

# Cross-Site Scripting (XSS): It's still a problem even though we have solutions.

---

Sid Stamm  
Associate Professor  
Computer Science and Software Engineering

**ROSE-HULMAN**

First: Who's this guy in front of you?

**ROSE-HULMAN**  
INSTITUTE OF TECHNOLOGY











# Overview

- A Brief History of The Web
- ... and its Security Model,
- ... which isn't secure.
- Origin Laundering
- FINALLY: Cross-Site Scripting
- Provenance and Control

# A Brief History of the Web

# Documents

## Reining in the Web with Content Security Policy

Sid Stamm  
Mozilla  
sid@mozilla.com

Brandon Sterne  
Mozilla  
bsterne@mozilla.com

Gervase Markham  
Mozilla  
gerv@mozilla.org

### ABSTRACT

The last three years have seen a dramatic increase in both awareness and exploitation of Web Application Vulnerabilities. 2008 and 2009 saw dozens of high-profile attacks against websites using Cross Site Scripting (XSS) and Cross Site Request Forgery (CSRF) for the purposes of information stealing, website defacement, malware planting, clickjacking, etc. While an ideal solution may be to develop web applications free from any exploitable vulnerabilities, real world security is usually provided in layers.

We present content restrictions, and a content restrictions enforcement scheme called Content Security Policy (CSP), which intends to be one such layer. Content restrictions al-

exploiting browser or site-specific vulnerabilities to steal or inject information.

Additionally, browser and web application providers are having a hard time deciding what exactly should be a “domain” or “origin” when referring to web traffic. With the advent of DNS rebinding [8] and with the gray area regarding ownership of sibling sub-domains (like `user1.webhost.com` versus `user2.webhost.com`), it may be ideal to allow the service providers who write web applications the opportunity to specify, or fence-in, what they consider to be their domain.

### 1.1 Uncontrolled Web Platform





# Hyper-transferring hyper-text hyper-docs

( <http://137.112.40.20/~stammsl/revolutionary-paper-is-here> )

```
GET /~stammsl/revolutionary-paper-is-here HTTP 1.0
```

```
HTTP 200 OK
```

```
[lots of data here]
```

# Documents, But Better!

## Reining in the Web with Content Security Policy

Sid Stamm  
Mozilla  
sid@mozilla.com

Brandon Sterne  
Mozilla  
bsterne@mozilla.com

Gervase Markham  
Mozilla  
gerv@mozilla.org

### ABSTRACT

The last three years have seen a dramatic increase in both awareness and exploitation of Web Application Vulnerabilities. 2008 and 2009 saw dozens of high-profile attacks against websites using Cross Site Scripting (XSS) and Cross Site Request Forgery (CSRF) for the purposes of information stealing, website defacement, malware planting, clickjacking, etc. While an ideal solution may be to develop web applications free from any exploitable vulnerabilities, real world security is usually provided in layers.

We present content restrictions, and a content restrictions enforcement scheme called Content Security Policy (CSP), which intends to be one such layer. Content restrictions al-

exploiting browser or site-specific vulnerabilities to steal or inject information.

Additionally, browser and web application providers are having a hard time deciding what exactly should be a “domain” or “origin” when referring to web traffic. With the advent of DNS rebinding [8] and with the gray area regarding ownership of sibling sub-domains (like `user1.webhost.com` versus `user2.webhost.com`), it may be ideal to allow the service providers who write web applications the opportunity to specify, or fence-in, what they consider to be their domain.

### 1.1 Uncontrolled Web Platform



# Documents, But Better!

## Reining in the Web with Content Security Policy

Sid Stamm  
Mozilla  
sid@mozilla.com

Brandon Sterne  
Mozilla  
bsterne@mozilla.com

Gervase Markham  
Mozilla  
gerv@mozilla.org

### ABSTRACT

The last three years have seen a dramatic increase in both awareness and exploitation of Web Application Vulnerabilities. 2008 and 2009 saw dozens of high-profile attacks against websites using Cross Site Scripting (XSS) and Cross Site Request Forgery (CSRF) for the purposes of information steal-

exploiting browser or site-specific vulnerabilities to steal or inject information.

Additionally, browser and web application providers are having a hard time deciding what exactly should be a “domain” or “origin” when referring to web traffic. With the advent of DNS rebinding [8] and with the gray area regarding

### 7. CONCLUSIONS

We propose the use of content restrictions to lock down web sites behavior, and have provided an implementation of content restrictions called Content Security Policy. CSP provides not only an ability for web sites to specify what types of content may be loaded (and from where), but also some protection from cross-site scripting and other common web attacks such as clickjacking.

While a site should not rely on something like CSP to provide a complete suite of security, CSP can be used as an early warning mechanism for attacks that appear in the wild, and even when not widely adopted by a majority of the web browser market, can prove a useful layer in protecting web applications and their users.

### 8. ACKNOWLEDGEMENTS

The authors would like to thank Adam Barth for all his rigorous scrutiny of CSP as it evolved. Robert “RSnake” Hansen also helped provide feedback in the early stages of the project, and helped publicize content restrictions as a

[9] C. Jackson, A. Bortz, D. Boneh, and J. C. Mitchell. Stanford safecache. <http://www.safecache.com>.

[10] C. Jackson, A. Bortz, D. Boneh, and J. C. Mitchell. Stanford safehistory. <http://www.safehistory.com>.

[11] C. Jackson, A. Bortz, D. Boneh, and J. C. Mitchell. Protecting browser state from web privacy attacks. In *WWW '06: Proceedings of the 15th international conference on World Wide Web*, pages 737–744, New York, NY, USA, 2006. ACM.

[12] M. Jakobsson and S. Stamm. Invasive browser sniffing and countermeasures. In *WWW '06: Proceedings of the 15th international conference on World Wide Web*, pages 523–532, New York, NY, USA, 2006. ACM.

injection attacks with browser-enforced embedded policies. In *WWW '07: Proceedings of the 16th international conference on World Wide Web*, pages 601–610, New York, NY, USA, 2007. ACM.

[14] N. Jovanovic, E. Kirda, and C. Kruegel. Preventing cross site request forgery attacks. In *the IEEE*

# Documents, but Betterer!

## Reining in the Web with Content Security Policy

Sid Stamm  
Mozilla  
sid@mozilla.com

Brandon Sterne  
Mozilla  
bsterne@mozilla.com

Gervase Markham  
Mozilla  
gerv@mozilla.org



Now with  
hyperlinks  
!!!

### ABSTRACT

The last three years have seen a dramatic increase in both awareness and exploitation of Web Application Vulnerabilities. 2008 and 2009 saw dozens of high-profile attacks against websites using Cross Site Scripting (XSS) and Cross Site Request Forgery (CSRF) for the purposes of information steal-

exploiting browser or site-specific vulnerabilities to steal or inject information.

Additionally, browser and web application providers are having a hard time deciding what exactly should be a “domain” or “origin” when referring to web traffic. With the advent of DNS rebinding [8] and with the gray area regarding

### 7. CONCLUSIONS

We propose the use of content restrictions to lock down web sites behavior, and have provided an implementation of content restrictions called Content Security Policy. CSP provides not only an ability for web sites to specify what types of content may be loaded (and from where), but also some protection from cross-site scripting and other common web attacks such as clickjacking.

While a site should not rely on something like CSP to provide a complete suite of security, CSP can be used as an early warning mechanism for attacks that appear in the wild, and even when not widely adopted by a majority of the web browser market, can prove a useful layer in protecting web applications and their users.

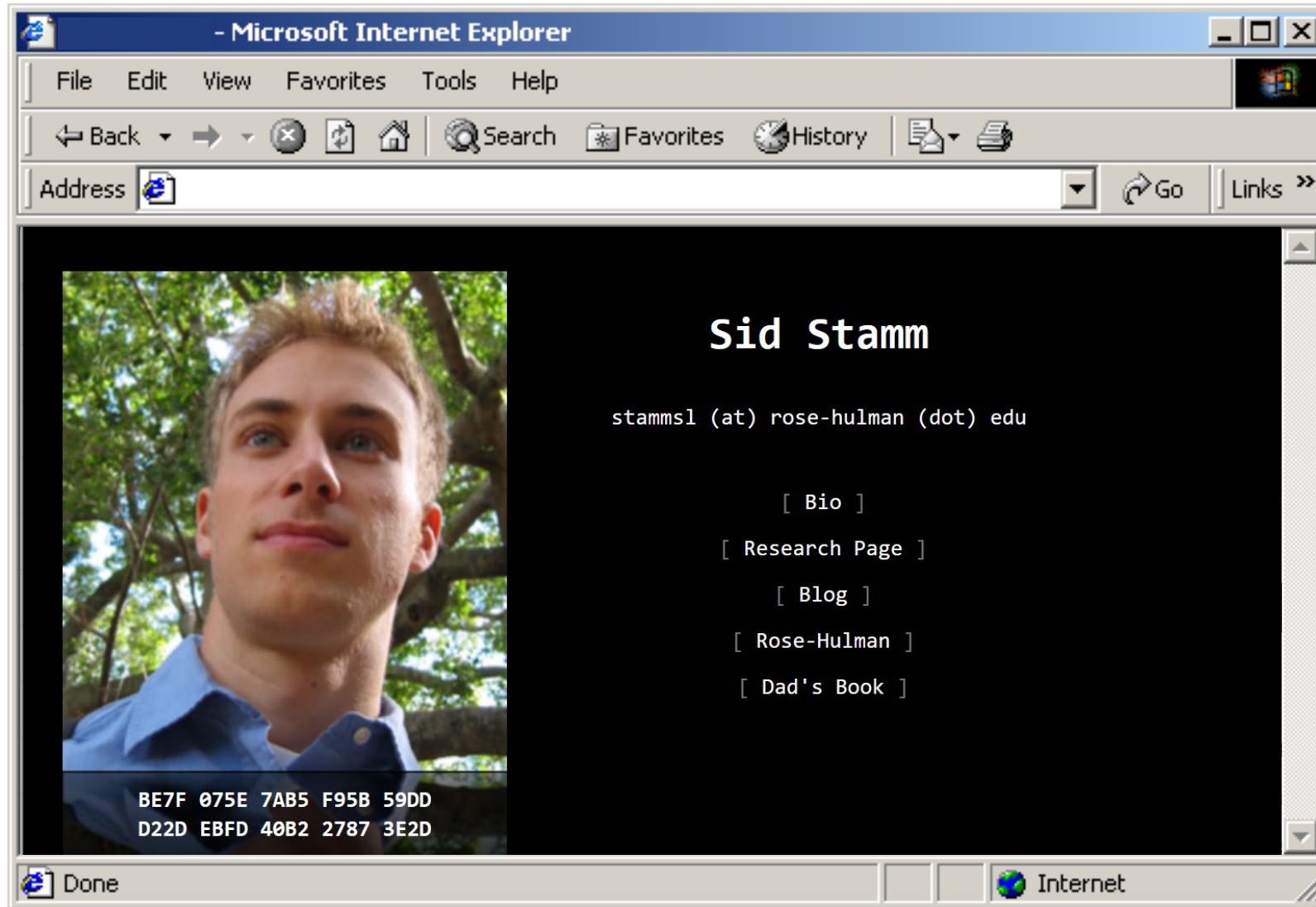
### 8. ACKNOWLEDGEMENTS

The authors would like to thank Adam Barth for all his rigorous scrutiny of CSP as it evolved. Robert “RSnake” Hansen also helped provide feedback in the early stages of the project, and helped publicize content restrictions as a

