



# CSE 127: Computer Security

## Modern client-side defenses

Deian Stefan

# Today

How can we build **flexible** and **secure** client-side web applications (from vulnerable/untrusted components)

# Modern web sites are complicated

The screenshot displays the New York Times website's 'N.Y. / Region' section. The browser window shows the URL [www.nytimes.com/pages/nyregion/index.html?module=HPMiniNav&contentCollection=New York&pgtype=Homepage&region=TopBar&action=click&t](http://www.nytimes.com/pages/nyregion/index.html?module=HPMiniNav&contentCollection=New York&pgtype=Homepage&region=TopBar&action=click&t). The page features a top navigation bar with the 'The New York Times' logo, a search icon, and links for 'SUBSCRIBE NOW', 'SIGN IN', and 'Register'. A large advertisement for Volkswagen's 'TDI CleanDiesel Event' is prominently displayed, offering a lease for \$289/month for a 2014 Golf TDI. Below the ad, the main article is titled 'In New York, Hard Choices on Health Exchange Spell Success' by ANEMONA HARTOCOLLIS, published 49 minutes ago. The article includes a photo of Malka Percal at a doctor's appointment. To the right of the article is a 'SIDE STREET' section titled 'An Artist Takes His Pay in Coffee and Community' by DAVID GONZALEZ, featuring a photo of David Ellis. Further right is a 'BIG CITY BOOK CLUB' section titled 'Brooklyn' with a photo of a storefront. On the far right, there is a Twitter feed for @NYTMETRO and a 'VIDEO' section featuring a portrait of a man. The page is cluttered with various elements, including multiple ads, social media links, and a complex navigation structure, illustrating the complexity of modern web design.

New York Region News - ...

www.nytimes.com/pages/nyregion/index.html?module=HPMiniNav&contentCollection=New York&pgtype=Homepage&region=TopBar&action=click&t

The New York Times

N.Y. / Region

SUBSCRIBE NOW SIGN IN Register

Volkswagen TDI CleanDiesel Event

Explore Offers SERRAMONTE VOLKSWAGEN

Lease a 2014 Golf TDI for \$289 /mo for 36 months. \$0 Down

Das Auto.

## In New York, Hard Choices on Health Exchange Spell Success

By ANEMONA HARTOCOLLIS 49 Minutes Ago



Chang W. Lee/The New York Times

Malka Percal, 63, had her blood pressure checked and got an EKG test at a doctor's appointment. Her co-pay was \$75.

More than 900,000 residents signed up for health plans, and premiums have dropped, though the state limited consumers' choices.

### SIDE STREET

#### An Artist Takes His Pay in Coffee and Community

By DAVID GONZALEZ

David Ellis, whose other work often fetches tens of thousands of dollars, is giving back to his neighborhood by painting a large mural in a bodega in Fort Greene, Brooklyn.

• More Side Street Columns »

### BIG CITY BOOK CLUB

#### Brooklyn

Brooklyn today is a destination for celebrities and wealthy creative types,



### VIDEO »

More New York Videos »



f t e +

### FASHION

#### Intersection: Elmhurst Style Ease

In the Queens neighborhood of Elmhurst, locals like to keep their style casual.

