - http://www.poopybrain.com/419/test.html
  - http://www.poopybrain.com/index.html
- **Different origin from above**
  - https://www.poopybrain.com/index.html – different URI scheme (https)
  - http://www.poopybrain.com:8080/index.html – different port
  - http://poopybrain.com/index.html – different host

How the same-origin policy works

- **Each frame is assigned the origin of its URL**

How the same-origin policy works

- **Each frame is assigned the origin of its URL**
- **Each origin access to its own client-side resources**
  - Cookies: simple way to implement state (sets of *name, value* tuples)
    - Browser sends cookies associated with the origin
  - DOM storage: key-value storage per origin
  - JavaScript namespace: functions & variables
  - DOM tree: JavaScript version of the HTML structure

How the same-origin policy works

- **Each frame is assigned the origin of its URL**
- **Each origin access to its own client-side resources**
  - Cookies: simple way to implement state (sets of *name, value* tuples)
    - Browser sends cookies associated with the origin
  - DOM storage: key-value storage per origin
  - JavaScript namespace: functions & variables
  - DOM tree: JavaScript version of the HTML structure
- **JavaScript code executes with the authority of its frame's origin**
  - If `cnn.com` loads JavaScript from `jQuery.com`, the script runs with the authority of `cnn.com`

How the same-origin policy works

- **Each frame is assigned the origin of its URL**
- **Each origin has access to its own client-side resources**
  - Cookies: simple way to implement state (sets of *name, value* tuples)
    - Browser sends cookies associated with the origin
  - DOM storage: key-value storage per origin
  - JavaScript namespace: functions & variables
  - DOM tree: JavaScript version of the HTML structure
- **JavaScript code executes with the authority of its frame's origin**
  - If `cnn.com` loads JavaScript from `jQuery.com`, the script runs with the authority of `cnn.com`
- **Passive content (CSS files, images) has *no* authority**
  - It doesn't (and shouldn't) contain executable code

Can two different frames communicate?

- **Generally, no – they're isolated if they're not the same origin**
- **But postMessage() allows a scritp to send a message to the Window**
  - A receiver in another frame can pick up an *onmessage* event
- **Both sides have to opt in**

Mixed content: http & https

- **HTTPS page may contain HTTP content:**
  `<script src="http://www.mysite.com/script.js"> </script>`
  - Active network attacker may now hijack the session
  - Content over the network is plain text

- **Safer approach: use HTTPS and don't specify the scheme for content**
  `<script src="//www.mysite.com/script.js"> </script>`
  - Served over the same protocol as the embedding page (frame)

- **Some browsers block mixed content**
  - But this behavior can be disabled

Passive content has no authority

Makes sense … but why does it matter?

Usually no … but …

MIME sniffing attack

- Old versions of IE would examine leading bytes of object to fix wrong `Content-Type` headers
- Suppose a malicious user uploaded an image (passive content) to a web server
  - The content could really be HTML & JavaScript
- IE would allow the download but reclassify the content as HTML with JavaScript

- **Fixes:**
  - Browser gives passive content *no authority*
  - Server can set an `X-Content-Type-Options: nosniff` header to tell the browser not to try to figure out the file's content type

Cross-origin weirdness

- **Images**
  - A frame can load images from anywhere
  - But … same-origin policy does not allow it to inspect the image
  - However, it can infer the size of the rendered image
- **CSS**
  - A frame can embed CSS from any origin but cannot inspect the text in the file
  - But:
    It can discover what the CSS does by creating DOM nodes and seeing how styling changes
- **JavaScript**
  - A frame can fetch JavaScript and execute it … but not inspect it
  - But … you can call `myfunction.toString()` to get the source
  - Or … just download the source via a *curl* command and look at it

Cross-Origin Resource Sharing (CORS)

- **Browsers enforce the same-origin policy**
  - JavaScript can only access content from the same origin
    - Images, CSS, iframes within the page, embedded videos, other scripts, …
    - It cannot make asynchronous requests to other origins (e.g.,via XMLHttpRequest)
- **But a page will often contain content from multiple origins**
  - Images, CSS, scripts, iframes, videos
- **CORS allows a server to define other origins as equivalent**
  - Example, a server at `service.example.com` may respond with

    ```
    Access-Control-Allow-Origin: http://www.example.com
    ```

    Stating that it will treat `www.example.com` as the same origin

CORS Summary

CORS allows a server to define other servers as having the same origin

Those origins can access the requested content as if it was their own origin

Changing binding between names & IP addresses

- **A frame can send http & https requests to hosts that match the origin**
- **The security of *same origin* is tied to the security of DNS**
  - Recall the DNS rebinding attack
    - Register `attacker.com`; get user to visit `attacker.com`
    - Browser generates DNS request for `attacker.com`
      ⇒ DNS response contains a really short TTL
    - After the first access, attacker reconfigures the DNS server
    - Binds `attacker.com` to the alternate IP address
  - JavaScript on a site can fetch a new object from a different address
  - The attacker can access data within the victim's servers and send data back to an attacker's site … all by dynamically changing the name-address mapping

