CS 419: Computer Security

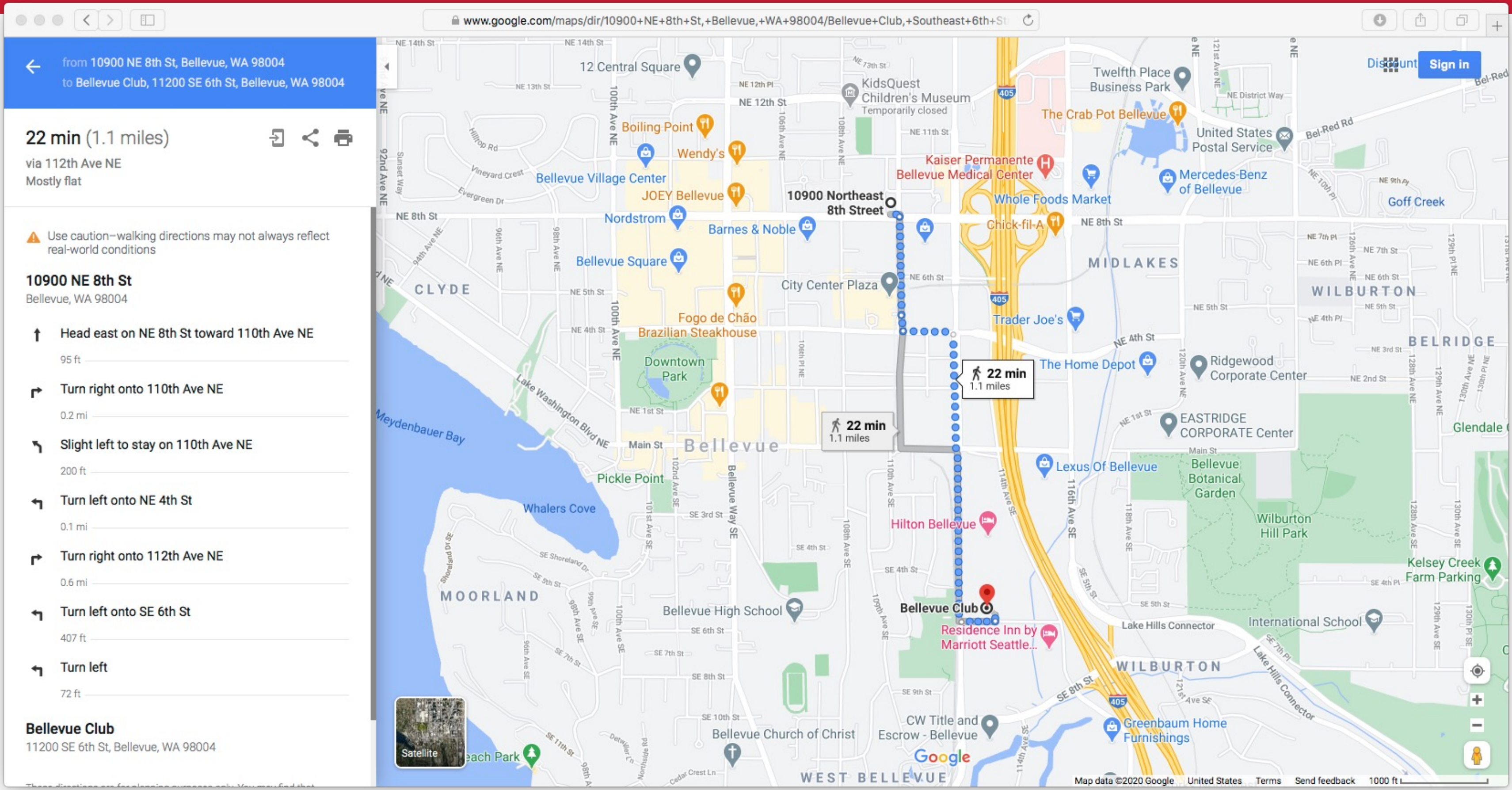
Week 12: Web Security

Paul Krzyzanowski

First browsers

- **Static content**
- **Security attacks were focused on servers**
  - Malformed URLs, buffer overflows, root paths, Unicode attacks
  - Code injection, command execution, data exfiltration/modification

Today's Browsers – Complexity Creeps In



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)