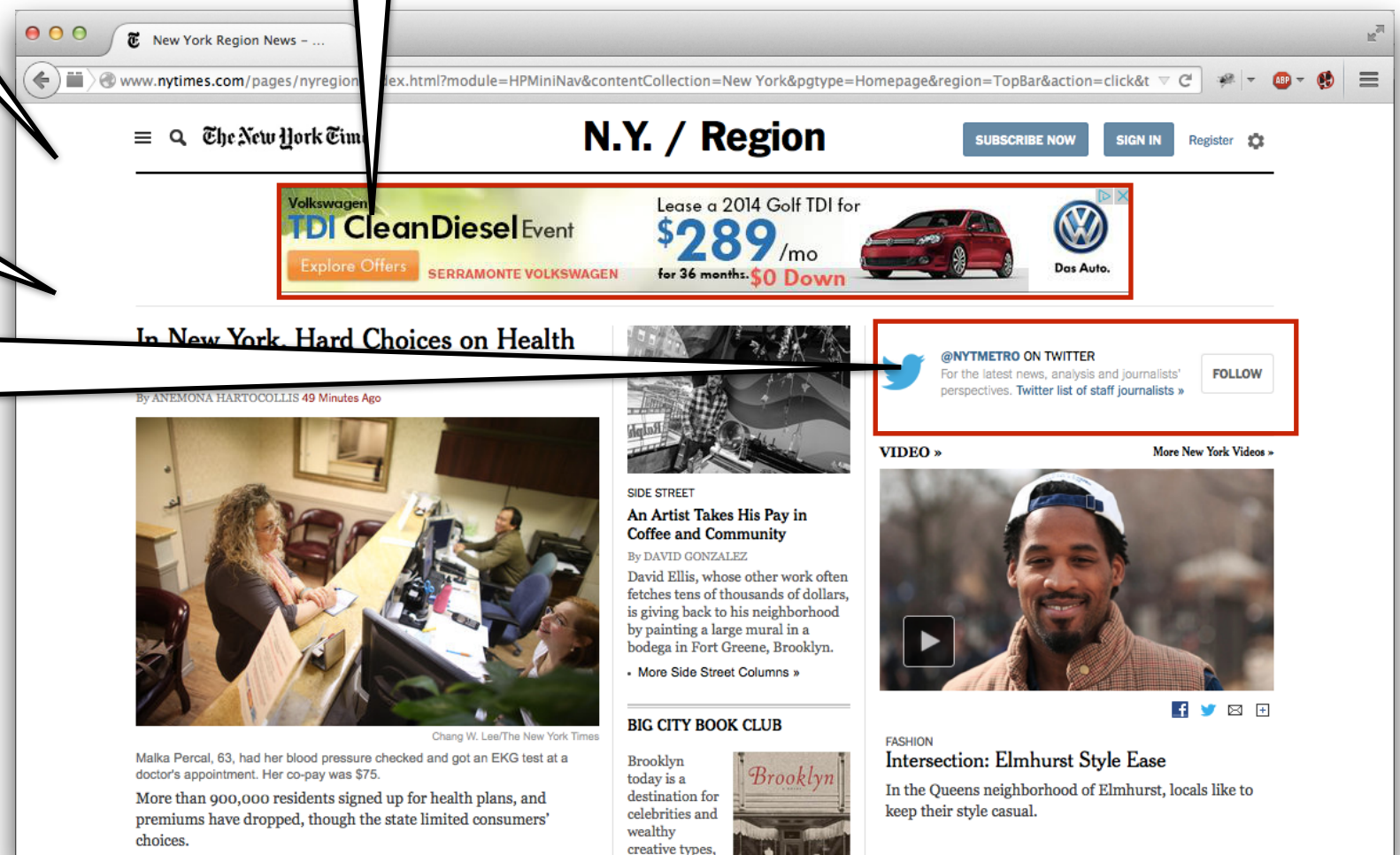
# Modern web sites are complicated

Page code

Ad code

3rd-party libs

3rd-party frame



# Many acting parties on a site

- Page developer
- Library developers
- Service providers
- Data provides
- Ad providers
- CDNs
- Network provider

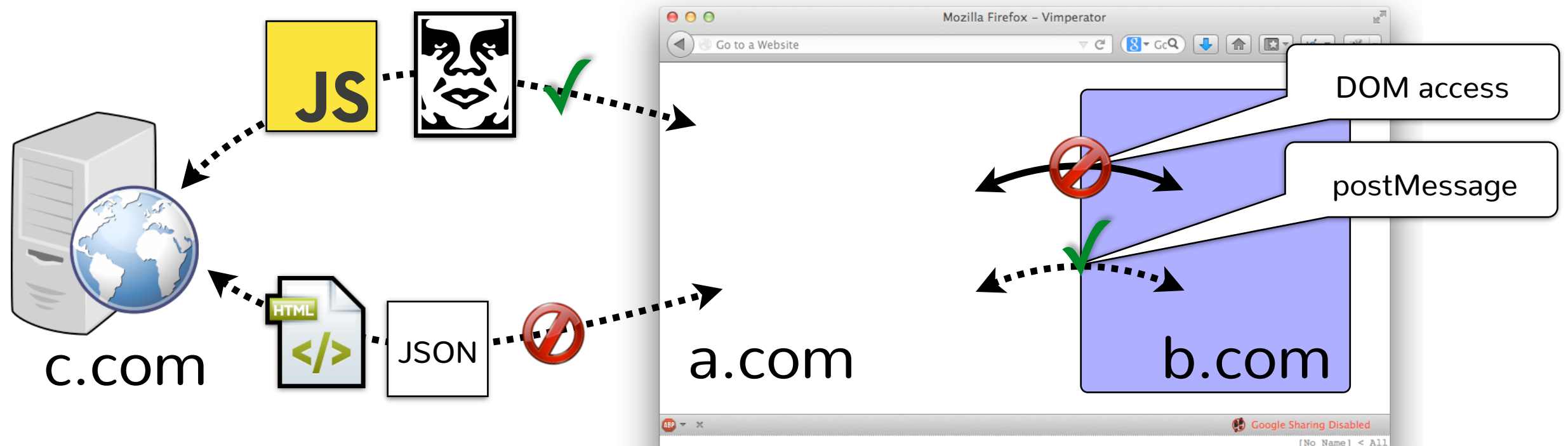
- How do we protect page from ads/services?
- How to share data with a cross-origin page?
- How to protect one user from another's content?
- How do we protect the page from a library?
- How do we protect the page from the CDN?
- How do we protect the page from network provider?



# Recall: Same origin policy

**Idea:** isolate content from different origins

- E.g., can't access document of cross-origin page
- E.g., can't inspect responses from cross-origin