DNS Rebinding attacks

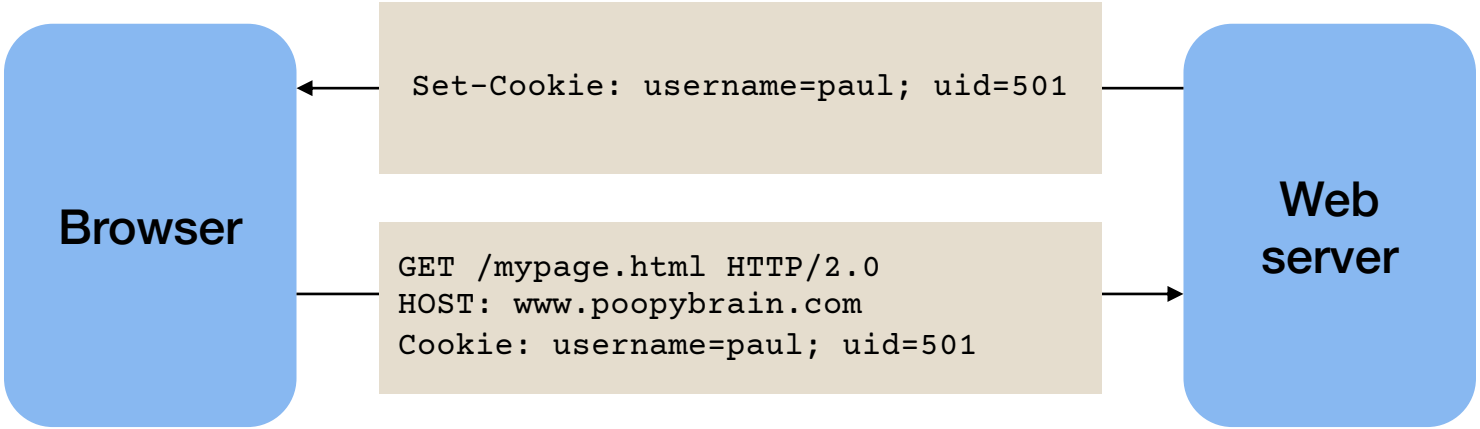
- **Solution – no foolproof solutions**
  - Don’t allow DNS resolutions to return internal addresses
  - Force longer TTL even if the DNS response has a short value

HTTP Cookies

HTTP Cookies

Mechanism created to allow websites to manage browser state

- Cookie = small chunk of data sent by a server to a browser with a page
- Browser sends the cookie back for future requests to the server
- A browser may have an arbitrary # of cookies for a site



When a browser generates an HTTP request it sends all matching cookies

What are cookies used for?

1. Session management (authentication cookies)

- Track a user's activity on a web site
  - Manage a shopping cart even if a user isn't logged in
- Track whether a user is logged into a site
  - Upon successful login, the server sends a session ID cookie
  - This is sent with every future request to the site so it knows you're logged in
- Allows sites like Amazon, eBay, Instagram, Facebook to not prompt you for repeated logins

What are cookies used for?

2. Personalization

- User preferences
- Page rendering options
- Content
- Form data

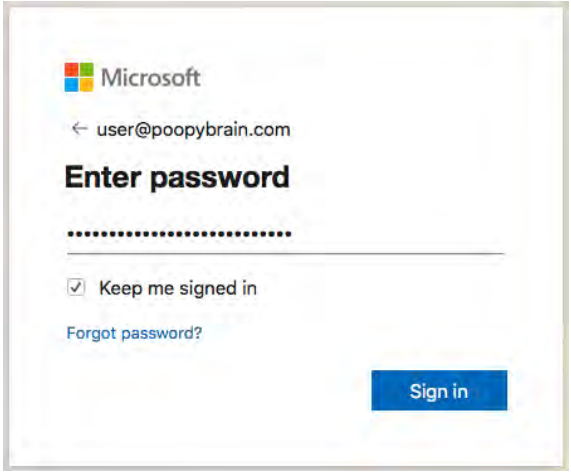
What are cookies used for?

3. Tracking

- Server creates a cookie with a unique ID if browser doesn't provide a cookie
- Cookie will be sent for each page requested from the web site
- Server tracks requested URL & time of request
- Correlate activity with user if (when) user logs in

Types of cookies

- **Session cookies**
  - Only stored in memory
  - Disappear when browser closes
  - No expiration date
- **Persistent cookies**
  - Stored to disk – persists across browser invocations
  - Have an expiration date



Session cookie: `Set-Cookie: name=paul;`

Persistent cookie: `Set-Cookie: name=paul; expires=Mon, 15 Apr 2024 17:30:00 GMT;`

When & where are cookies sent?