- **WebAssembly**
  - Execution of compiled code by a browser via a processor virtual machine
  - Simple, stack-based virtual machine
- **Risks?**
  - Runs a sandbox, providing restricted access to the environment
  - Payload is binary compiled code
  - Harder to detect malware & more opportunities to disguise malware
  - Great for cryptomining because of increased performance

Complexity creates a huge threat surface

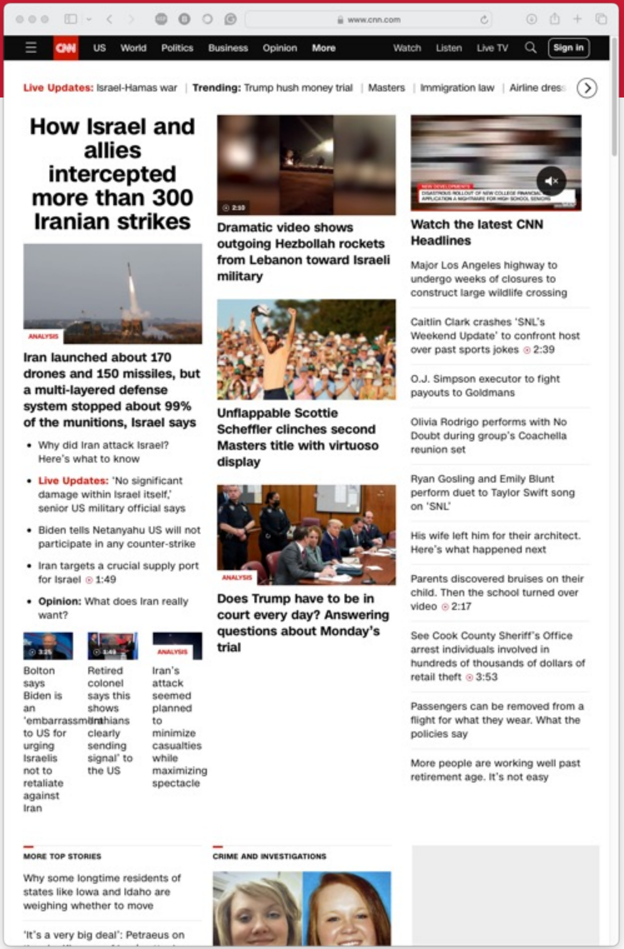
- More features → more bugs
- Browsers experienced a rapid introduction of features
- Browser vendors don’t necessarily conform to all specs
- Check out `quirksmode.org`

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: iFrames

- **A browser window may contain embedded frames**

  *An **iFrame** (inline frame) is an HTML element that embeds another HTML document within the current webpage, enabling the display of external content such as videos or other web pages within a section of the parent page*

  – Each frame contains *independent* web or media content that may come from *different sources*

- **Why use them?**
  – Delegate screen area on the page to content from another source
  – Browser provides isolation based on iFrames
  – The parent can continue to function even if the page in an iFrame is broken

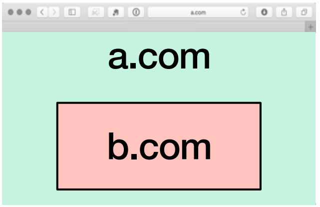
Web application security policy goals

- **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 }

e.g., http, https

A fully-qualified hostname (pk.org), partially-qualified host (localhost), or IP address

e.g., 443

A web server can be configured to provide different content based on the scheme

It's common for one web server to host content for multiple hostnames Also, subdomains may resolve to different systems and thus run different servers.

You can run different web servers on different ports

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
  - `http://73.112.71.100/index.html` – different hostname

How the same-origin policy works

- **Each frame is assigned the <u>origin</u> 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 (permissions) 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 *<u>no</u>* 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 script to send a message to the window**
  – A receiver in another frame can pick up an *`onmessage`* event
- **Both sides have to opt in (i.e., they expect to communicate)**

Frame A (`trusted-domain.com`)

```
window.parent.postMessage("message",
"https://target-domain.com");
```

→

Frame B (`target-domain.com`)

```
window.addEventListener("message", (event) =>
{
  if (event.origin ===
                   https://trusted-domain.com") {
    console.log("Received:", event.data);
  }
});
```

Mixed content: http & https

- **An HTTPS page may download HTTP content:**

```
<script src="http://www.mysite.com/script.js"> </script>
```

  – Http content over the network is plain text
  – An active network attacker may hijack the session

- **Safer approach: use HTTPS and don't specify the scheme for content**

```
<script src="//www.mysite.com/script.js"> </script>
```