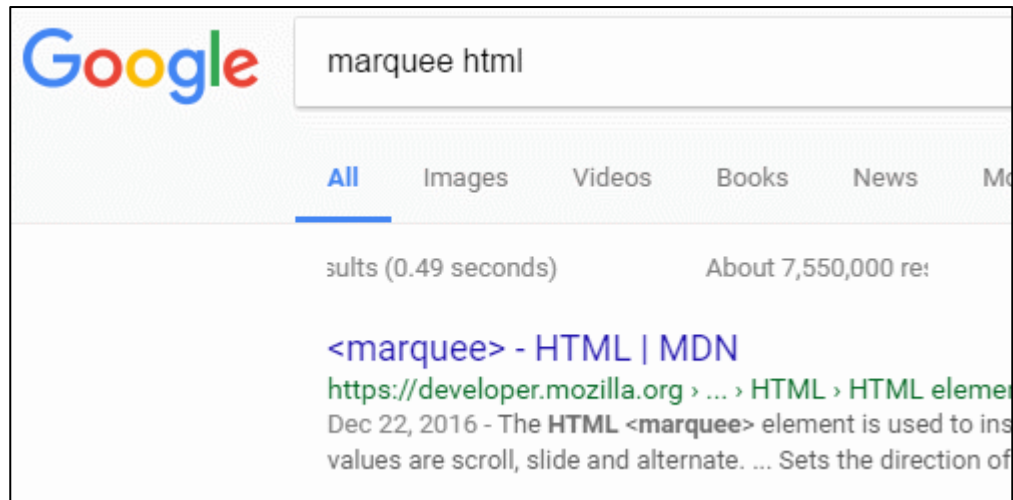
- [9] C. Jackson, A. Bortz, D. Boneh, and J. C. Mitchell. Stanford safecache. <http://www.safecache.com>.
- [10] C. Jackson, A. Bortz, D. Boneh, and J. C. Mitchell. Stanford safehistory. <http://www.safehistory.com>.
- [11] C. Jackson, A. Bortz, D. Boneh, and J. C. Mitchell. Protecting browser state from web privacy attacks. In *WWW '06: Proceedings of the 15th international conference on World Wide Web*, pages 737–744, New York, NY, USA, 2006. ACM.
- [12] [M. Jakobsson and S. Stamm. Invasive browser sniffing and countermeasures. In WWW '06: Proceedings of the 15th international conference on World Wide Web, pages 523–532, New York, NY, USA, 2006. ACM.](#)
- [13] T. Jim, N. Swamy, and M. Hicks. Defeating script injection attacks with browser-enforced embedded policies. In *WWW '07: Proceedings of the 16th international conference on World Wide Web*, pages 601–610, New York, NY, USA, 2007. ACM.
- [14] N. Jovanovic, E. Kirda, and C. Kruegel. Preventing cross site request forgery attacks. In *the IEEE*



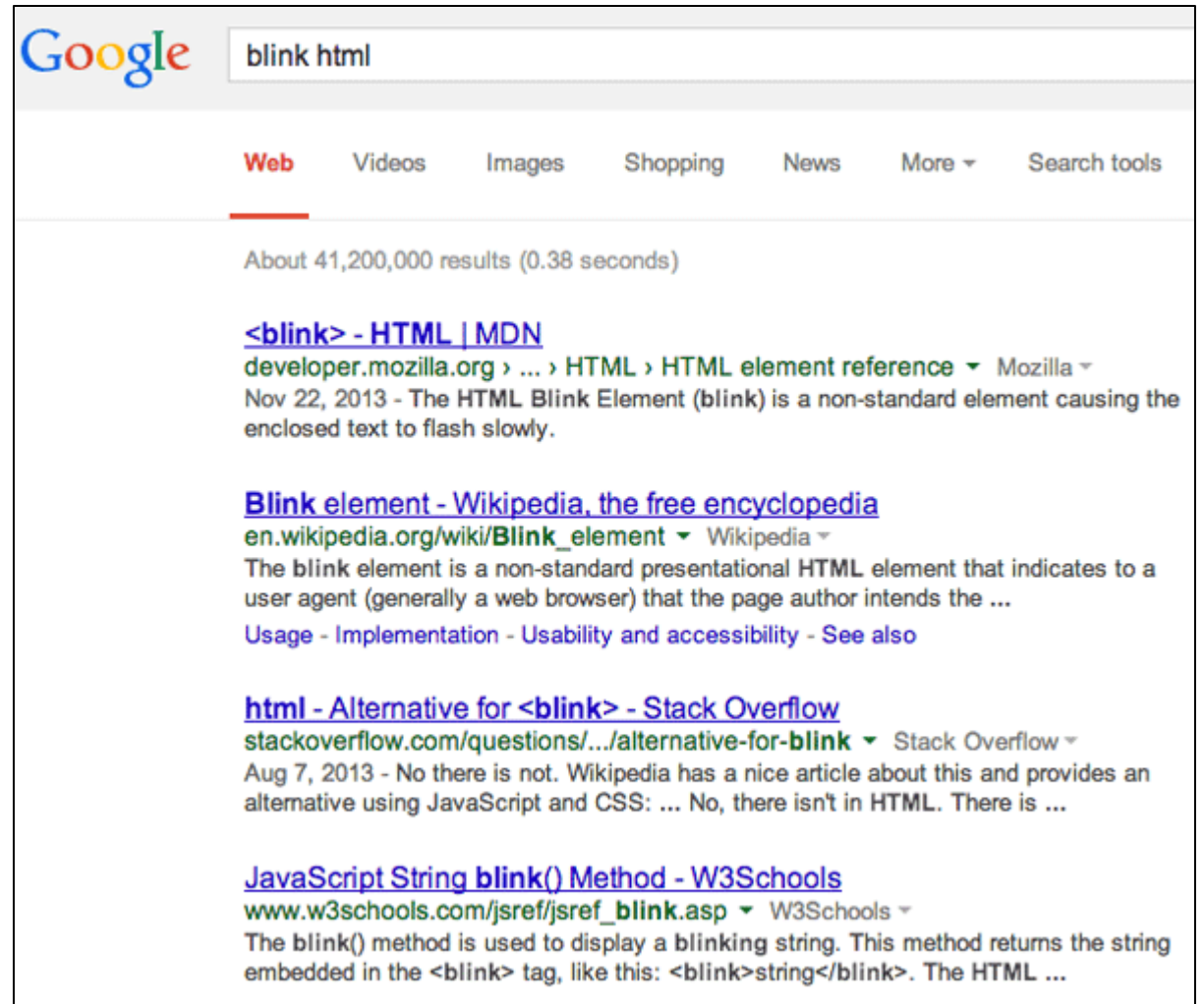
# Pretty documents



# Pretty documents



A screenshot of a Google search interface. The search bar contains the text "marquee html". Below the search bar, there are tabs for "All", "Images", "Videos", "Books", "News", and "More". The "All" tab is selected. Below the tabs, the search results are displayed. The first result is titled "<marquee> - HTML | MDN" and includes a snippet of text: "https://developer.mozilla.org > ... > HTML > HTML element reference > Mozilla > Nov 22, 2016 - The HTML <marquee> element is used to ins values are scroll, slide and alternate. ... Sets the direction of".



A screenshot of a Google search interface. The search bar contains the text "blink html". Below the search bar, there are tabs for "Web", "Videos", "Images", "Shopping", "News", "More", and "Search tools". The "Web" tab is selected. Below the tabs, the search results are displayed. The first result is titled "<blink> - HTML | MDN" and includes a snippet of text: "developer.mozilla.org > ... > HTML > HTML element reference > Mozilla > Nov 22, 2013 - The HTML Blink Element (blink) is a non-standard element causing the enclosed text to flash slowly." The second result is titled "Blink element - Wikipedia, the free encyclopedia" and includes a snippet of text: "en.wikipedia.org/wiki/Blink\_element > Wikipedia > The blink element is a non-standard presentational HTML element that indicates to a user agent (generally a web browser) that the page author intends the ... Usage - Implementation - Usability and accessibility - See also". The third result is titled "html - Alternative for <blink> - Stack Overflow" and includes a snippet of text: "stackoverflow.com/questions/.../alternative-for-blink > Stack Overflow > Aug 7, 2013 - No there is not. Wikipedia has a nice article about this and provides an alternative using JavaScript and CSS: ... No, there isn't in HTML. There is ...". The fourth result is titled "JavaScript String blink() Method - W3Schools" and includes a snippet of text: "www.w3schools.com/jsref/jsref\_blink.asp > W3Schools > The blink() method is used to display a blinking string. This method returns the string embedded in the <blink> tag, like this: <blink>string</blink>. The HTML ...".

# Interactive Documents

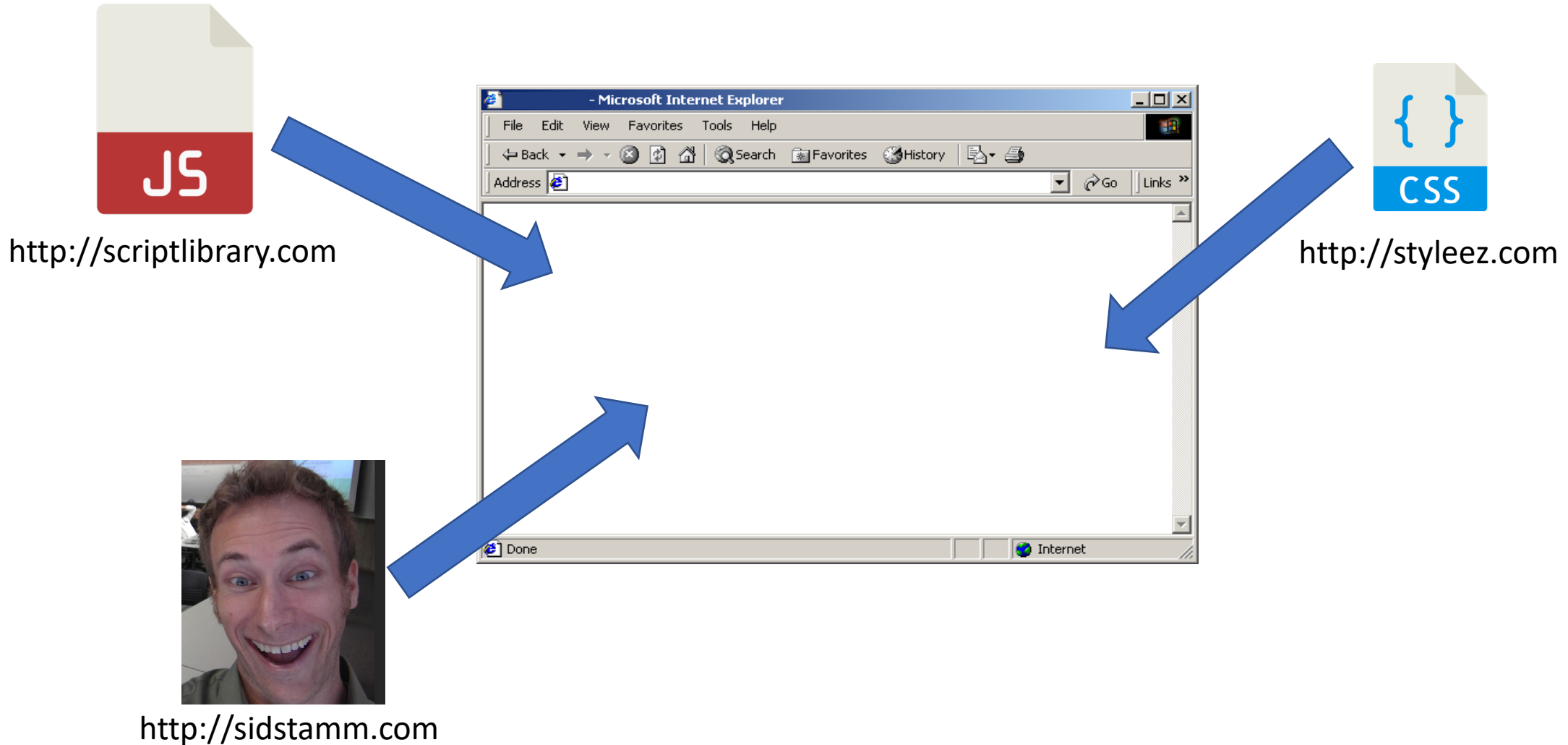
```
<img onclick="window.alert('ouch!');">
```

```
<form action="http://secret.com/putithere">  
  <input type="submit"  
    onclick="window.validate(this);">  
</form>
```



**Brendan Eich**  
( Invented JavaScript )

# Interactive Documents in Many Pieces





# Documents made of many documents!

*The World's Worst Website*

Gratuitous use of frames is a common mistake of web designers.