Cookies are sent only to the domain & path associated with them

- **domain:** domain to send the cookie with each HTTP request
  - Default: cookie belongs to the domain of the origin
  - Server can specify domain for a cookie
    - Tail component pattern match for domain name

      `domain=poopybrain.com`
    - Will match `www.poopybrain.com`, `419.poopybrain.com`, `public.poopybrain.com`, etc.
- **path:** path in the URL for which to send the cookie
  - Leading substring match. Browser will send the cookie to all paths under the root:

    `Set-Cookie: name=paul; path=/`
    - Browser will send the cookie to `/419`, `/419grades`, `/419-backup`, etc.

      `Set-Cookie: name=paul; path=/419`

Securing Cookies

Cookies are often used to track server sessions

If malicious code can modify the cookie or give it to someone else, an attacker may be able to

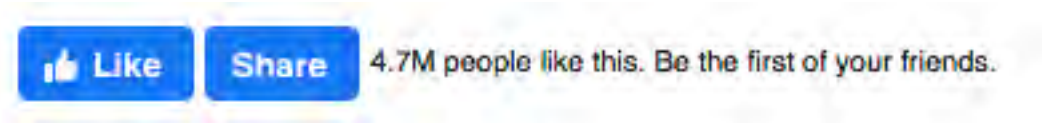
- View your shopping cart
- Get or use your login credentials
- Have your web documents or email get stored into a different account

- **HttpOnly flag: disallows scripts from accessing the cookie**
- **Secure flag: send the cookie only if there is an https session**

```
Set-Cookie: username=paul; path=/; HttpOnly; Secure
```

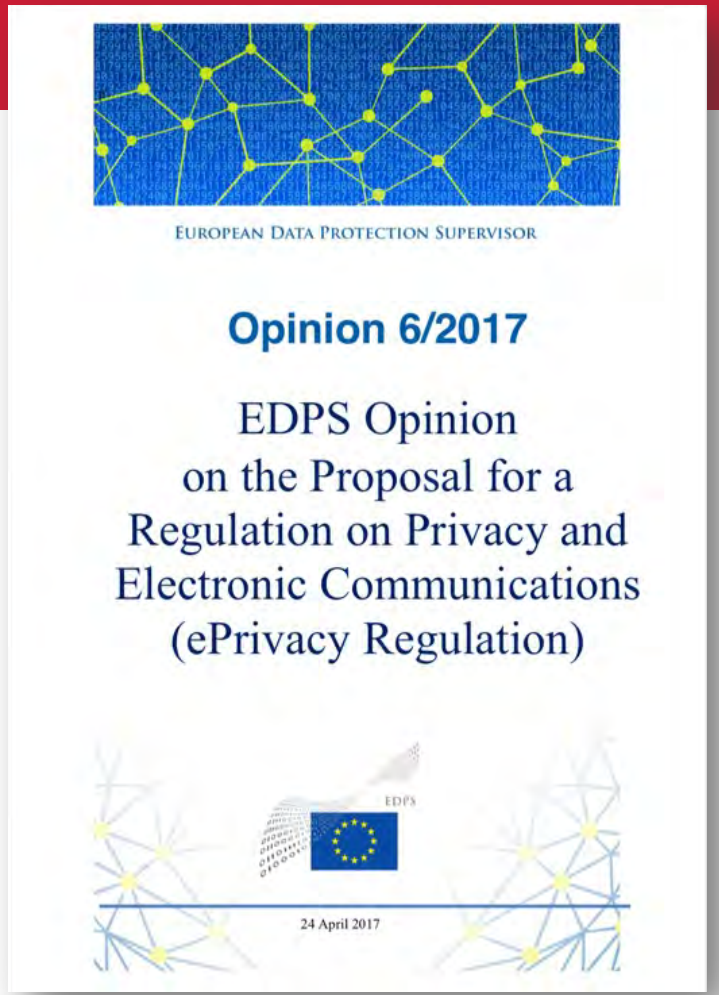
Third-party cookies: tracking cookies

- **First-party cookies: sent by the domain of the page you loaded**
- **Third-party cookies:**
  - sent by content in iFrames, media, and scripts (ads, Facebook, Google)
- **Page = collection of first-party and third-party content**
  - Third-party servers get their cookie whenever you visit a page they have a presence on
  - Record parent page, time of visit, user ID (if you have a cookie for one)



Cookies and privacy

- **Cookies are essential but their use for tracking can also be invasive**
- **EU ePrivacy Directive**
  - Receive consent from users
  - Provide info about each cookie
  - Store the user's consent
  - Provide the service without consent
  - Allow users to change their minds



Web-based Attacks

Malicious JavaScript & drive-by downloads

- **Most web pages load JavaScript files**
  - Malicious pages or iFrames may load malicious JavaScript
  - Visiting a malicious page that loads scripts is called a ***drive-by download***
- **What's the harm?**
  - The script redirects you to another site & downloads an exploit kit
  - Exploit kit from the site probes, OS, browser, and other software to find vulnerabilities
  - Exploit kit downloads malware payload
- **Other actions**
  - Present ads, generate ad click-through, generate *likes* for social media content, mine cryptocurrency.