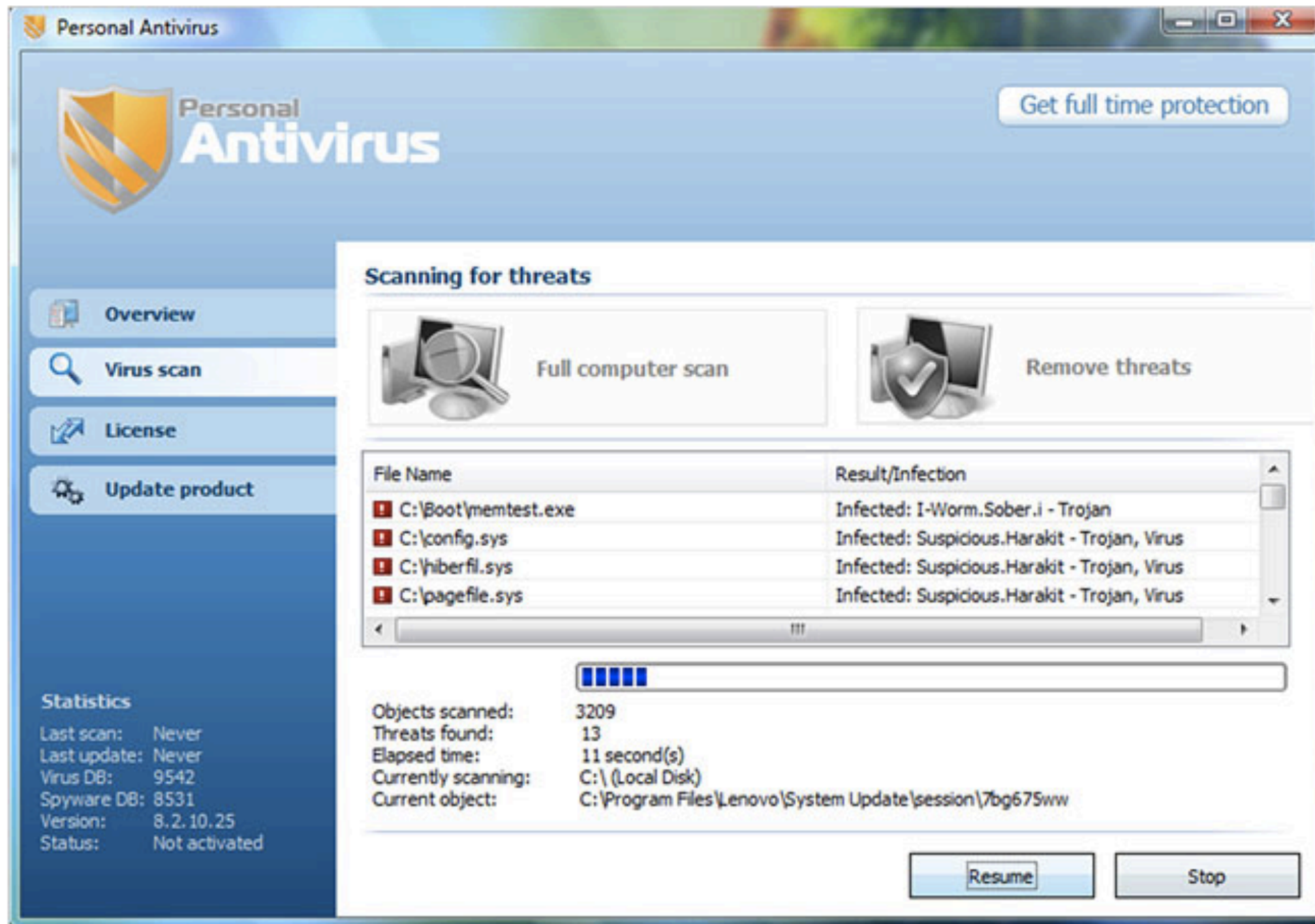


Why is the SOP not good enough?



# The SOP is not strict enough

- Third-party libs run with privilege of the page
- Code within page can arbitrarily leak data
  - How?
- iframes isolation is limited
  - Can't isolate user-provided content from page (why?)
  - Can't isolate third-party ad placed in iframe (why?)



why?)

- Can't isolate third-party ad placed in iframe (why?)



**Personal Antivirus**

Get full time protection

**Scanning for threats**

Full computer scan | Remove threats

File Name	Result/Infection
C:\Boot\memtest.exe	Infected: I-Worm.Sober.i - Trojan
C:\config.sys	Infected: Suspicious.Harakit - Trojan, Virus
C:\hiberfil.sys	Infected: Suspicious.Harakit - Trojan, Virus
C:\pagefile.sys	Infected: Suspicious.Harakit - Trojan, Virus

Statistics

- Last scan: Never
- Last update: Never
- Virus DB: 9542
- Spyware DB: 8531
- Version: 8.2.10.25
- Status: Not activated

Objects scanned: 3209  
Threats found: 13  
Elapsed time: 11 second(s)

➤ Can't isolate the



**The New York Times** ✓  
@nytimes



+ Follow

Attn: NYTimes.com readers: Do not click pop-up box warning about a virus -- it's an unauthorized ad we are working to eliminate.



# The SOP is not flexible enough

- Can't read cross-origin responses
  - What if we want to fetch data from provider.com?
  - JSONP
    - To fetch data, insert new script tag:  
`<script src="https://provider.com/getData?cb=f"></script>`
    - To share data, reply back with script wrapping data  
`f({ ...data... })`
  - Why is this a terrible idea?

# The SOP is not flexible enough

- Can't read cross-origin responses
  - What if we want to fetch data from provider.com?
  - JSONP
    - To fetch data, insert new script tag:  
`<script src="https://provider.com/getData?cb=f"></script>`
    - To share data, reply back with script wrapping data  
`f({ ...data... })`
  - Why is this a terrible idea?
    - Provider data can easily be leaked (CSRF)
    - Page is not protected from provider (XSS)

The SOP doesn't make for some things...

# Outline: modern mechanisms

- iframe sandbox
- Content security policy (CSP)
- HTTP strict transport security (HSTS)
- Subresource integrity (SRI)
- Cross-origin resource sharing (CORS)



# iframe sandbox

**Idea:** restrict actions iframe can perform

**Approach:** set sandbox attribute, by default:

- disallows JavaScript and triggers (autofocus, autoplay videos etc.)
- disallows form submission
- disallows popups
- disallows navigating embedding page
- runs page in unique origin: no storage/cookies

# Whitelisting privileges

Can enable dangerous features by whitelisting:

- **allow-scripts:** allows JS + triggers (autofocus, autoplay, etc.)
- **allow-forms:** allow form submission
- **allow-pointer-lock:** allow fine-grained mouse moves
- **allow-popups:** allow iframe to create popups
- **allow-top-navigation:** allow breaking out of frame
- **allow-same-origin:** retain original origin

# What can you do with iframe sandbox?

- Run content in iframe with least privilege
  - Only grant content privileges it needs
- Privilege separate page into multiple iframes
  - Split different parts of page into sandboxed iframes

# Least privilege: twitter button

```
<a class="twitter-share-button" href="https://twitter.com/share">Tweet</a>
<script>
window.twttr=(function(d,s,id){var js,fjs=d.getElementsByTagName(s)
[0],t=window.twttr||{};if(d.getElementById(id))return
t;js=d.createElement(s);js.id=id;js.src="https://platform.twitter.com/
widgets.js";fjs.parentNode.insertBefore(js,fjs);t._e=[];t.ready=function(f)
{t._e.push(f);};return t;}(document,"script","twitter-wjs"));
</script>
```

- What's the problem with this embedding approach?

# Least privilege: twitter button

```
<a class="twitter-share-button" href="https://twitter.com/share">Tweet</a>
<script>
window.twttr=(function(d,s,id){var js,fjs=d.getElementsByTagName(s)
[0],t=window.twttr||{};if(d.getElementById(id))return
t;js=d.createElement(s);js.id=id;js.src="https://platform.twitter.com/
widgets.js";fjs.parentNode.insertBefore(js,fjs);t._e=[];t.ready=function(f)
{t._e.push(f);};return t;}(document,"script","twitter-wjs"));
</script>
```

➤ What's the problem with this embedding approach?