Many browsers do not support frames. They disrupt the flow of the website and can be difficult to anticipate where a page may appear when a link is clicked.

If you must use frames, use the tag `<base target="_blank">` between `<head>` and `</head>` to assure links will open in a new window.

Check out these links to websites whose opinions about frames is self evident:

[The "I Hate Frames" Frames Page](#)

[Another I Hate Frames Page](#)

[The International I Hate Frames Club](#)



BE7F 075E 7AB5 F95B 59DD  
D22D EBF4 40B2 2787 3E2D

**Sid Stamm**

stammsl (at) rose-hulman (dot) edu

[ Bio ]

[ Research Page ]

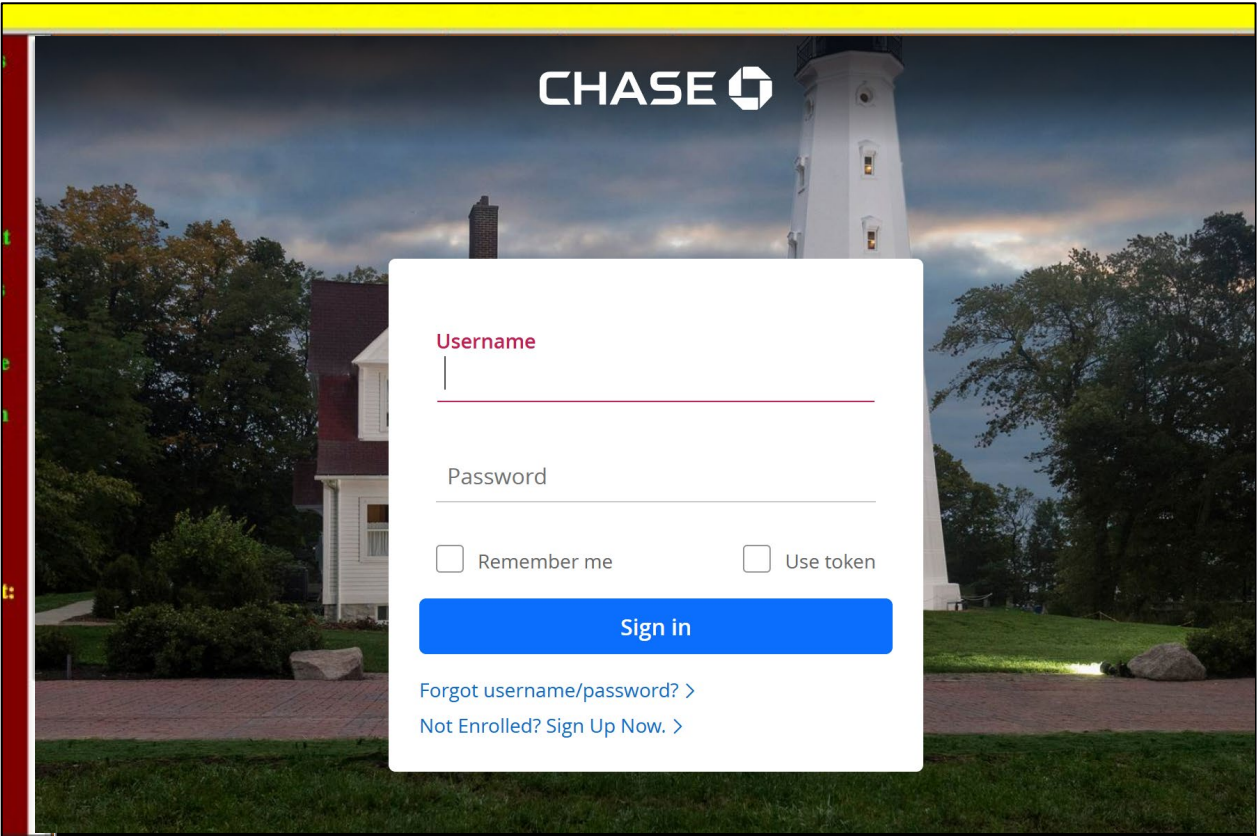
[ Blog ]

[ Rose-Hulman ]

[ Dad's Book ]

© 2020 Sid Stamm

# Thin-Frame Attack



# Setting a Boundary

- SAME ORIGIN POLICY



# Documents have become Interactive apps that we still pretend are documents

Now  
with  
the  
thing

The screenshot shows a web browser interface with a network inspector overlay. The overlay displays a table of network requests:

Status	Method	Domain	File
200	POST	www.linkedin.com	track
200	GET	www.linkedin.com	graphql?
200	POST	www.linkedin.com	track
200	POST	www.linkedin.com	track

Below the table, the overlay shows summary statistics: 267 requests, 34.92 MB / 4.91 MB transferred, and Finish: 51.96 s.

The background shows a LinkedIn page with a navigation bar at the top, a search bar, and a list of news articles. A FixDex advertisement is visible in the middle of the page.

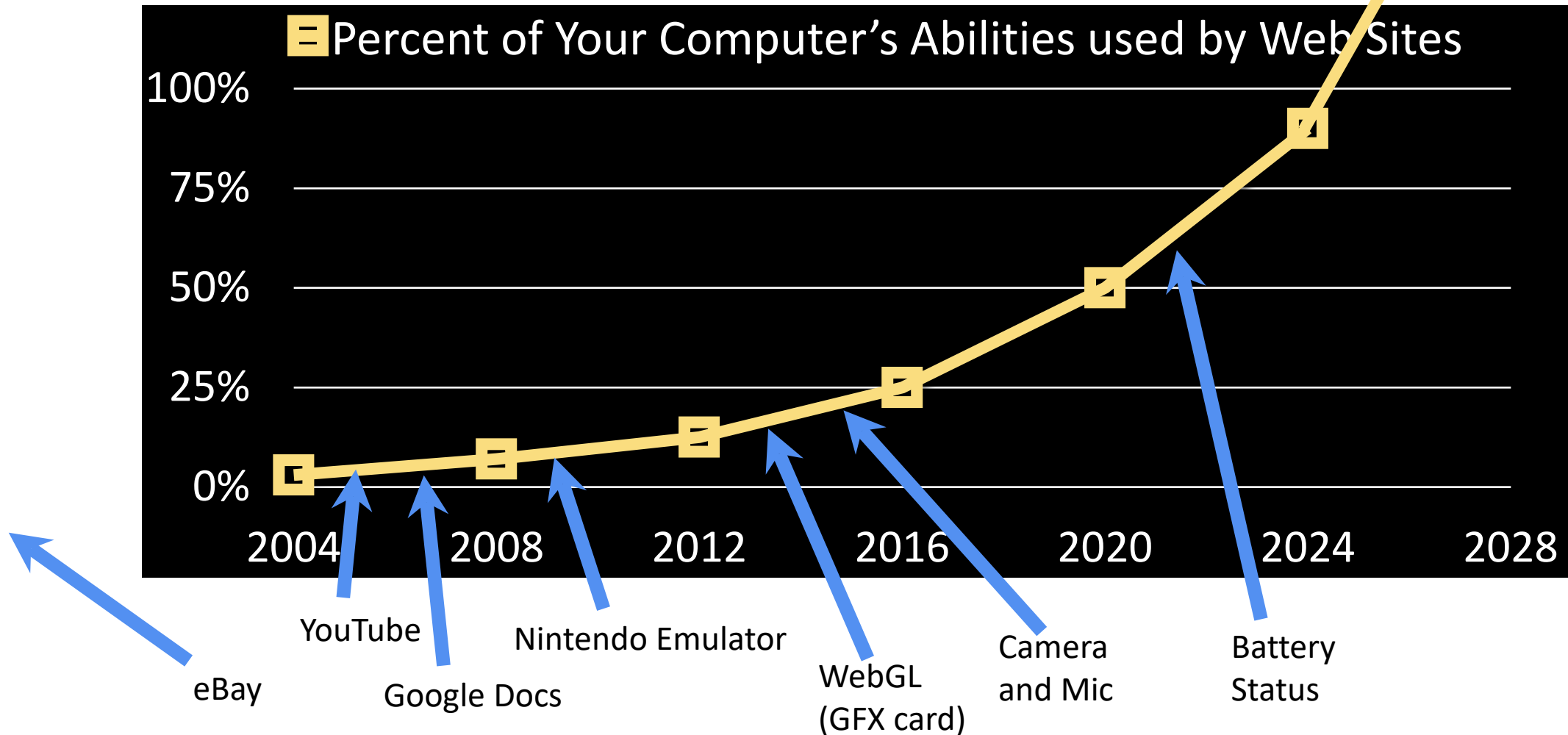
This screenshot shows the network tab of a developer tool. It displays a detailed view of a network request:

Initiator	Type	Transferred	Size	0 ms
4kmg4d0sew3r95ot76j5c9wzpz...	plain	143.81 kB	0 B	143 ms
ables={0&8queryId=vo...}	vnd.linkedin...	1.22 kB	551 B	95 ms
4kmg4d0sew3r95ot76j5c9wzpz...	plain	14.06 kB	0 B	83 ms
4kmg4d0sew3r95ot76j5c9wzpz...	plain	4.67 kB	0 B	61 ms