Cross-Site Request Forgery (CSRF)

A browser sends cookies for a site along with each server request

- **If an attacker gets a user to access a site**
  … the user's cookies will be sent with that request
- **If the cookies contain the user's identity or session state**
  - The attacker can create actions on behalf of the user

https://mybank.com?action=transfer&to=*attacker_account*&amount=1000.00

- **Plant the link in forums, email, ads, …**

Important notice!

The user sees: Important notice!

Cross-Site Request Forgery (CSRF) – HTTP POST

Embed a script on a malicious site (could be malicious content in a frame)

- The victim visits the page, which runs a script that sends a request to post a transfer request to the bank
- The HTTP POST requesst to the bank will send the user's session cookie if they were previously loggged in

```
<head>
    <script>
      document.addEventListener("DOMContentLoaded", function () {
        document.getElementById("steal_money").submit(); });
    </script>
</head>
<body>
    <form
      id=" steal_money"
      method="POST"
      action="https://mybank.com/transfer.php"
    >
      <input type="hidden" name="attacker_account" value="123456" />
      <input type="hidden" name="amount" value="10000" />
    </form>
</body>
```

Some past CSRF attacks

- Create accounts on behalf of user
- Transfer funds out of user's accounts

- Add videos to *Favorites*
- *A*dd attacker to a user's *Friends* or *Family* list
- Send messages as user, share video with user's contacts
- Subscribe a user to a channel

- Find email address of arbitrary user

- Add a movie to a user's queue

- Take over any Facebook user account

CSRF Defenses

- **For the user**
  - Log off sites when you're done with them
    - "Logging off" clears session cookies
  - Don't allow browsers to store authentication cookies for sites
- **On the server**
  - The server can create a unique random token for every session
    - Token sent via hidden fields or headers
    - Validated with each page request from the user
  - Add a token that's an *HMAC(request, timestamp)*
  - Identify the origin of the request (*Origin* or *Referer* header)
  - Set cookies with a "SameSite" flag – that will send them only if the request is coming from the same origin

Screen sharing attack

- **HTML5 added a screen sharing API**
- **Normally: no cross-origin communication from client to server**
- **This is violated with the screen sharing API**
  - If a frame is granted permission to take a screenshot, it can get a screenshot of the entire display (monitor, windows, browser)
  - Can also get screenshots within the user's browser without consent
- **User might not be aware of the scope of screen sharing**

http://dl.acm.org/citation.cfm?id=2650789

http://mews.sv.cmu.edu/papers/oakland-14.pdf

Unsanitized Input Attacks

Input sanitization

Remember command injection attacks?

- **Any user input must be parsed carefully**

```
<script> var name = "untrusted_data"; </script>
```

- **Attacker can set `untrusted_data` to something like:**

```
hi"; </script> <h1>Hey!</h1> <script> malicious code …
```

```
<script> var name = "hi"; </script> <h1>Hey!</h1> <script> malicious code … "; </script>
```

- **Sanitization should be used with any user input that may be part of**
  - HTML
  - URL
  - JavaScript
  - CSS

SQL Injection

- Many web sites use a back-end database
- Queries may be constructed with user input

```
username = getRequestString("uname");
pwd = getRequestString("passwd");

query = 'select * from Users where name = "'
                        + username + '" and pwd = "' + pass + '"'
```

User Name:

ramesh

Password:

letmein


```
select * from Users where
  name = "ramesh" and pwd = "letmein"
```

SQL Injection

- Many web sites use a back-end database
- Queries may be constructed with user input

```
username = getRequestString("uname");
pwd = getRequestString("passwd");

query = 'select * from Users where name = '''
                        + username + ''' and pwd = ''' + pass + '''';
```

User Name:

" or ""="

Password:

" or ""="


```
select * from Users where
  name = "" or ""="" and pwd = "" or ""=""
```

will return all rows from the Users table

OS command injection

- **Servers that use web form data in an command may be vulnerable to OS command injection – due to unsanitized inputs**

```
<?php
print("Enter name of the file to delete");
print("<p>");
$file=$_GET['filename'];
system("rm $file");
?>
```

See https://owasp.org/www-community/attacks/Command_Injection

Cross-Site Scripting (XSS)

Code injection attack

- **Allows attacker to execute JavaScript in a user's browser**
- **Exploit vulnerability in a website the victim visits**
  - **Possible if the website includes user input in its pages**
  - Example: user content in forums (feedback, postings)

- **Main types of attack:**
  - Stored XSS
  - Reflected XSS

Stored (Persistent) XSS

- **Website stores user input and serves it back to other users at a later stage**
- **Victims do not have to click on a malicious link to run the payload**
- **Example: forum comments & postings**