  – Will be 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

- Older versions of Internet Explorer, Chrome, and Firefox would examine the leading bytes of the 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
- Browsers would allow the download but reclassify the content as HTML with JavaScript

Fixes:

- Browsers give passive content *no authority*
- Modern browsers now rely on the `Content-Type` header to determine the file type
- The server can set a header:
  `X-Content-Type-Options: nosniff`
  to specifically tell the browser not to try to figure out the file's content type

Cross-origin weirdness

- **Images**
  - A frame can load images from another origin
  - 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 another 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 from another origin and execute it … but not inspect it
  - But … you can call `myfunction.toString()` to get the source

In all cases, a user can just download the content 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 send this header:

    ```
    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

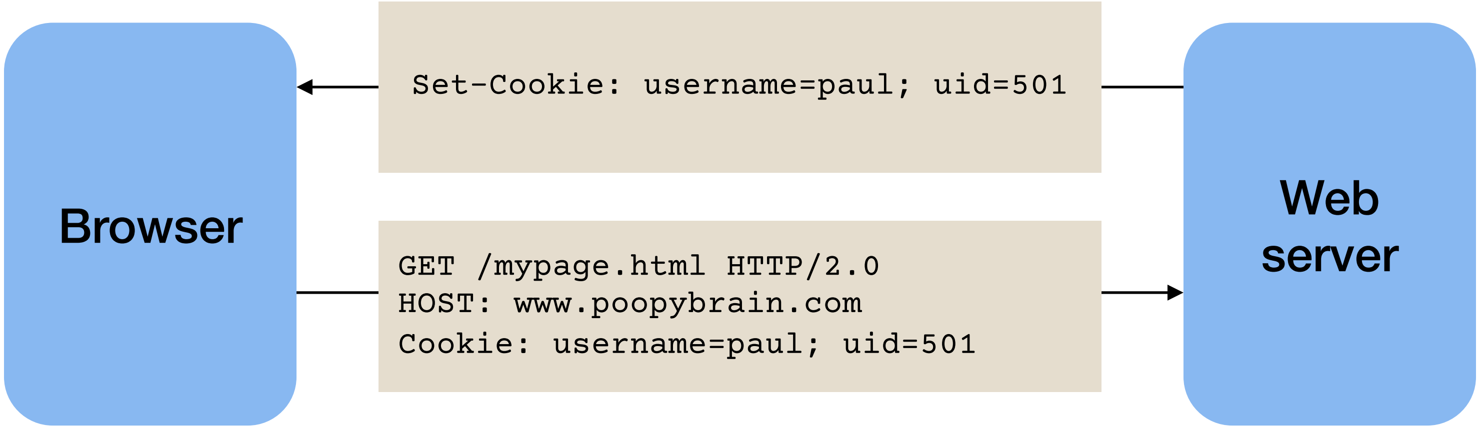
- **Solution – no foolproof solutions**
  - Don't allow DNS resolutions to return internal addresses
  - Force longer TTL even if the DNS response has a short value

HTTP Cookies

HTTP Cookies

Cookies are a mechanism created to allow websites to manage browser state

- **Cookie** = small chunk of data sent by a server to a browser with a page
- The 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 sends 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: Allows servers & JavaScript to configure the user experience

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

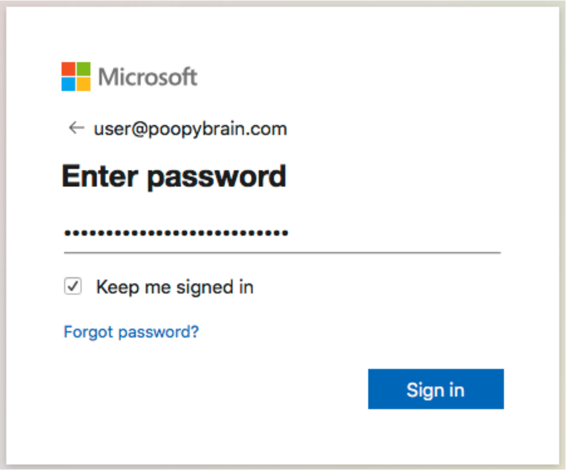
What are cookies used for?

3. Tracking: track user activity across different websites

- Server creates a cookie with a unique ID if the 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, 21 Apr 2025 17:30:00 GMT;`

When & where are cookies sent?