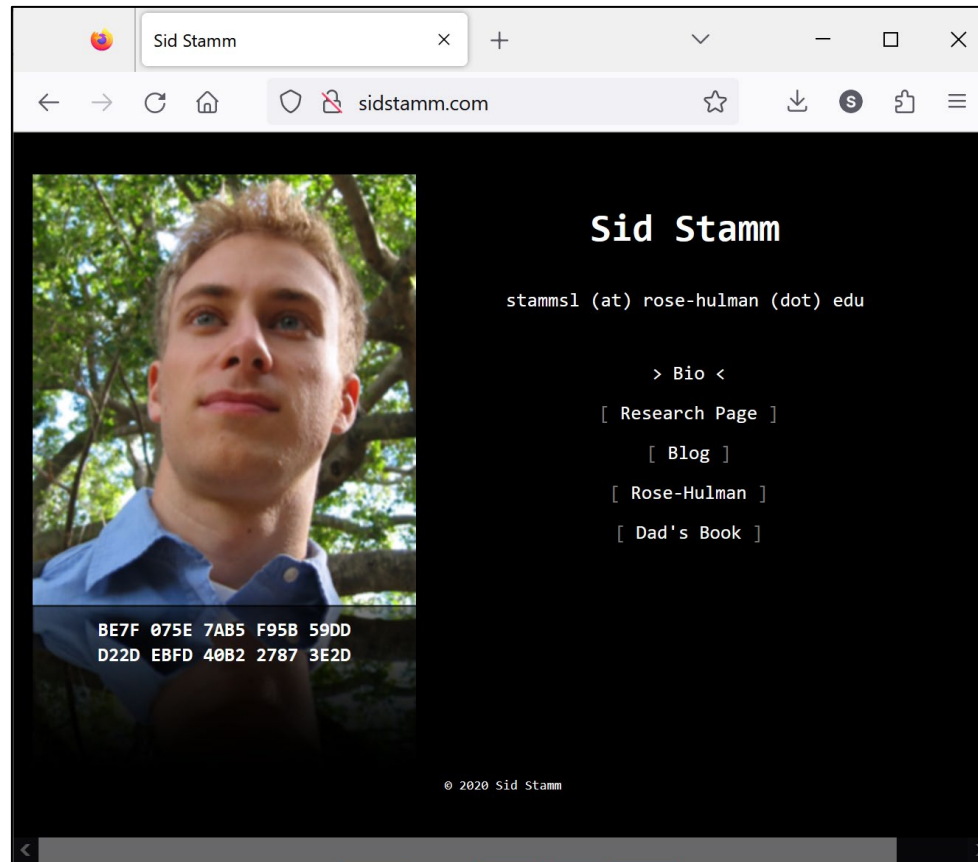
At the bottom, it shows summary statistics: 267 requests, 34.92 MB / 4.91 MB transferred, and Finish: 51.96 s.



Documents are now:  
Interactive apps that we still pretend are documents



# Bonus Lesson: Web browsers are also web pages!



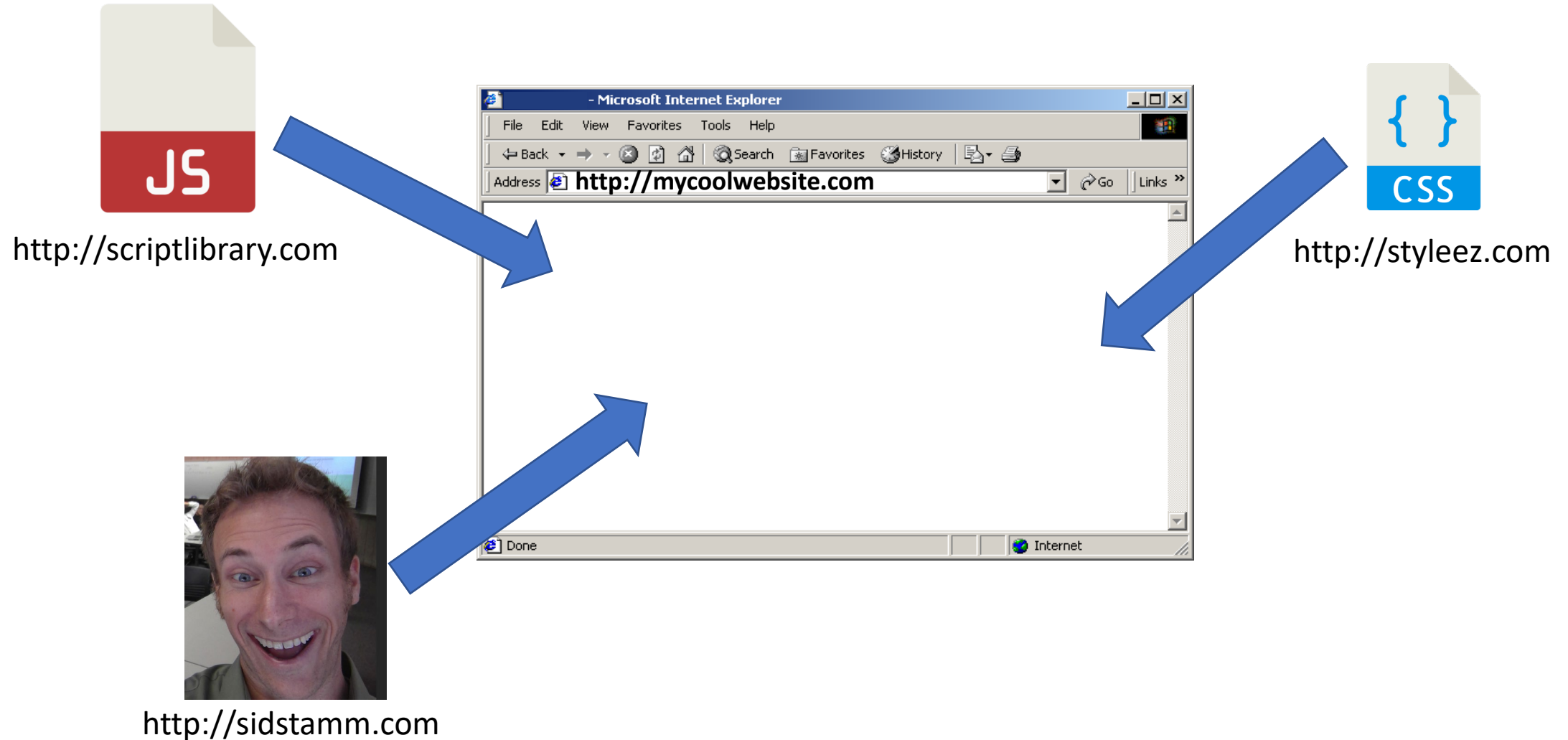
Yep! That's a web page.

The "real" page is in a frame!