Item Description

```
<h1> Official Radioactive Man collectible doll.</h1>
<p> <strong>Brand new in box</strong>. </p>
<p style="color:red">Comment</p>

<script function loadurl(url){
    window.creator.location=url; . . . .
} </script>
```

Reflected XSS

- **Malicious code is not stored on the server**
  - It is returned as part of the HTTP response
  - Attack string is part of the link
    - HTTP query parameters used without sanitization and contain code
  - Distributed as links on spam email or web sites
    - Links look legitimate because the domain name is a valid, trusted server
  - Attacks may take advantage of existing cookies that will authenticate a user
- **Web application passes unvalidated input back to the client**

  The script in the link is returned in its original form inside the page & executed

```
www.mysite.com/login.asp?user=<script>malicious_code</script>
```

What's the harm?

- Access a user's cookies related to that website
- Hijack a session (using session authentication cookie)
- Create arbitrary HTTP requests with arbitrary content via XMLHtttpRequest
- Make arbitrary modifications to the HTML document by modifying the DOM
- Install keyloggers
- Download malware – or run JavaScript ransomware
- Try phishing by manipulating the DOM and adding a fake login page or redirecting

XSS Defenses

- **Key defense is sanitizing ALL user input**
  - E.g., Django templates: `<b> hello, {{name}} </b>`
  - Use a less-expressive markup language for user input (e.g., markdown)
- **One of the problems in preventing XSS is character encoding**
  - Filters might check for "`<script>`" but not "`%3cscript%3e`"
- **Privilege separation**
  - Use a different domain for untrusted content
    - E.g., google would use `googleusercontent.com` for serving user-supplied content
    - Limits damage to the main domain (e.g., `google.com`)
- **Content Security Policy (CSP)**
  - Designed to prevent XSS & clickjacking
  - Allows website owners to identify approved origins & types of content the user can access

Deception: Typographic Attacks

Homograph attacks

Unicode confusion

Unicode represents virtually all the worlds glyphs

Some symbols look the same (or similar) but have different values

Potential for deception

They're totally different to software but look the same to humans

/ = solidus (slash) = U+002F

⁄ = fraction slash = U+2044

∕ = division slash = U+2215

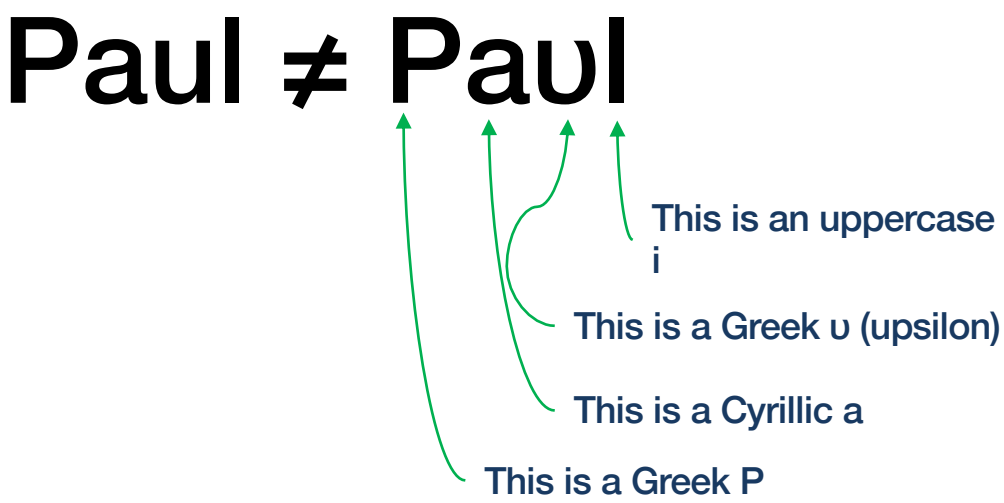
̷ = combining short solidus overlay = U+0337

̸ = combining long solidus overlay = U+0338

／ = fullwidth solidus = U+FF0F

Yuck!

Paul ≠ Paul


Paul ≠ Paul
This is an uppercase i
This is a Greek υ (upsilon)
This is a Cyrillic a
This is a Greek P

Homograph (Homoglyph) Attacks

- **Some characters may look alike:**
  - 1 (one), l (L), I (i)
  - 0 (zero), O
- **Homograph attack = deception**
  - paypal.com vs. paypal.com (`I` instead of L)

google.com or **googie.com**

- **It got worse with internationalized domain names (IDN)**
  - wikipedia.org
    - Cyrillic a (U+0430), e (U+435), p (U+0440)
    - Belarusian-Ukrainian i (U+0456)
  - Paypal
    - Cyrillic P, a, y, p, a; ASCII l

instagram.com or **instɑgrɑм.com**

Check out the Homoglyph Attack Generator at https://www.irongeek.com/homoglyph-attack-generator.php