- Using iframes

```
<iframe src="https://platform.twitter.com/widgets/tweet_button.html"
style="border: 0; width:130px; height:20px;"></iframe>
```

➤ What's the problem without sandbox flag?

# Least privilege: twitter button

- With sandbox: remove all permissions and then enable JS, popups, form submission, etc.

```
<iframe src="https://platform.twitter.com/widgets/tweet_button.html"  
  sandbox="allow-same-origin allow-scripts allow-popups allow-forms"  
  style="border: 0; width:130px; height:20px;"></iframe>
```

# Privilege separation: blog feed

- Typically include user content inline:

```
<div class="post">  
  <div class="author">{{post.author}}</div>  
  <div class="body">{{post.body}}</div>  
</div>
```

- Problem with this?



# Privilege separation: blog feed

- Typically include user content inline:

```
<div class="post">  
  <div class="author">{{post.author}}</div>  
  <div class="body">{{post.body}}</div>  
</div>
```

- Problem with this?

- With iframe sandbox:

```
<iframe sandbox srcdoc="...  
<div class="post">  
  <div class="author">{{post.author}}</div>  
  <div class="body">{{post.body}}</div>  
</div>..."></iframe>
```

- May need allow-scripts - why?
- Is allow-same-origin safe to whitelist?

What are some limitations of iframe sandbox?

# Too strict vs. not strict enough

- Consider running library in sandboxed iframes
  - E.g., password strength checker



- **Desired guarantee:** checker cannot leak password
- Problem: sandbox does not restrict exfiltration
  - Can use XHR to write password to b.ru

# Too strict vs. not strict enough

- Can we limit the origins that the page (iframe or otherwise) can talk to?
  - Can only leak to a trusted set of origins
  - Gives us a more fine-grained notion of least privilege
- This can also prevent or limit damages due to XSS

# Outline: modern mechanisms

- iframe sandbox (quick refresher)

## Content security policy (CSP)

- HTTP strict transport security (HSTS)
- Subresource integrity (SRI)
- Cross-origin resource sharing (CORS)

# Content Security Policy (CSP)

- **Idea:** restrict resource loading to a whitelist
  - By restricting to whom page can talk to: restrict where data is leaked!
- **Approach:** send page with CSP header that contains fine-grained directives
  - E.g., allow loads from CDN, no frames, no plugins

```
Content-Security-Policy: default-src https://cdn.example.net;  
child-src 'none'; object-src 'none'
```

script-src: where you can load scripts from

connect-src: limits the origins you can XHR to

font-src: where to fetch web fonts from

form-action: where forms can be submitted

child-src: where to load frames/workers from

img-src: where to load images from

...

default-src: default fallback



# Special keywords

- 'none' - match nothing
- 'self' - match this origin
- 'unsafe-inline' - allow unsafe JS & CSS
- 'unsafe-eval' - allow unsafe eval (and the like)
- http: - match anything with http scheme
- https: - match anything with https scheme

# How can CSP help with XSS?

- If you whitelist all places you can load scripts from:
  - Only execute code from trusted origins
  - Remaining vector for attack: inline scripts
- CSP by default disallows inline scripts
  - If scripts are enabled at least it disallows eval

# Adoption challenge

- Problem: inline scripts are widely-used
  - Page authors use the 'unsafe-inline' directive
  - Is this a problem?

# Adoption challenge

- Problem: inline scripts are widely-used
  - Page authors use the 'unsafe-inline' directive
  - Is this a problem?
- Solution: ~~script nonce~~ and script hash
  - Allow scripts that have a particular hash
  - Allow scripts that have a white-listed nonce

# Other adoption challenges

- Goal: set most restricting CSP that is permissive enough to not break existing app
- How can you figure this out for a large app?
  - CSP has a report-only header and report-uri directive
  - Report violations to server; don't enforce
- In practice: devs hardly ever get the policy right