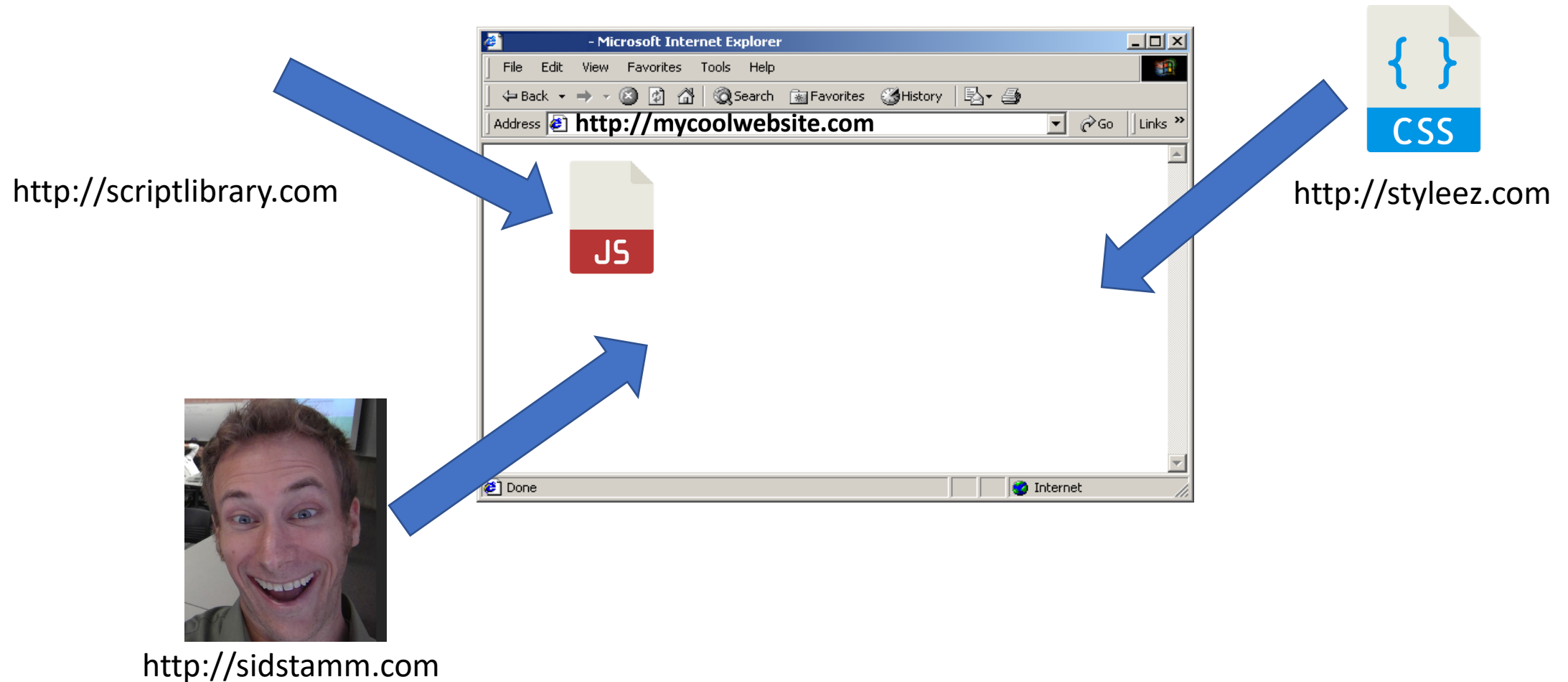
# Origin, Referer, etc

- Same Origin Policy isn't gonna save you
- No matter where stuff comes from

# Origin vs Principal

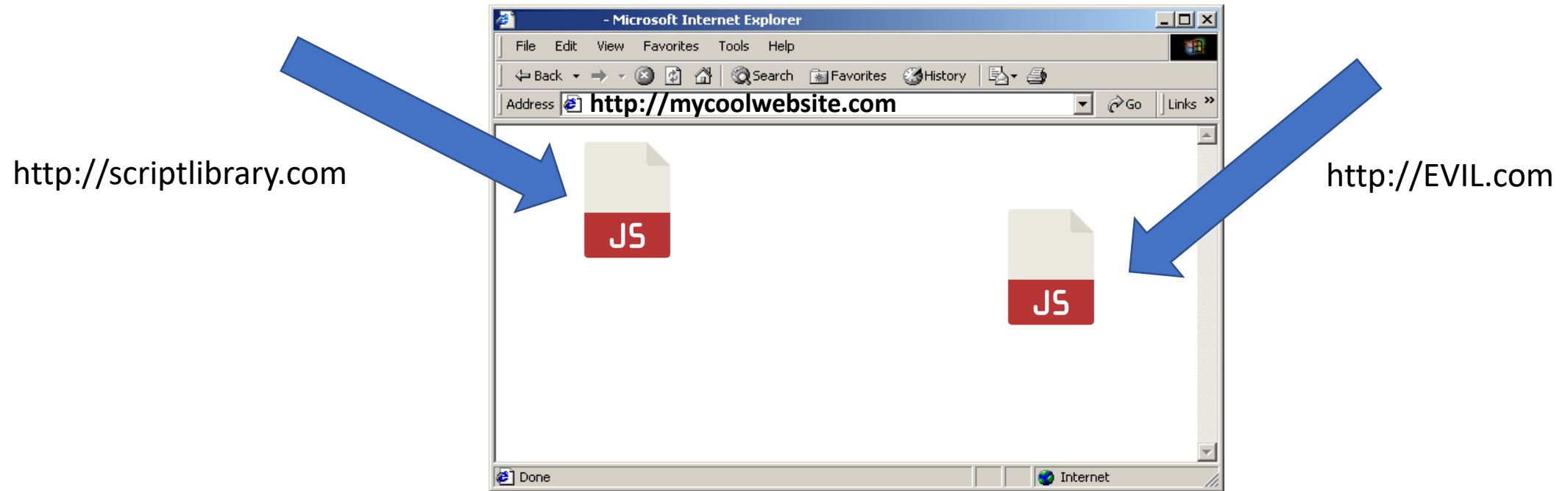


# Origin vs Principal



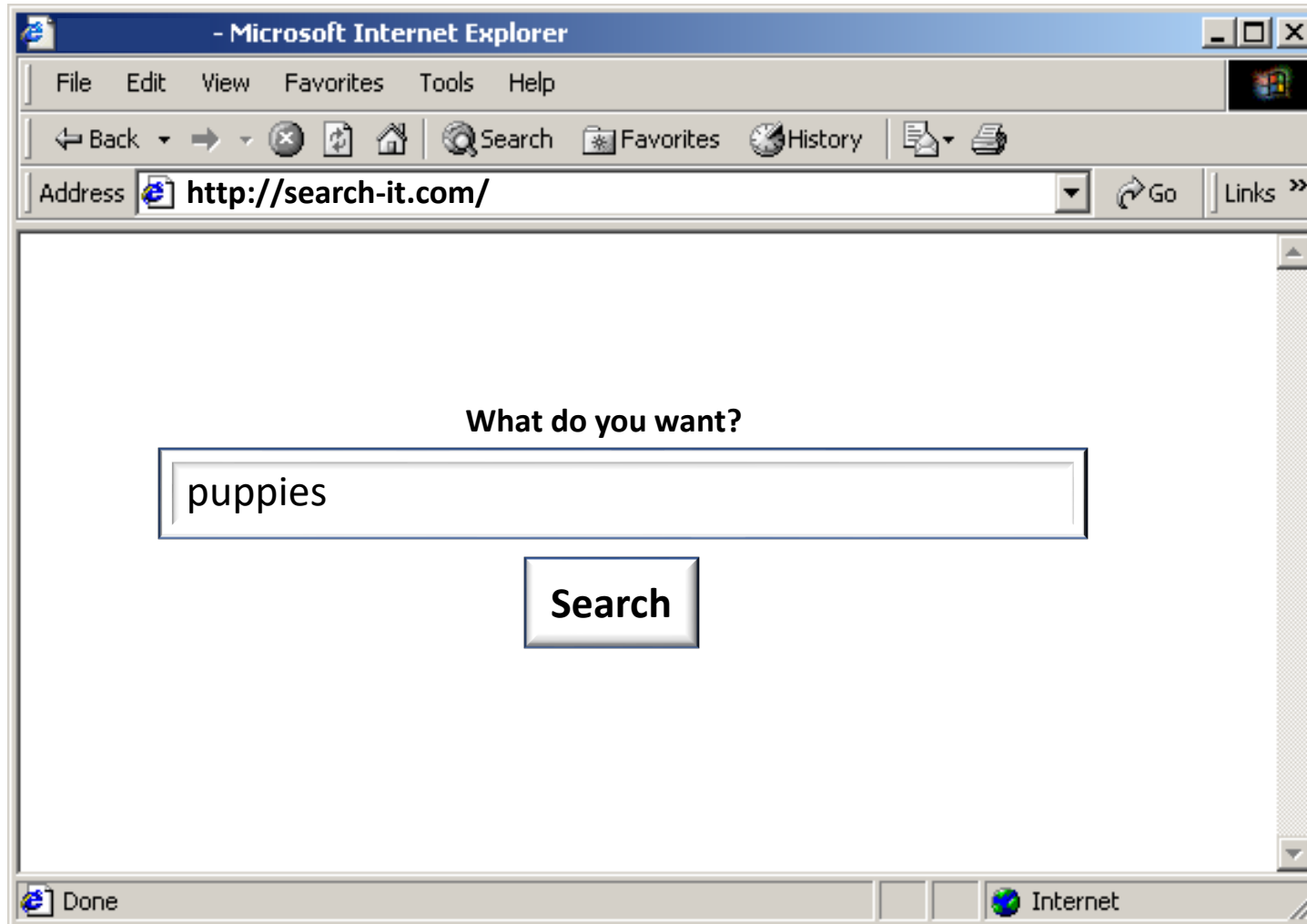


# Origin Laundering

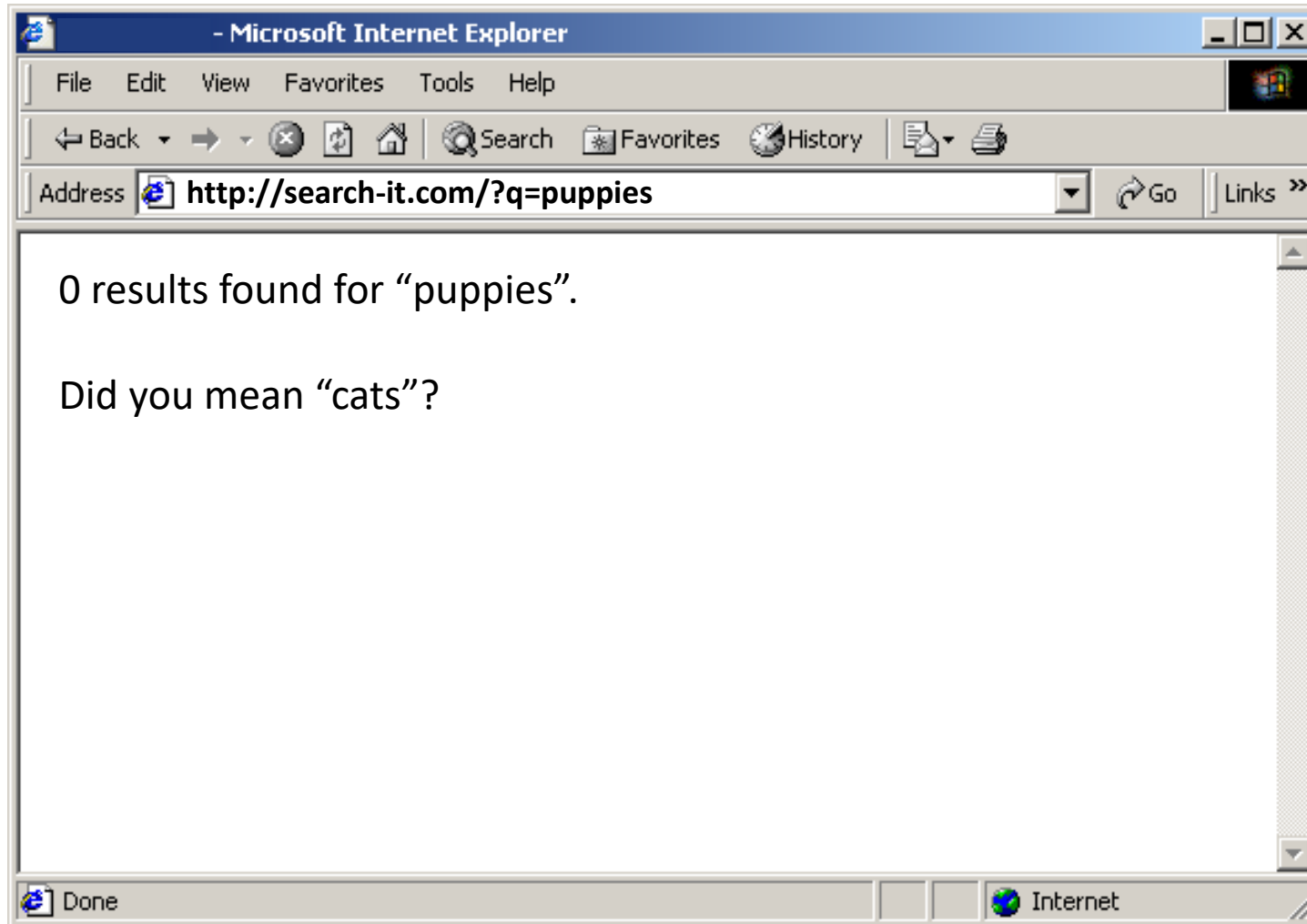


# Cross-Site Scripting (XSS)

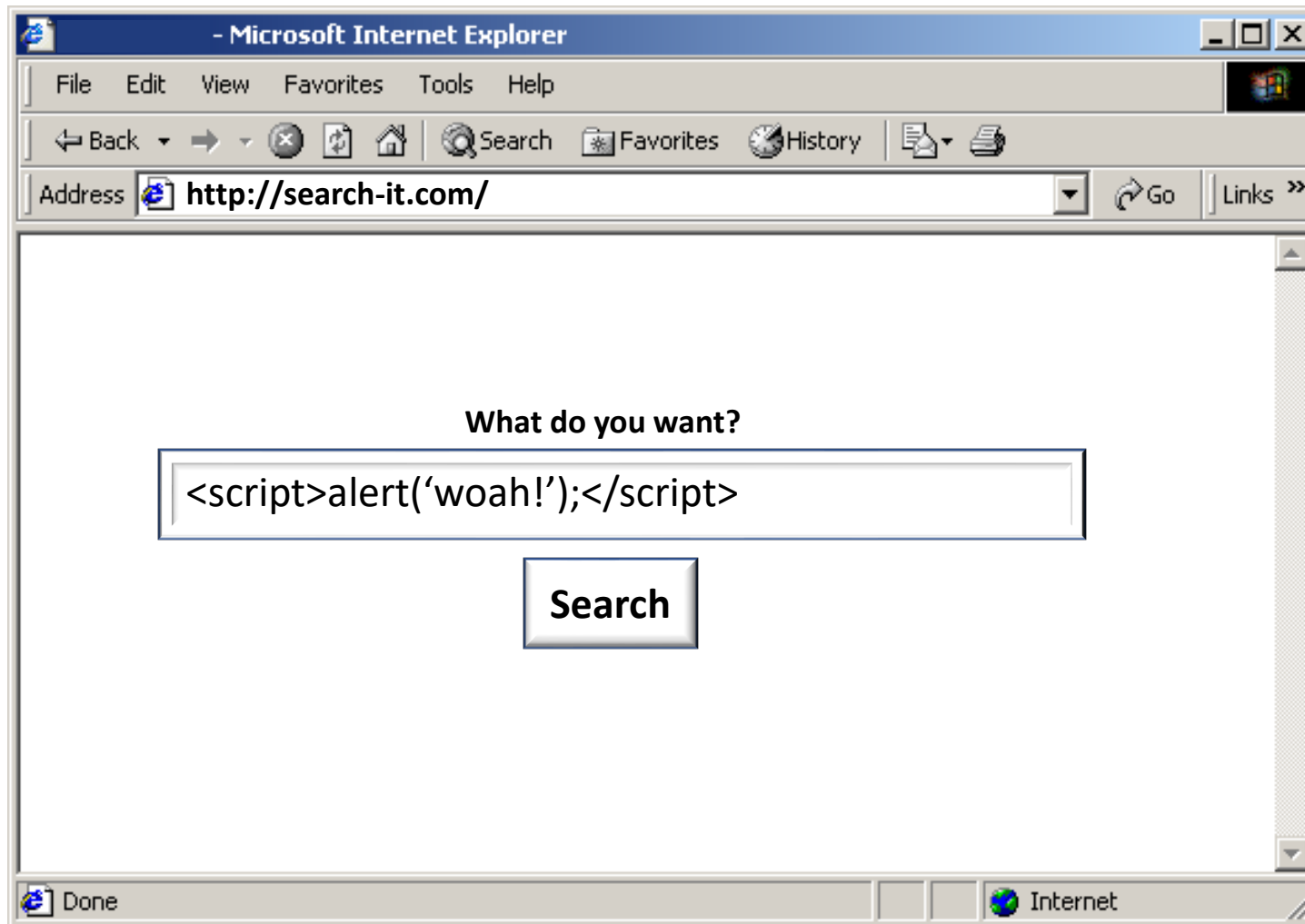
# Cross-Site Scripting (unwanted injection)