https://en.wikipedia.org/wiki/IDN_homograph_attack

URL Hijacking

URL Hijacking – Typosquatting

- **Misspelled domain names**
  - Confuse people into thinking the domain is something else
  - Hope that users make a typo when they type a URL
- **Where do they take you?**
  - Non-malicious content (usually)
    - Parked page (non-configured site) – accounts for most typosquats
    - Advertising
    - Brand-damaging content
    - Redirect to a different site (e.g., competitor)
    - The legitimate site
  - Malicious content (credential stealing, malware)
- **Domain names can also be used in email-based phishing campaigns**

www.googlec.om

Combosquatting
`chase-bank.com`

Revisiting Typosquatting And The 2020 US Presidential Election

digital shadows_

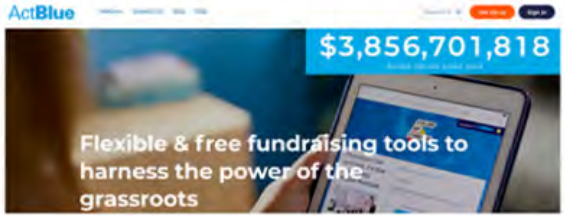
Kacey C. • September 2, 2020

In October 2019, Digital Shadows' Photon Research Team embarked on an adventure involving election typosquats that could potentially affect the presidential election and its candidates. If you haven't read our original report, I'll fill you in on a brief recap:

We detected over 550 typosquats for the 34 candidate- and election-related domains from open-source research. Not every single domain was interesting; most of the time, the typosquat was parked and not hosting content. Still, there were some worthwhile areas to dig into deeper: Misconfigured or illegitimate sites, non-malicious sites, and website redirects.

https://www.digitalshadows.com/blog-and-research/revisiting-typosquatting-and-the-2020-us-presidential-election/

Examples



winrde.com

tulsi2020.co

elizibethwarren.com

donaldtrump.digital

https://www.digitalshadows.com/blog-and-research/typosquatting-and-the-2020-u-s-presidential-election/

https://www.cyberscoop.com/dhs-bulletin-typosquatting-2020-election-officials/

April, 2024

Transitioning to the .gov Domain: Why It Matters

Foreign adversaries and cyber threat actors have demonstrated the intent to target U.S. elections and election infrastructure in previous election cycles, and we expect the threat these actors pose to future elections will continue.[1] These actors may use a variety of tactics, including engaging in cyber threat activity targeting election office websites and email accounts, as well as conducting influence operations that seek to impersonate election offices or election officials.

The Cybersecurity and Infrastructure Security Agency (CISA) and Federal Bureau of Investigation (FBI) recommend all election offices adopt a .gov domain to help election offices and other state, local, tribal, and territorial (SLTT) government entities mitigate impersonation and cybersecurity risks. Similar to .com, .org, or .us domains, organizations use the .gov domain for online services, like websites or email. Unlike other domains, .gov is only available to official U.S.-based government organizations and publicly controlled entities. This means that users visiting a .gov website or receiving an email from a .gov email address can be more confident that the content is genuine government information. Similarly, use of the .gov domain can help the public better recognize official government sites and emails while avoiding phishing attempts and websites that impersonate government officials.

https://www.cisa.gov/sites/default/files/2024-04/CISA-FBI-The_.gov_Domain-Helping_Mitigate_Election_Office_Cybersecurity_and_Impersonation_Risks_v2_508c.pdf

Examples

- **160 domains containing `appleid` registered between October 2019-April 2020**
  - e.g., `www-appleid[.]com`
  - None owned by apple
  - 28% had malicious content
- **Some famous examples:**
  - MikeRoweSoft.com → *targeted Microsoft (sort of)*
  - hotmail.com variations → *targeted hotmail.com*
  - Fallwell.com → *targeted Jerry Falwell (falwell.com)*
  - PETA.org → *targeted PETA*

PayPal-related domains registered in June 2020

```
paypalticket91661[.]info
paypal-team[.]space
paypal-service[.]website
paypalticket91644[.]info
mypaypal[.]online
paypal-service[.]site
team-paypal[.]space
paypalticket91640[.]info
paypal-service[.]space
paypal-updateconfirmationsaccounts[.]com
paypal-updateconfirmationsaccount[.]com
paypal-support[.]space
team-paypal[.]website
paypalticket91645[.]info
paypalticket91642[.]info
paypalticket91664[.]info
paypal-updateconfirmationaccount[.]com
```