# How is CSP really used in practice?

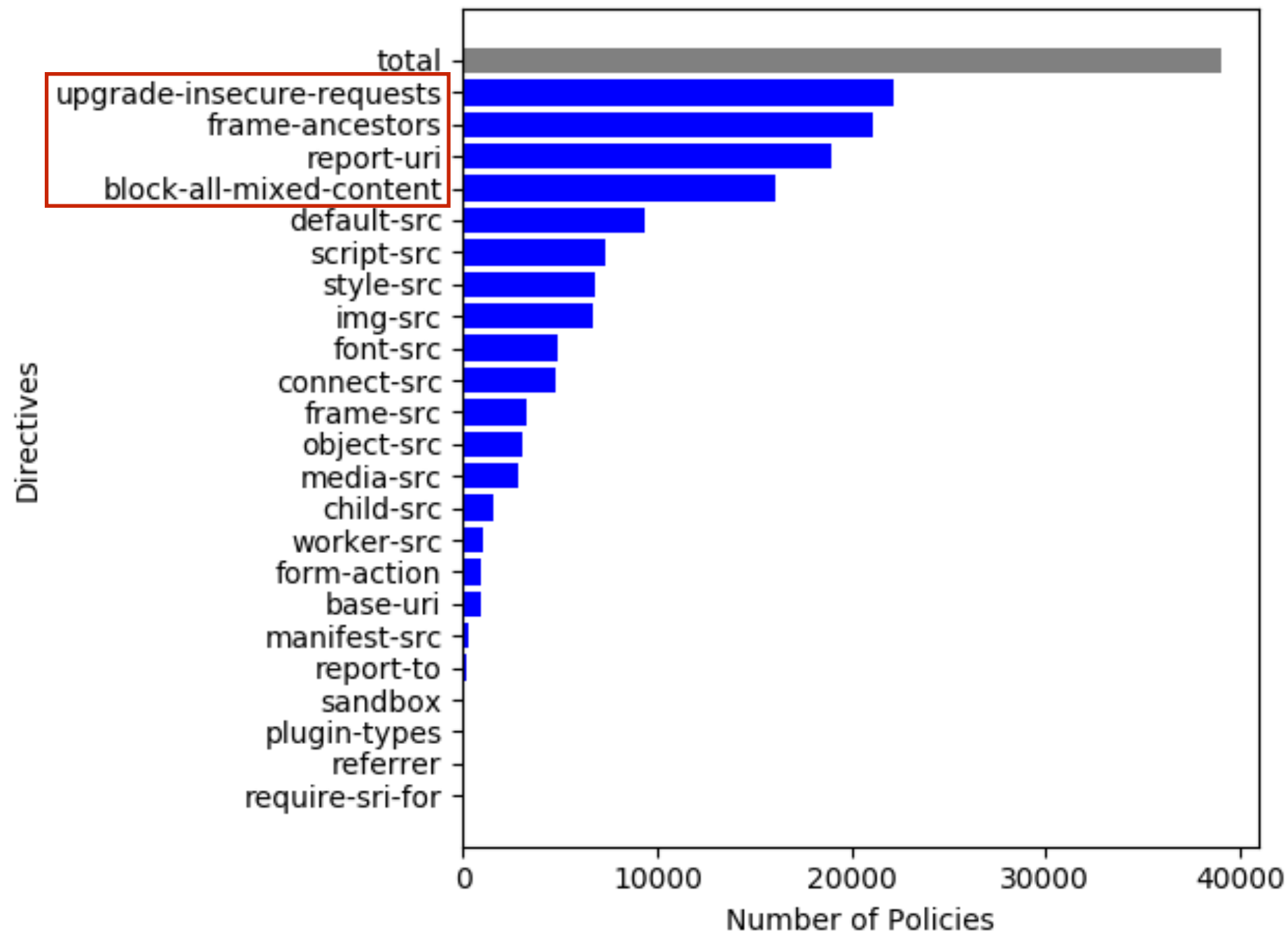


Figure 1: Distribution of CSP directives.

# How is CSP really used in practice?

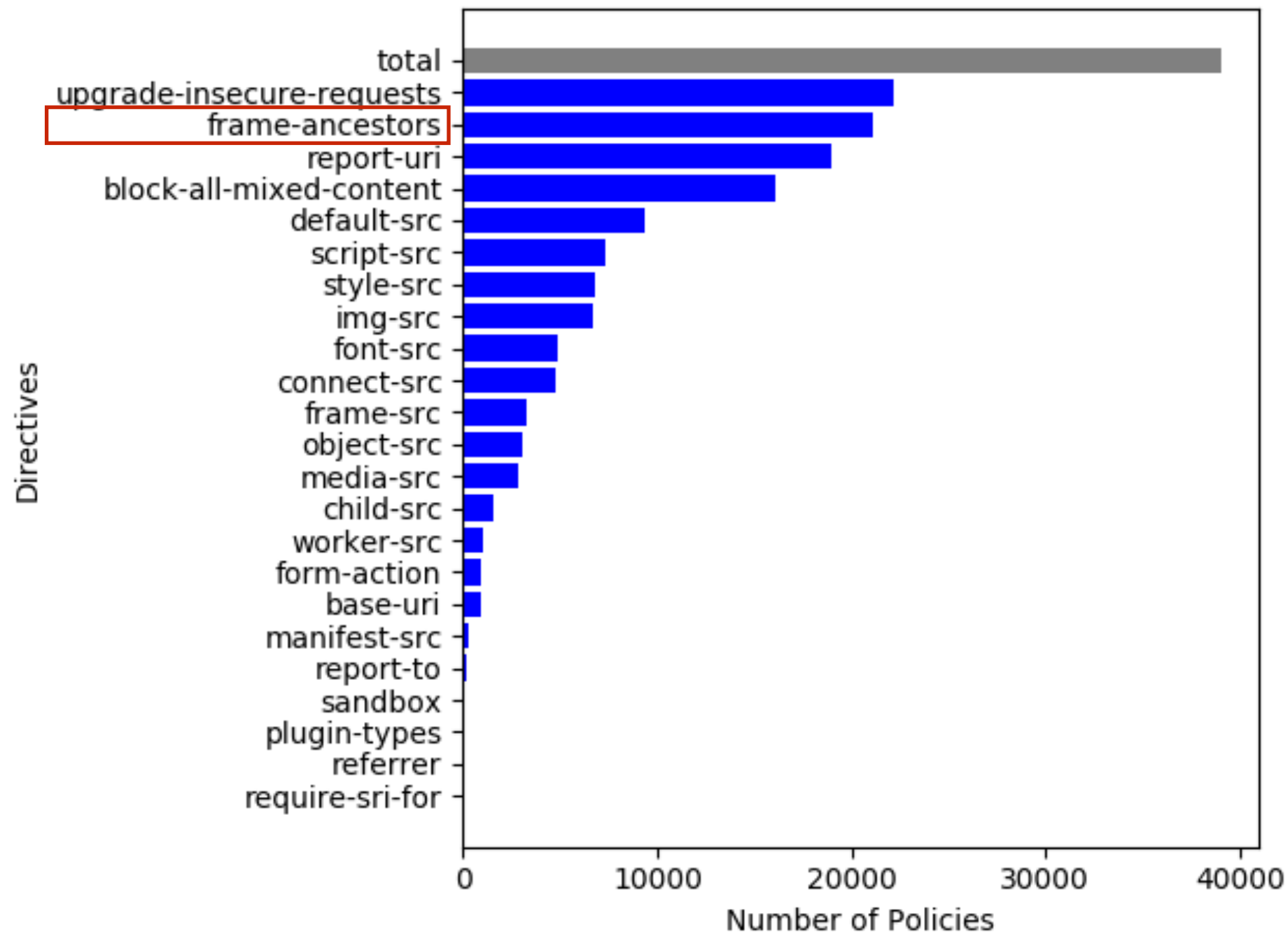


Figure 1: Distribution of CSP directives.



# What's frame-ancestors?

The HTTP `Content-Security-Policy` (CSP) `frame-ancestors` directive specifies valid parents that may embed a page using `<frame>`, `<iframe>`, `<object>`, `<embed>`, or `<applet>`.