# Cross-Site Scripting (unwanted injection)

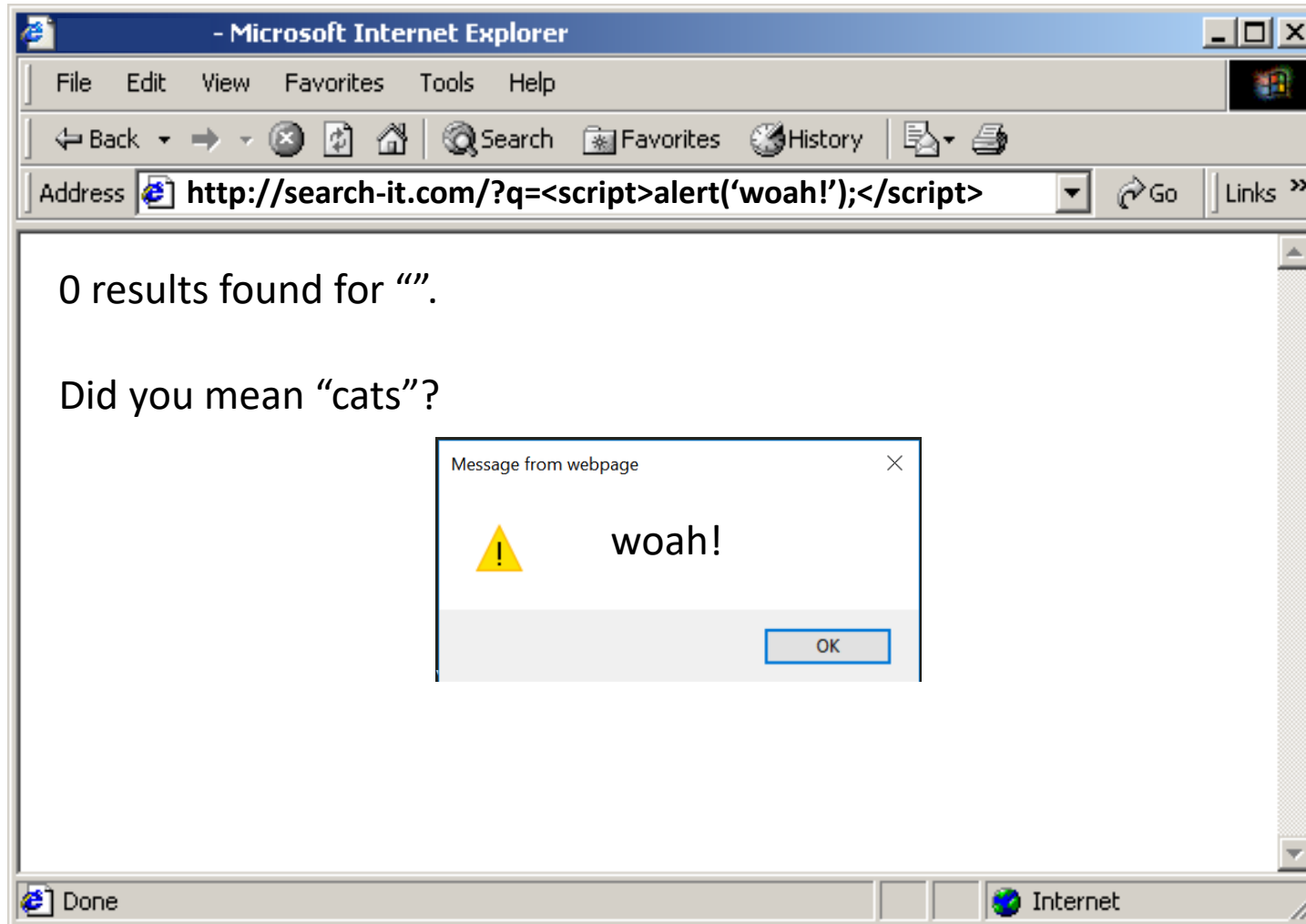


# Cross-Site Scripting (unwanted injection)

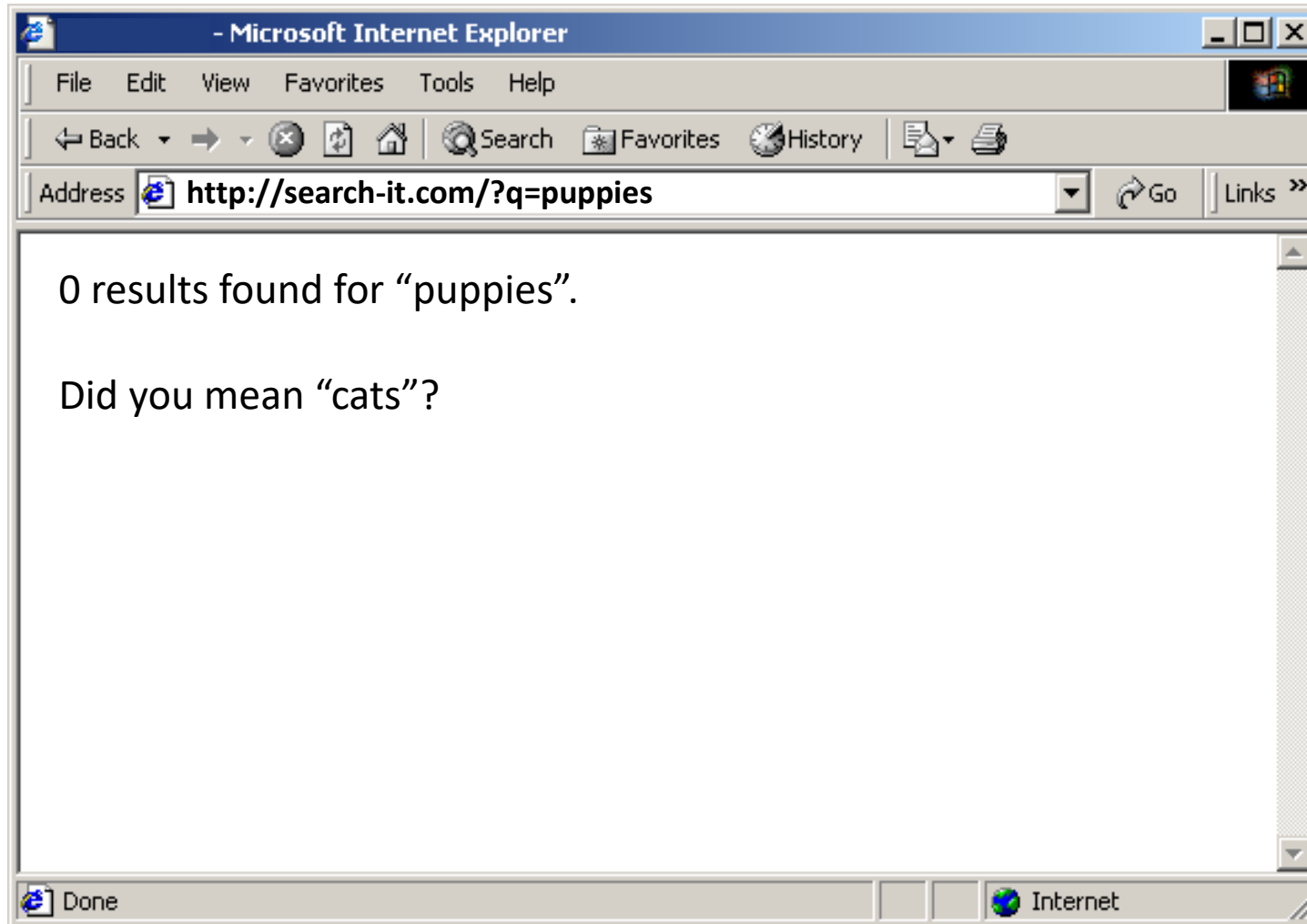




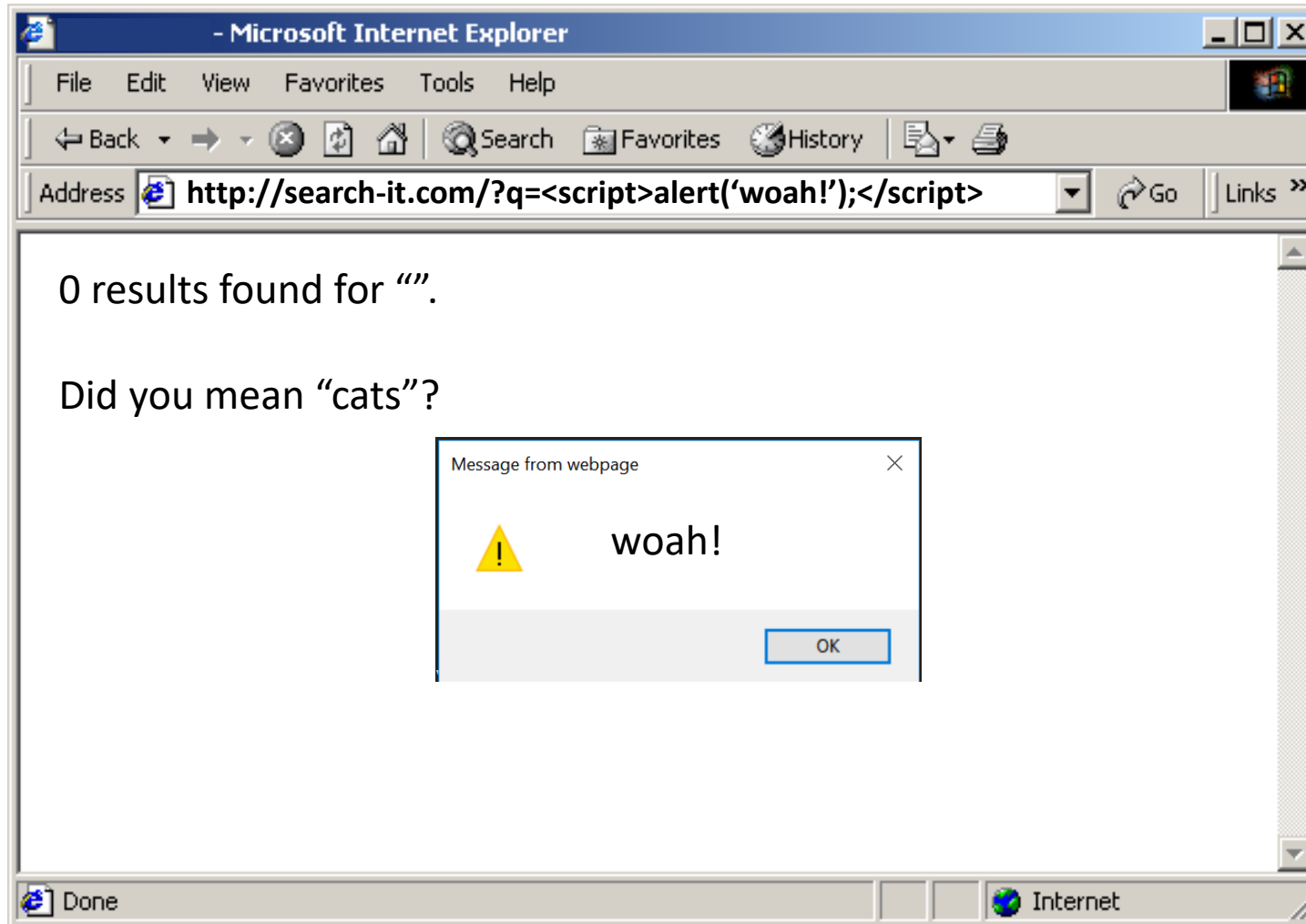
# Cross-Site Scripting (Reflected)