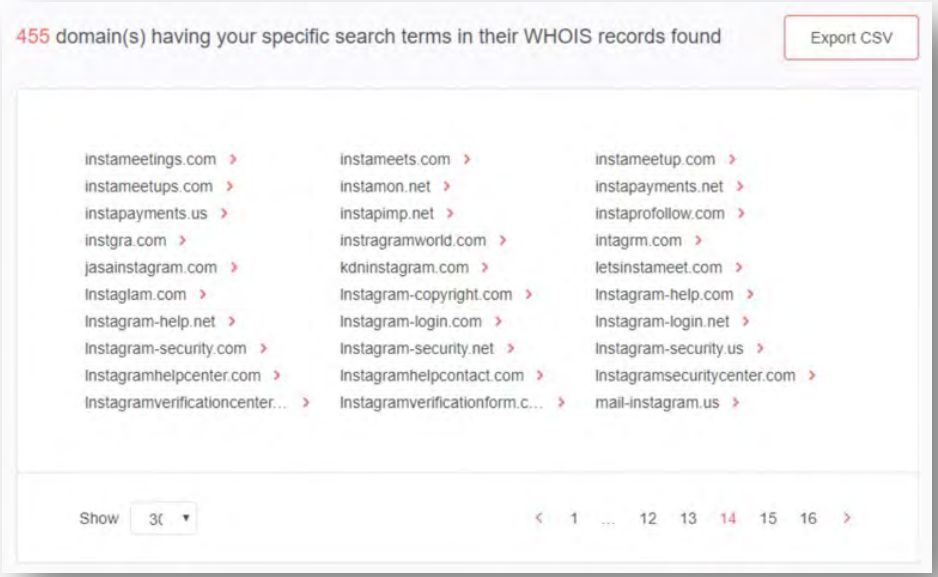
Typosquatting data feed: https://typosquatting.whoisxmlapi.com

https://www.darkreading.com/threat-intelligence/typosquatting-wave-shows-no-signs-of-abating
http://www.circleid.com/posts/20200609-typosquatting-domains-every-appleid-owner-should-avoid/
http://www.circleid.com/posts/20200618-60-paypal-potential-typosquatting-domains-detected-in-june/

Protection against typosquatting

- **It's a race – legitimate domain owners race to get domain names before adversaries**
- **Example:**

  A WHOIS search shows
  455 instagram-related names that belong to Facebook
  (who owns Instagram)



http://www.circleid.com/posts/20200605-typosquatting-protection-a-look-into-instagram-themed-domain-names/

But you can't register every possible domain!

- **For example, IBM detected these registrations:**
  - `copyright-lnstagram[.]ml`
  - `instagram-verifybadge-support[.]ml`
  - `lnstagram-copyright-help-a3623vas336-va6f63a6ogsa824[.]ml`
- **The letter before "nstagram" in the first domain is an L, not an I**

Legal remedies

- 1999 U.S. Anticybersquatting Consumer Protection Act (ACPA)
  - Prove good-faith use of URL
  - Have a domain that is not similar to existing trademarks or brands
- World Intellectual Property Organization (WIPO)
  - Petition the court that a domain is confusingly similar to yours
  - Holder had no rights to your brand
  - Site is used in bad faith

http://www.circleid.com/posts/20200605-typosquatting-protection-a-look-into-instagram-themed-domain-names/

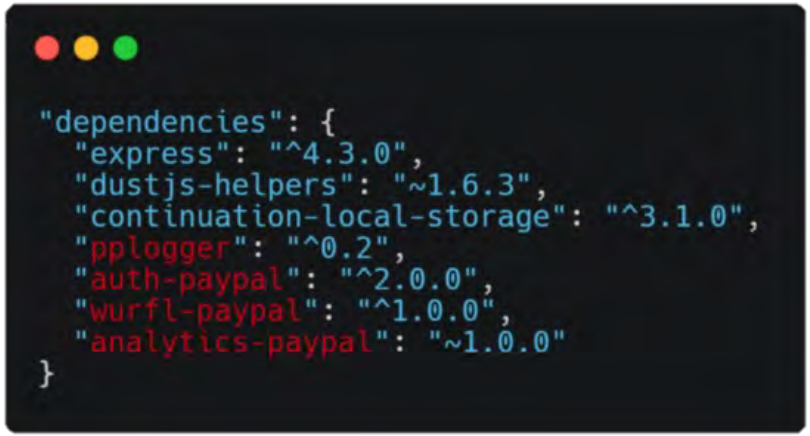
Typosquatting: Not just a URL problem!

- **Package managers (PyPi, npm, ...) often contact public repositories of source**
  - Anyone can add new packages
- **50% of packages are installed with admin privileges**
- **Attacker can create fake packages with similar names to legitimate ones and hope victims make grammatical mistakes or typos when installing**

See https://incolumitas.com/2016/06/08/typosquatting-package-managers/

Typosquatting: Dependency Confusion

- **Example – PayPal Node.js source code on GitHub**
  - Meant for internal PayPal use (other internal package names published on Internet forums)
  - Package (package.json) contained a mix of public & private dependencies
    - Public ones are probably hosted from npm
    - Private ones are hosted internally



- **An attacker (white hat – with permission!) uploaded malicious Node packages to npm with those private component names**
- **Software ended up loading these components instead of the private ones**
- **Apple, Shopify, Yelp, & Tesla were a few of the companies that were exposed!**

See https://medium.com/@alex.birsan/dependency-confusion-4a5d60fec610

PyPI Halts Sign-Ups Amid Surge of Malicious Package Uploads Targeting Developers

The Hacker News

March 29, 2024

The maintainers of the Python Package Index (PyPI) repository briefly suspended new user sign-ups following an influx of malicious projects uploaded as part of a typosquatting campaign.