Setting this directive to `'none'` is similar to `X-Frame-Options : deny` (which is also supported in older browsers).



## What problem is this addressing?

# Clickjacking!



- How does frame-ancestor help?
  - Don't allow non twitter origins to frame delete page!

# What about the other two?

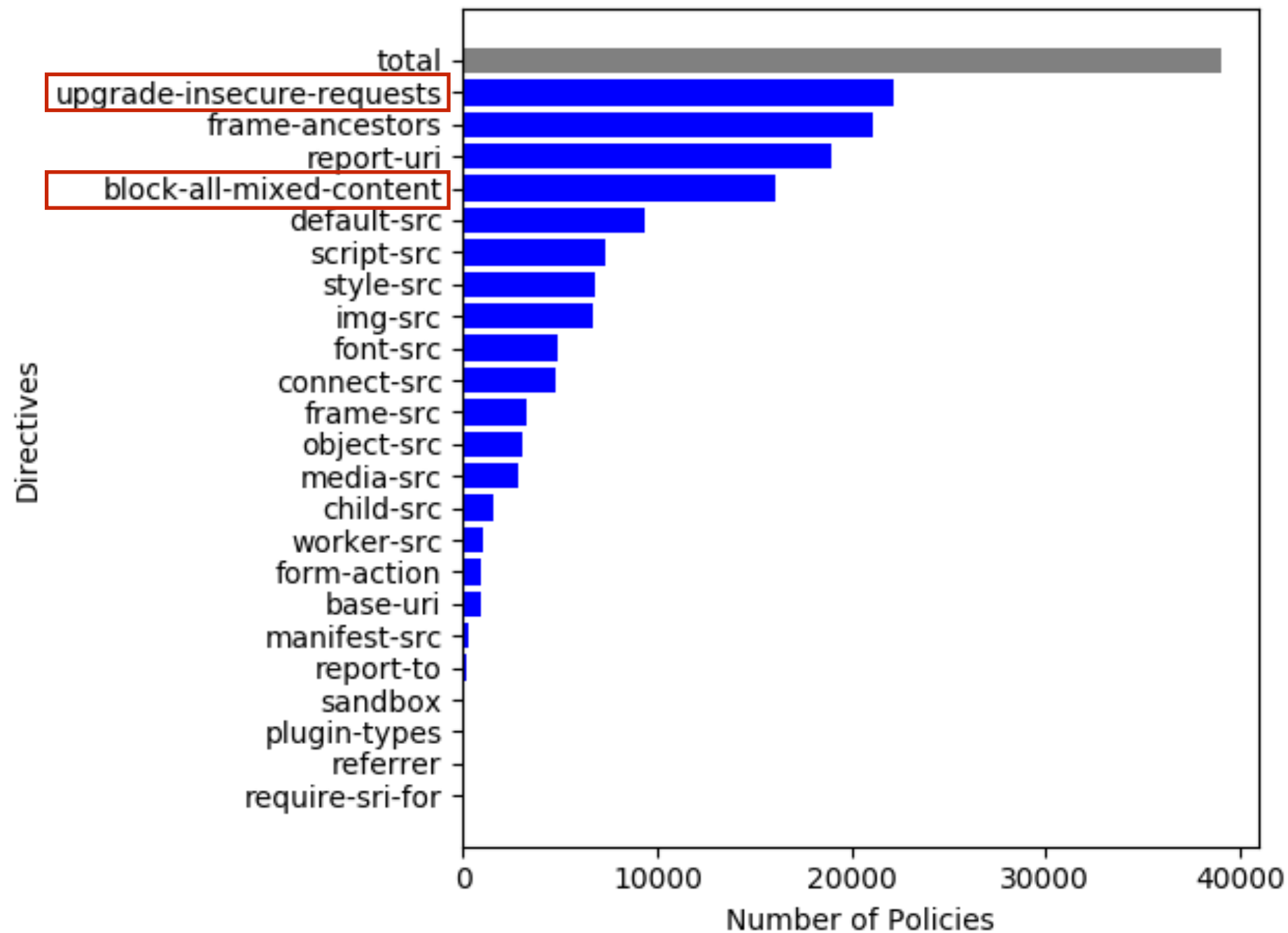
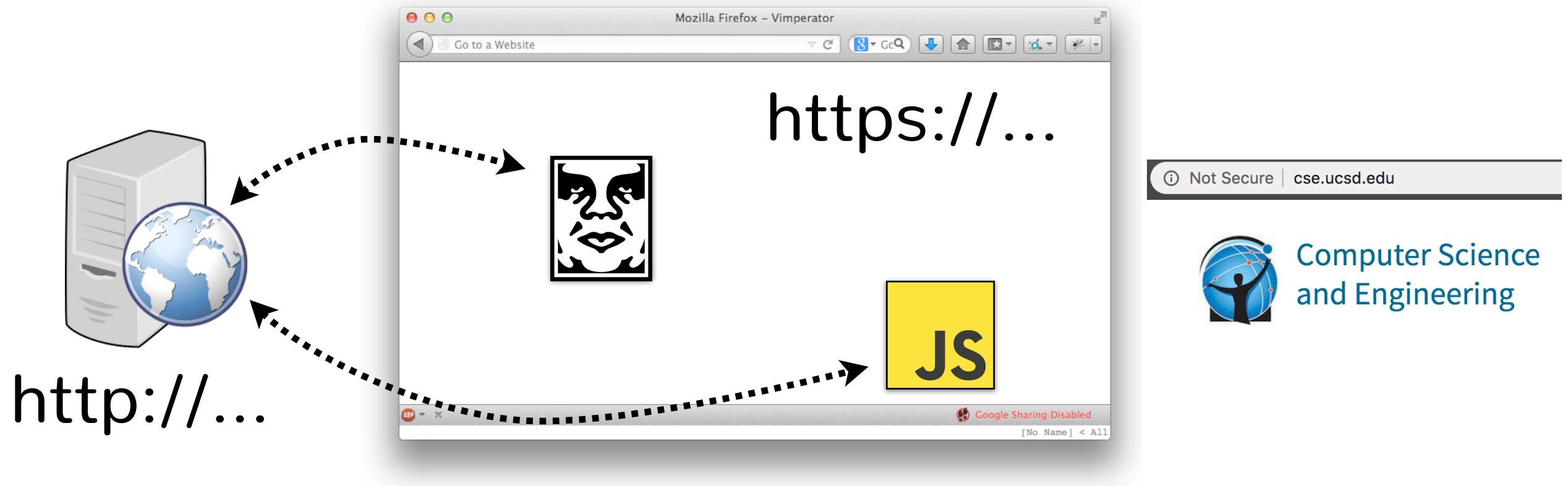