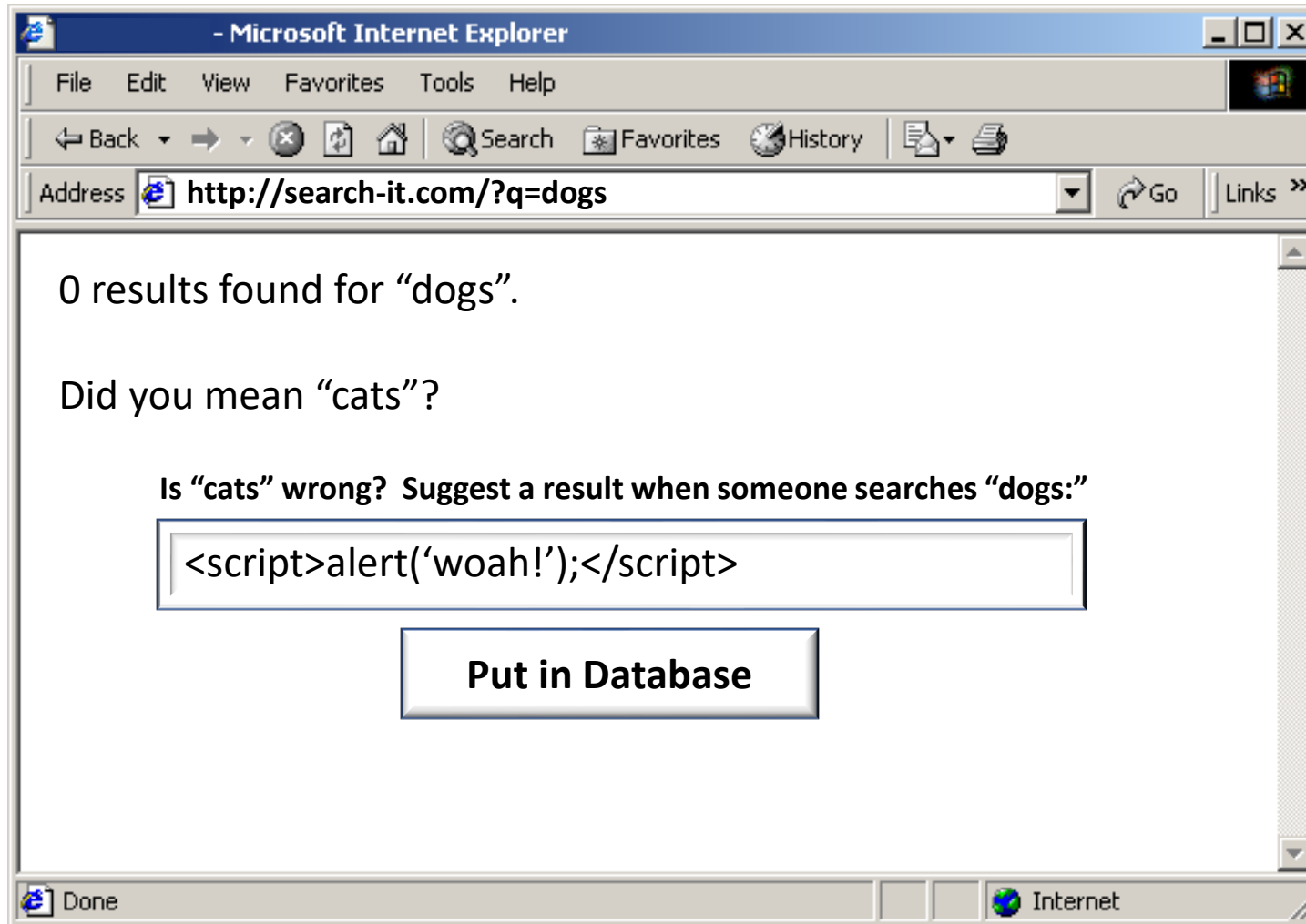
# Cross-Site Scripting (Reflected)



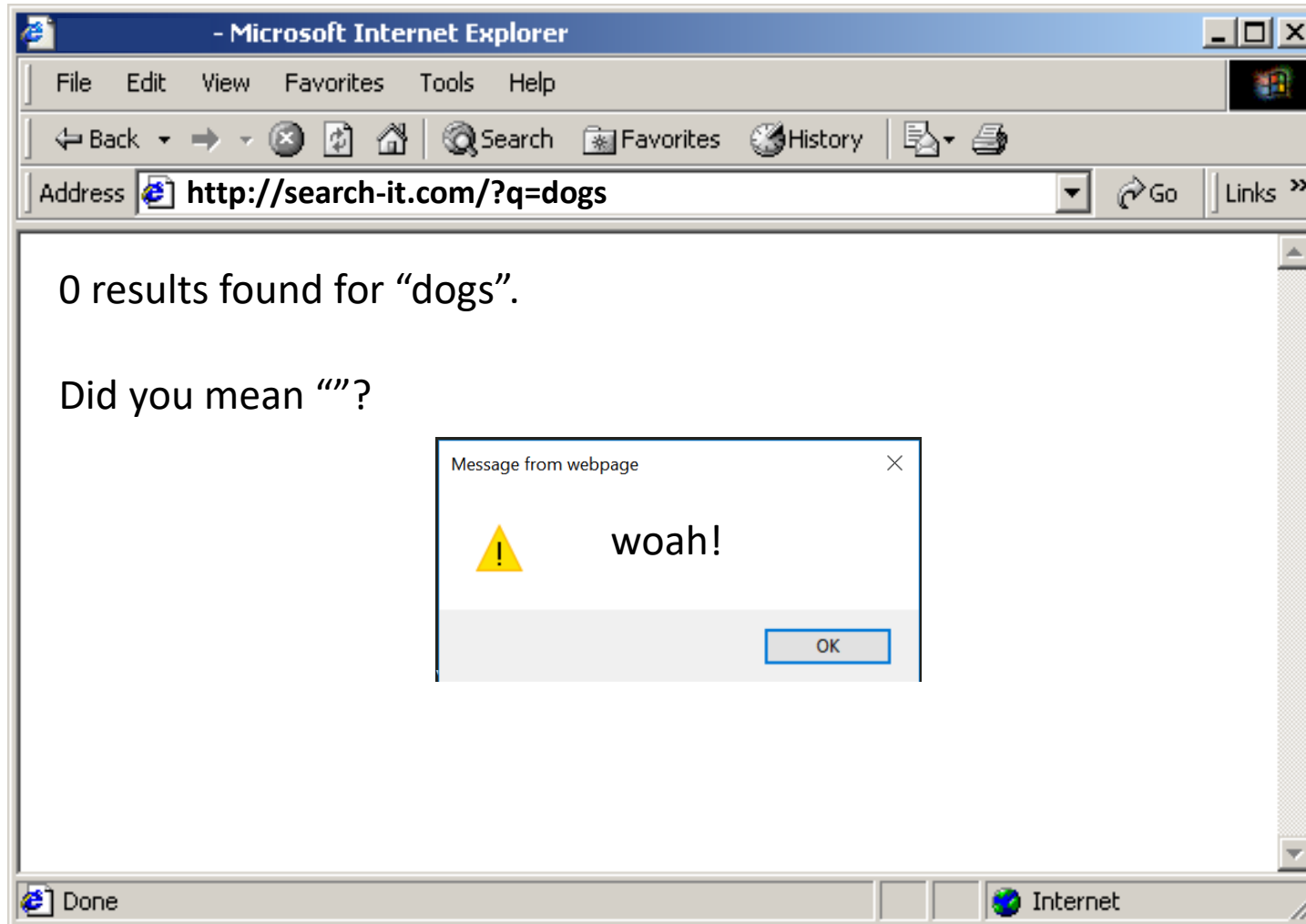
# Cross-Site Scripting (Reflected)



# Cross-Site Scripting (unwanted injection)



# Cross-Site Scripting (Persistent)





# Why is this bad?

- Mostly: Information Theft
- Also: Resource (computation/memory) Theft

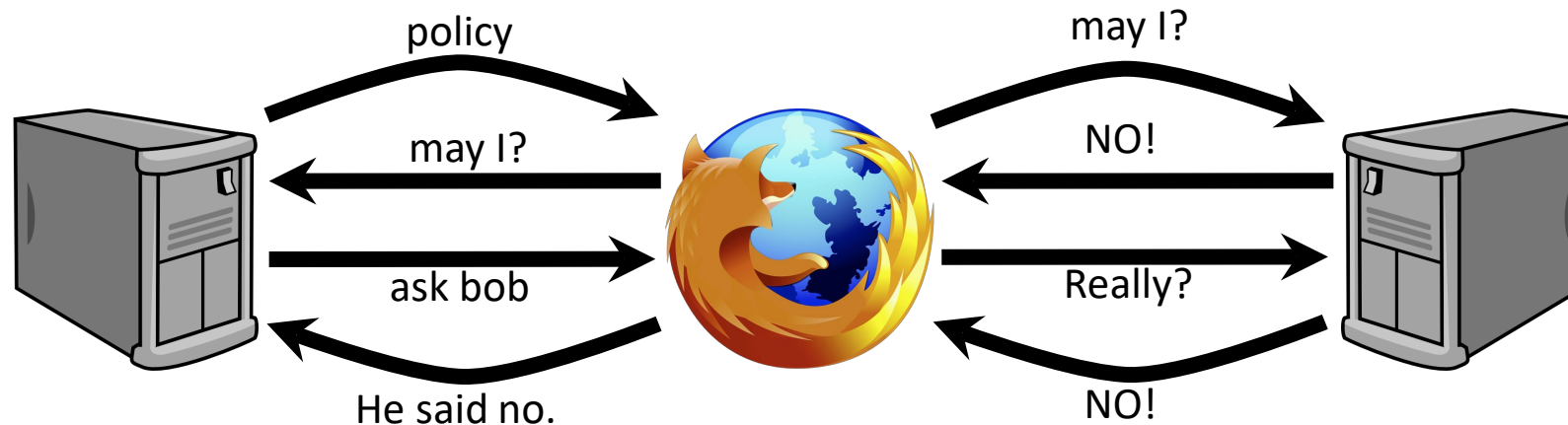
So what do we do?

# Provenance and Control

- Filtering Scripts (reflected) is hard!
- But we do it anyway... poorly.

# Provenance and Control

- Filtering Scripts (reflected) is hard
- Mutual approval for remote sources (library import) is expensive



# Provenance and Control

- Filtering Scripts (reflected) is hard
- Mutual approval for remote sources (library import) is expensive
- How about a finer-grained set of rules?