Cookies follow the same-origin rules *but* can be made more flexible:

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

domain	domain to send the cookie with each HTTP request **Default domain**: cookie belongs to the domain of the origin The server can specify a domain for a cookie in the `Set-Cookie` HTTP header **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
```

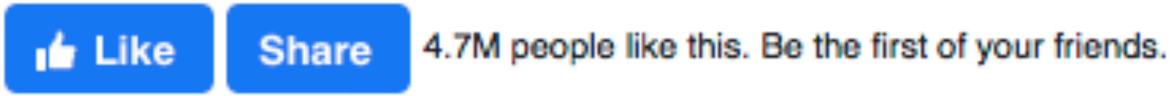
Third-party cookies: tracking cookies

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)
- Example:
  - You visit `news.com`
  - A script from `ads.tracker.com` loads an ad and sets a cookie
  - The cookie belongs to `ads.tracker.com`, making it a third-party cookie.

- **Web page = a collection of first-party and third-party content**
  - Third-party servers get their cookie whenever you visit a page where they have a presence
  - The server can record parent page URL, 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 – which is automatically run in the victim's browser when they visit the site
  - **Drive-by download**:
    - Malicious software that is automatically downloaded onto a user's device without their knowledge or explicit consent
- **What's the harm?**
  - The script can redirect the page to another site & download & run 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
  - The attacker wants to get the victim to make a specific web request that contains the commands or parameters that the attacker wants to run with the victim's credentials (stored in cookies at the victim's browser

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

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

```
<img src="https://mybank.com?action=transfer…/">
```

Cross-Site Request Forgery (CSRF) – HTTP POST

The same attack can be done with an HTTP POST request. In this case, the parameters are not embedded in the URL

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 request to the bank will send the user's session cookie if they were previously logged 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

ING

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

YouTube

- Add videos to *Favorites*
- *Add* 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

The New York Times

- Find email address of arbitrary user

NETFLIX

- Add a movie to a user's queue

facebook

- 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**
  - CSRF Tokens:
    - Include a unique random token for every session – or set it to *HMAC(request, timestamp)*
    - Token sent via hidden fields or headers and verify them on the server
  - SameSite Cookies:
    - Set cookies with a `SameSite` flag - they will be sent only if the request is from the same origin
  - Referer validation:
    - Verify the origin of the request (*Origin* or *Referer* header)

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
- **This isn't really an attack**
  - The user has to opt in but 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

Malicious Extensions

Browser extensions

- Downloadable software modules that customize a browser
- **Extensions**: JavaScript source
- **Plug-ins**: object code (no longer supported)

Risks

- May modify the interface, replace content, access browsing history
- Users may grant permissions to bypass browser security, giving access to:
  - Storage, screen, content, native programs
  - Transmitting content

BLEEPINGCOMPUTER

Chrome extensions with 6 million installs have hidden tracking code

By Bill Toulas April 17, 2025 12:50 PM 2

A set of 57 Chrome extensions with 6,000,000 users have been discovered with very risky capabilities, such as monitoring browsing behavior, accessing cookies for domains, and potentially executing remote scripts.

These extensions are 'hidden,' meaning they don't show up on Chrome Web Store searches, nor do search engines index them, and can only be installed if the user has the direct URL.

Typically, such extensions are private software like internal company tools or add-ons still under development. Still, threat actors might be using them to evade detection while aggressively pushing them through ads and malicious sites.

Risky Chrome extensions

The extensions were discovered by Secure Annex researcher John Tuckner, who uncovered the first 35 after examining what he claims is a suspicious extension named 'Fire Shield Extension Protection.'

The extension is heavily obfuscated and contains callbacks to an API for sending information collected from the browser.

https://www.bleepingcomputer.com/news/security/chrome-extensions-with-6-million-installs-have-hidden-tracking-code/

Unsanitized Input Attacks

Input sanitization

Remember command injection attacks?

Any user input must be parsed carefully

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

An attacker can set `untrusted_data` **to something like:**

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

Which can make the page content look like this:

```
<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 the

- 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"
```

This is good – what the developer expected

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

This is not good – what the developer didn't test!

OS command injection

Servers that use web form data in an OS 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");
?>
```

The user can enter a filename that will cause system() to run multiple commands

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 – attacker adds a message containing HTML and malicious JavaScript