PyPI said "new project creation and new user registration" was temporarily halted to mitigate what it said was a "malware upload campaign." The incident was resolved 10 hours later, on March 28, 2024, at 12:56 p.m. UTC.

Software supply chain security firm Checkmarx said the unidentified threat actors behind flooding the repository targeted developers with typosquatted versions of popular packages.

"This is a multi-stage attack and the malicious payload aimed to steal crypto wallets, sensitive data from browsers (cookies, extensions data, etc.), and various credentials," researchers Yehuda Gelb, Jossef Harush Kadouri, and Tzachi Zornstain said. "In addition, the malicious payload employed a persistence mechanism to survive reboots."

https://thehackernews.com/2024/03/pypi-halts-sign-ups-amid-surge-of.html

Image-based attacks & tracking

Clickjacking: User Interface Redress Attack

- **Trick users into clicking on content**
- **User sees this**

- **But does not realize there's an invisible frame *over* the image**
- **Clicking on the frame could generate a Facebook *like***
  … or download malware … or change security settings for a plugin
- **Defense**
  - JavaScript in the legitimate code to check that it's the top layer
    `window.self == window.top`
  - Set `X-Frame-Options` to not allow frames from other domains

HTML image tags

```
<img src="http://pk.org/images/balloons.jpg" height="300" width="400"/>
```

- Images are static content with no authority
- Any problems with images?

HTML image tags & tracking pixels

```
<img src="http://pk.org/images/balloons.jpg?extra_information" height="300" width="400"/>
```

- **URL may pass arguments**
  - Communicate with other sites
- **Hide the image: spy pixel (tracker pixel)**
  `<img src="…" height="1" width="1"/>`

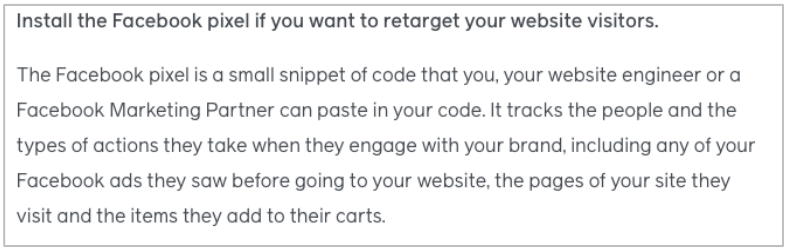
Common way for a sender to force HTML-formatted email to provide read notifications

- **The request for the image will send cookies for that domain**
  – and the server response can set cookies

Almost 25% of mail messages contain a tracking link
Of popular sending domains, about 50% perform tracking

Ad retargeting

```
<img height="1" width="1" style="display:none”
src="https://www.facebook.com/tr?id=1563912751991l8&amp;ev=PageView&amp;noscript=1"/>
```



- **Origin = www.facebook.com**
- **Accessing the web page with this pixel will**
  - Contact Facebook to load the image
  - Send Facebook cookies from your browser to Facebook
  - Enable Facebook to record the fact that you visited this page

Google ad for GIMP.org served info-stealing malware via lookalike site

BLEEPINGCOMPUTER

Ax Sharma • Nov 1 2022

Google ads 'display URL' vs. 'landing URL'

Google lets publishers create ads with two different URLs: a display URL to be shown in the ad, and a landing URL where the user will actually be taken to.

The two need not be the same, but there are strict policies around what is permitted when it comes to display URLs, and these need to use the same domain as the landing URL.

"Advertisers use a landing page URL to send people to a specific area of their website," explains Google.

"Your ads' URLs should give customers a clear idea of what page they'll arrive at when they click on an ad. For this reason, Google's policy is that both display and landing page URLs should be within the same website. This means that the display URL in your ad needs to match the domain that visitors land on when they click on your ad."

https://www.bleepingcomputer.com/news/security/google-ad-for-gimporg-served-info-stealing-malware-via-lookalike-site/

Deception via image tags

Social engineering: add logos to fool a user

- Impersonate site
- Impersonate credentials

Can You Trust the Browser Status Bar?

Mouseover on a link shows link target

https://www.paypal.com/signin/

Trivial to spoof with JavaScript

```
<a href="http://www.paypal.com/signin"
       onclick="this.href='http://www.evil.com/';">
       PayPal</a>
```

The situation is not good

- **HTML, JavaScript, and CSS continue to evolve**
- **All have become incredibly complex**
- **Web apps themselves can be incredibly complex, hence buggy**
- **Web browsers are forgiving**
  - You don't see errors
  - They try to correct syntax problems and guess what the author meant
  - Usually, *something* gets rendered

The End