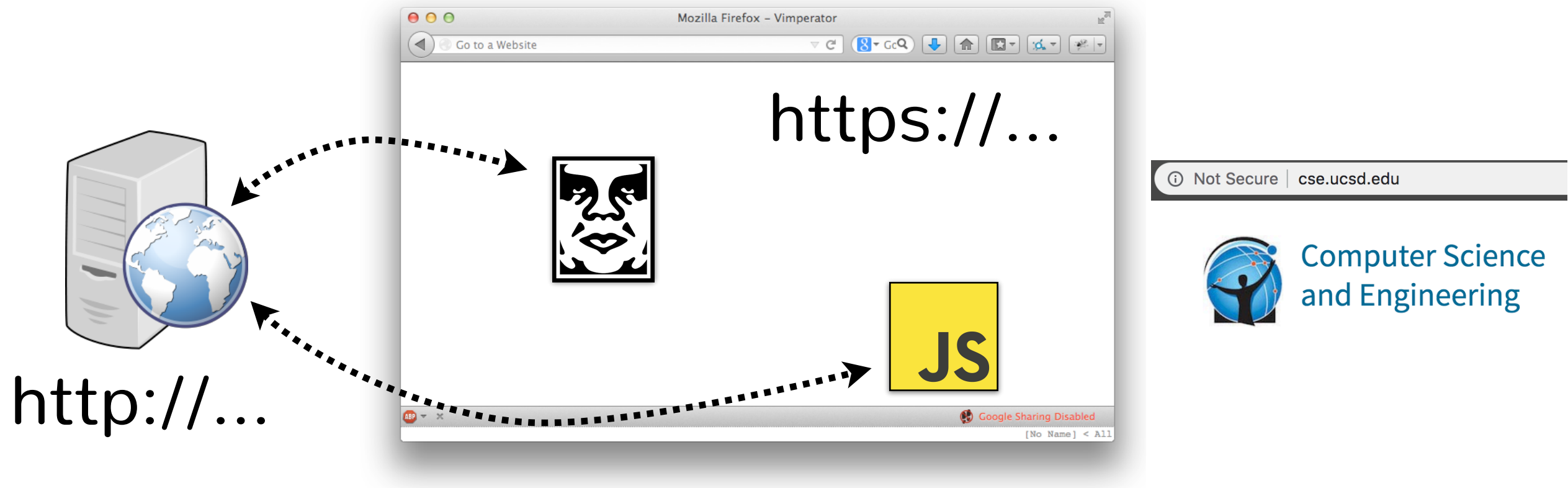
Figure 1: Distribution of CSP directives.

# What is MIXed content?



- Why is this bad?

# What is MIXed content?



- Why is this bad?
  - Network attacker can inject their own scripts, images, etc.!

# How does CSP help?

- upgrade-insecure-requests
  - Essentially rewrite every HTTP URL to HTTPS before making request
- block-all-mixed-content
  - Don't load any content over HTTP

# Outline: modern mechanisms

- iframe sandbox (quick refresher)
- Content security policy (CSP)
- ➔ HTTP strict transport security (HSTS)
- Subresource integrity (SRI)
- Cross-origin resource sharing (CORS)

# Motivation for HSTS

- Attacker can force you to go to HTTP vs. HTTPS
  - SSL Stripping attack (Moxie)
    - They can rewrite all HTTPS URLs to HTTP
    - If server serves content over HTTP: doom!
- HSTS header: never visit site over HTTP again
  - Strict-Transport-Security: max-age=31536000



# Outline: modern mechanisms

- iframe sandbox (quick refresher)
- Content security policy (CSP)
- HTTP strict transport security (HSTS)
- ➔ Subresource integrity (SRI)
- Cross-origin resource sharing (CORS)

# Motivation for SRI

- CSP+HSTS can be used to limit damages, but can't really defend against malicious code
- How do you know that the library you're loading is the correct one?

**Massive denial-of-service attack on GitHub tied to Chinese government**

Reports: Millions of innocent Internet users conscripted into Chinese DDoS army.

Won't using HTTPS address this problem?

**jQuery.com compromised to serve malware via drive-by download**

## MaxCDN

*Date: 2013-07-02*

MaxCDN, a content-delivery network service had their servers compromised. MaxCDN is running bootstrapcdn.com, a CDN download for popular Bootstrap front end framework.

The vendor of MaxCDN had laid off a support engineer having access to the servers where BootstrapCDN runs. The credentials of the support engineer were not properly revoked. The attackers had gained access to these credentials. The attackers rebooted the server into single-user mode, changed the root password, and SSH'd into the server. Bootstrap JavaScript files were modified to serve an exploit toolkit.

Bootstrap is widely deployed and CDN option is one of the recommended ways to include Bootstrap on your website. BootstrapCDN gets a lot of downloads. Thus, the attack payload was served to tens of thousands of visitors in short period of time.