Item Description Page:

```
<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
  - The 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

  For example, the server page might try to present the attack string as an error message

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

What's the harm of Cross-Site Scripting?

- 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**
  - Use a framework that sanitizes: 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 world's glyphs

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

Potential for deception

<u>They’re totally different to software but look the same to humans</u>

/ = 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 ≠ Ρаυl

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. paypaI.com (I instead of L)
- **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

google.com or **googie.com**

instagram.com or **instɑgraм.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 & Combosquatting

- **Misspelled or deceptive domain names**
  - Confuse people into thinking the domain is something else
  - **Typosquatting**: hope that users make a typo when they type a URL
  - **Combosquatting**: use a variant of a domain that might make a user think is valid
- **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)
- **Deceptive domain names are often used in phishing campaigns**

Typosquatting:
`www.googlec.om`
`mybank.co`

Combosquatting:
`chase-bank.com`
`office365-support.com`

The .gov Domain: Helping Mitigate Election Office Cybersecurity and Impersonation Risks

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 examples from the past:

- 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
```

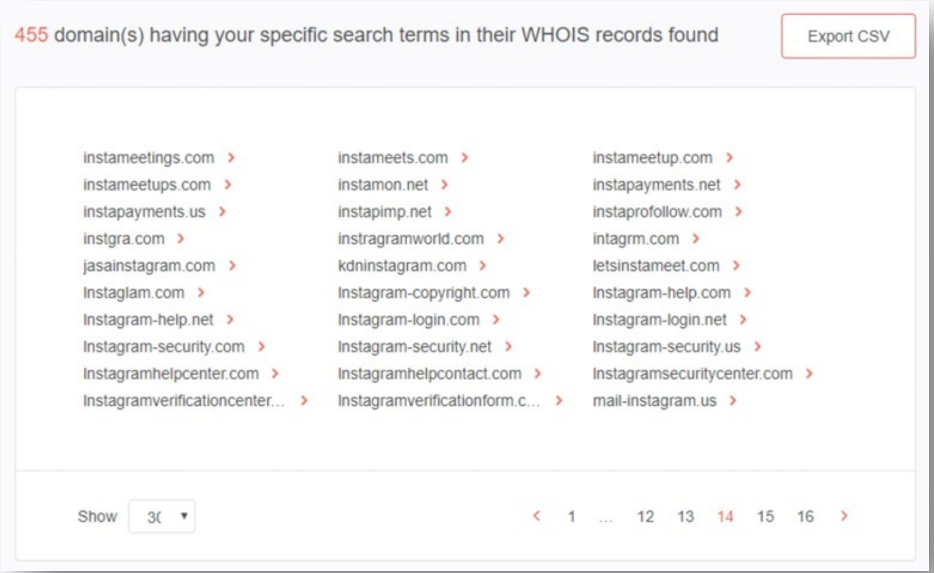
Typosquatting data feed: https://typosquatting.whoisxmlapi.com

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

```
"dependencies": {
  "express": "^4.3.0",
  "dustjs-helpers": "~1.6.3",
  "continuation-local-storage": "^3.1.0",
  "pplogger": "^0.2",
  "auth-paypal": "^2.0.0",
  "wurfl-paypal": "^1.0.0",
  "analytics-paypal": "~1.0.0"
}
```

- **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 a malicious webpage element**
- **The user sees this:**

- **But does not realize there's an invisible frame *over* the image placed by malicious JavaScript**
- **Clicking on the frame could generate a Facebook *like***
  … or download malware … or change security settings for a plugin
- **Defenses**
  - **Client-side:** JavaScript in the legitimate code to check that it's the top layer
    `window.self == window.top`
    but client checks are not reliable
  - **Server-side:** 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 security issues with images?

HTML image tags & tracking pixels

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

- **A URL may pass arguments**
  - Communicate with other sites
- **Hide the image: spy pixel (tracker pixel)**

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

By downloading this image, the browser will send pk.org ALL the cookies associated with the domain.

The extra_information can contain site-specific info, like a message ID or page ID.

The server can look at the Referer header to find the URL of the page where the image was embedded.

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"/>
```

Install the Facebook pixel if you want to retarget your website visitors.

The Facebook pixel is a small snippet of code that you, your website engineer or a Facebook Marketing Partner can paste in your code. It tracks the people and the types of actions they take when they engage with your brand, including any of your Facebook ads they saw before going to your website, the pages of your site they visit and the items they add to their carts.

- **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."

What you see isn't always what you get!

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

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>
```

What you see isn't always what you get!

Deception via images & page layout

Social engineering: add logos to fool a user

- Impersonate a site
- Impersonate credentials
- Provide a false sense of security

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