Related evaluation points:

- [Passphrase on server login keys](#)
- [Audited server login keys](#)
- [HTTPS / TLS only](#)

Links:

- [BootstrapCDN Security Post-Mortem](#)

Mikko Ohtamaa

# Subresource integrity

- Idea: page author specifies hash of (sub)resource they are loading; browser checks integrity
  - E.g., integrity for scripts

```
<link rel="stylesheet" href="https://site53.cdn.net/style.css"  
      integrity="sha256-SDfwewFAE...wefjijfE">
```

- E.g., integrity for link elements

```
<script src="https://code.jquery.com/jquery-1.10.2.min.js"  
        integrity="sha256-C6CB9UYIS9UJeqinPHWTHVqh/E1uhG5Tw+Y5qFQmYg=">
```

# What happens when check fails?

- Case 1 (default):
  - Browser reports violation and does not render/execute resource
- Case 2: CSP directive with integrity-policy directive set to report
  - Browser reports violation, but may render/execute resource

# Multiple hash algorithms

- Authors may specify multiple hashes

- E.g., 

```
<script src="hello_world.js"  
        integrity="sha256-...  
                sha512-...  
"></script>
```

- Browser uses strongest algorithm
- Why support multiple algorithms?
  - Don't break page on old browser

# Outline: modern mechanisms

- iframe sandbox (quick refresher)
- Content security policy (CSP)
- HTTP strict transport security (HSTS)
- Subresource integrity (SRI)
- ➔ Cross-origin resource sharing (CORS)

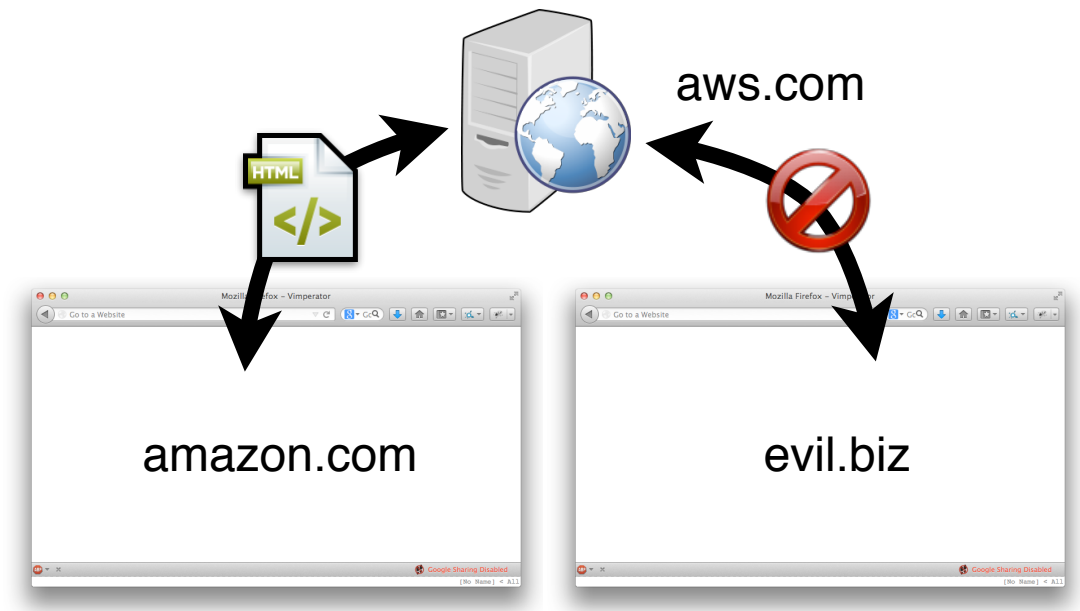
# Recall: SOP is also inflexible

- Problem: Can't fetch cross-origin data
  - Leads to building insecure sites/services: JSONP
- Solution: Cross-origin resource sharing (CORS)
  - Data provider explicitly whitelists origins that can inspect responses
  - Browser allows page to inspect response if its origin is listed in the header



# E.g., CORS usage: amazon

- Amazon has multiple domains
  - E.g., amazon.com and aws.com
- Problem: amazon.com can't read cross-origin aws.com data
- With CORS amazon.com can whitelist aws.com



# How CORS works

- Browser sends Origin header with XHR request
  - E.g., Origin: https://amazon.com
- Server can inspect Origin header and respond with Access-Control-Allow-Origin header
  - E.g., Access-Control-Allow-Origin: https://amazon.com
  - E.g., Access-Control-Allow-Origin: \*
- CORS XHR may send cookies + custom headers
  - Need “preflight” request to authorize this

- ✓ How do we protect page from ads/services?
- ✓ How to share data with cross-origin page?
- ✓ How to protect one user from another's content?
- ✓ How do we protect the page from a library?
- ✓ How do we protect the page from the CDN?
- ✓ How do we protect the page from network provider?

# References

- [Sandbox] - **Play safely in sandboxed IFrames** by Mike West.
  - <http://www.html5rocks.com/en/tutorials/security/sandboxed-iframes/>
  - <http://www.w3.org/TR/2010/WD-html5-20100624/the-iframe-element.html>
- [CSP] - **An Introduction to Content Security Policy** by Mike West.
  - <http://www.html5rocks.com/en/tutorials/security/content-security-policy/>
  - <http://www.w3.org/TR/CSP2/>
- [CORS] - **Using CORS** by Monsur Hossain.
  - <http://www.html5rocks.com/en/tutorials/cors/>
  - <http://www.w3.org/TR/cors>
- [SRI] - **Subresource Integrity** by Frederik Braun, Francois Marier, Devdatta Akhawe, and Joel Weinberger.

# References

- [COWL] - **Protecting Users by Confining JavaScript with COWL** by Deian Stefan, Edward Z. Yang, Petr Marchenko, Alejandro Russo, Dave Herman, Brad Karp, and David Mazières. In OSDI 2014
  - <http://cowl.ws>
- [HotOS] - **The Most Dangerous Code in the Browser** by Stefan Heule, Devon Rifkin, Deian Stefan, and Alejandro Russo. In HotOS 2015
  - <http://deian.org/pubs/heule:2015:the-most.pdf>
- [RAID] - **Why is CSP Failing? Trends and Challenges in CSP Adoption** by Michael Weissbacher, Tobias Lauinger, and William Robertson. In RAID, 2014.
  - [http://www.iseclab.org/people/mweissbacher/publications/csp\\_raid.pdf](http://www.iseclab.org/people/mweissbacher/publications/csp_raid.pdf)