# Provenance and Control

- Filtering Scripts (reflected) is hard
- Mutual approval for remote sources (library import) is expensive
- How about a finer-grained set of rules?

Content Security Policy  
To the rescue!



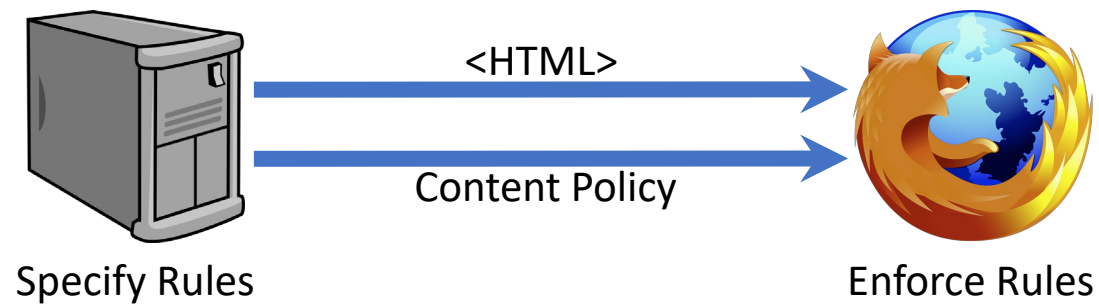
# What is “Content Security Policy”?

- Document “Good” behavior...
- Suppress the “Bad”



# How?

- Content Rules & Regulations
- Specify a “Normal Behavior” Policy
- Catch and Block Violations

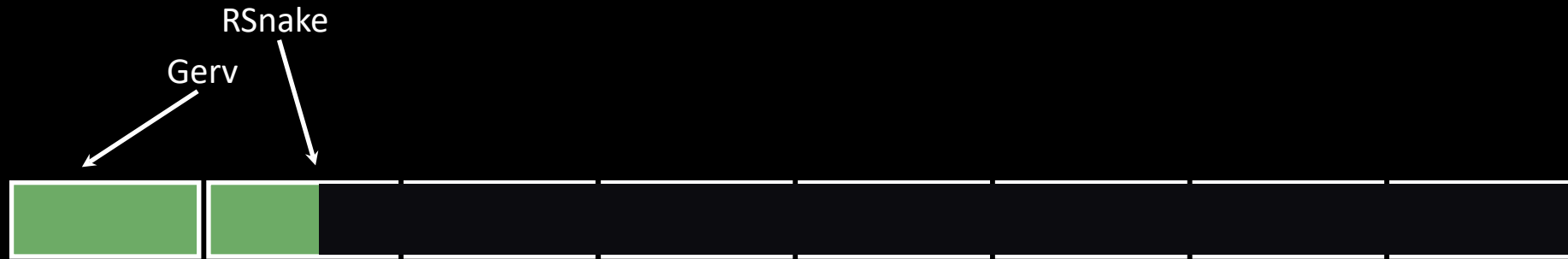


# The dark history of CSP



- April 2005  
Gervase Markham  
<http://www.gerv.net/security/content-restrictions/>

# CSP



- June 2006  
Robert “RSnake” Hansen  
<http://ha.ckers.org/blog/20060601/content-restrictions-and-xss/>

# CSP



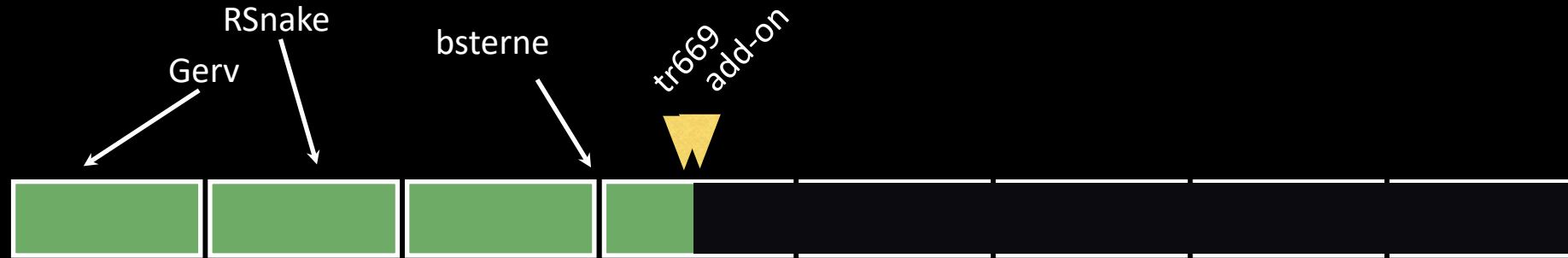
- August 2007  
RSnake again  
<http://ha.ckers.org/blog/20070811/content-restrictions-a-call-for-input/>

# CSP



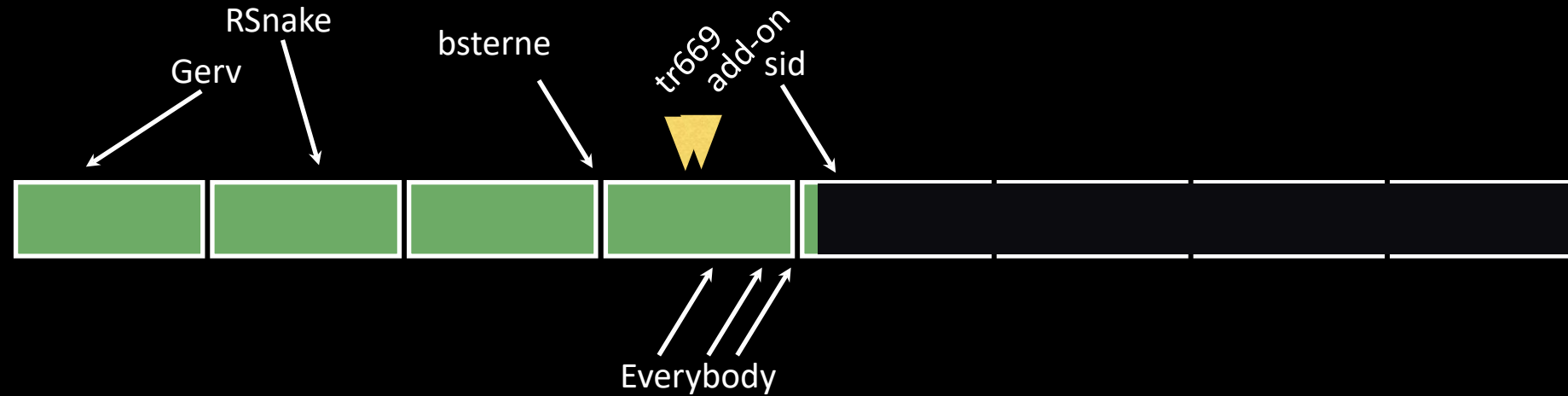
- Dec 2007  
Brandon Sterne (Site Security Policy)  
<http://people.mozilla.org/~bsterne/content-security-policy/details-0.1.html>

# CSP



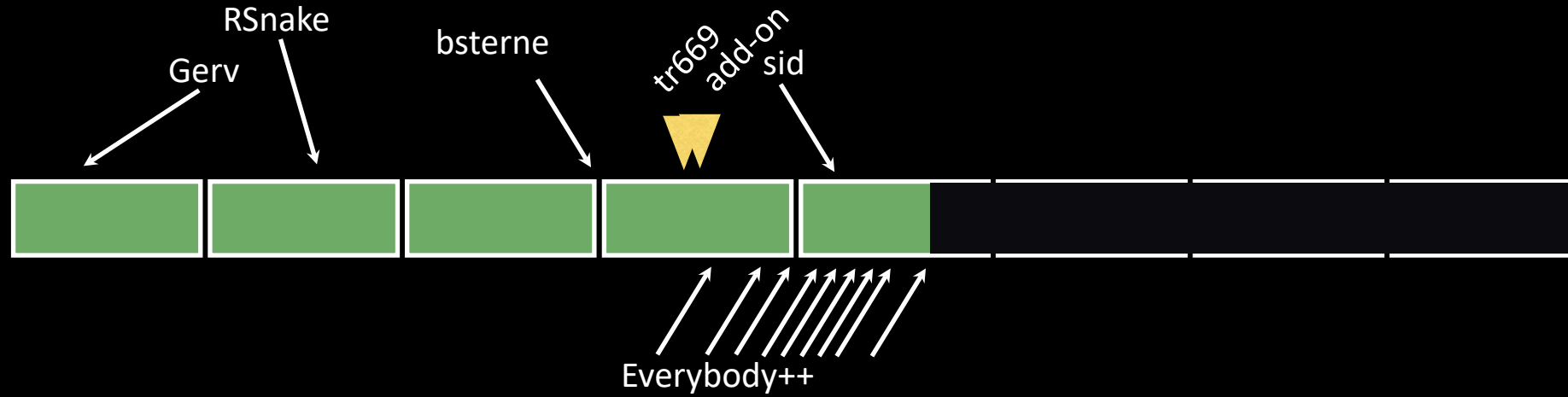
- July 2008  
IUCS TR669: Immigration Control for Web Pages
- August 2008  
Content Security Policy Add-On

# CSP



- Jan 2009  
Detailed Specification  
<https://wiki.mozilla.org/Security/CSP/Spec>

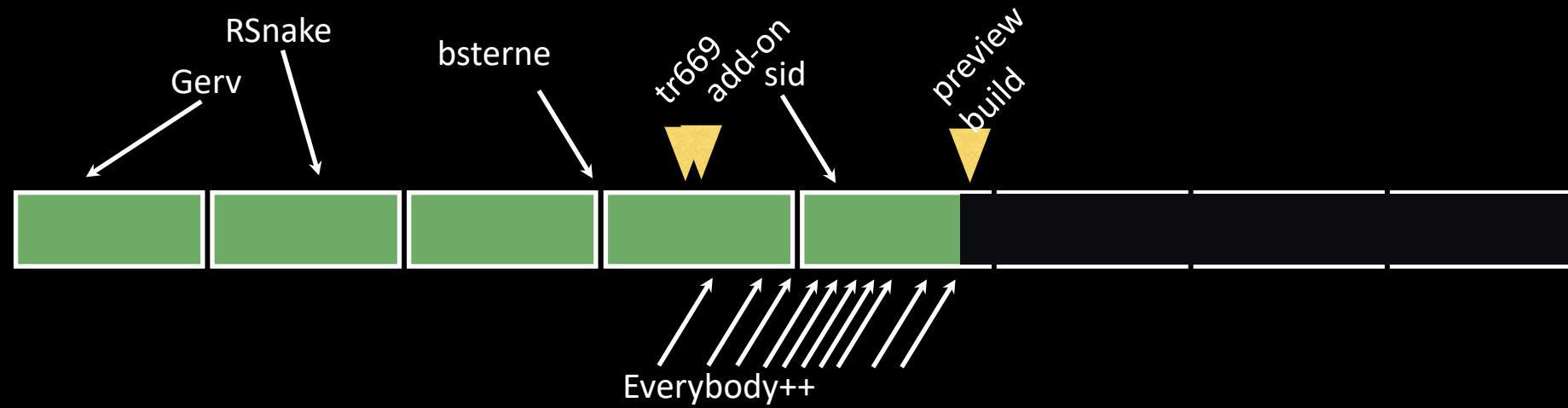
# CSP



- August 2009  
Discussion frenzy cools down

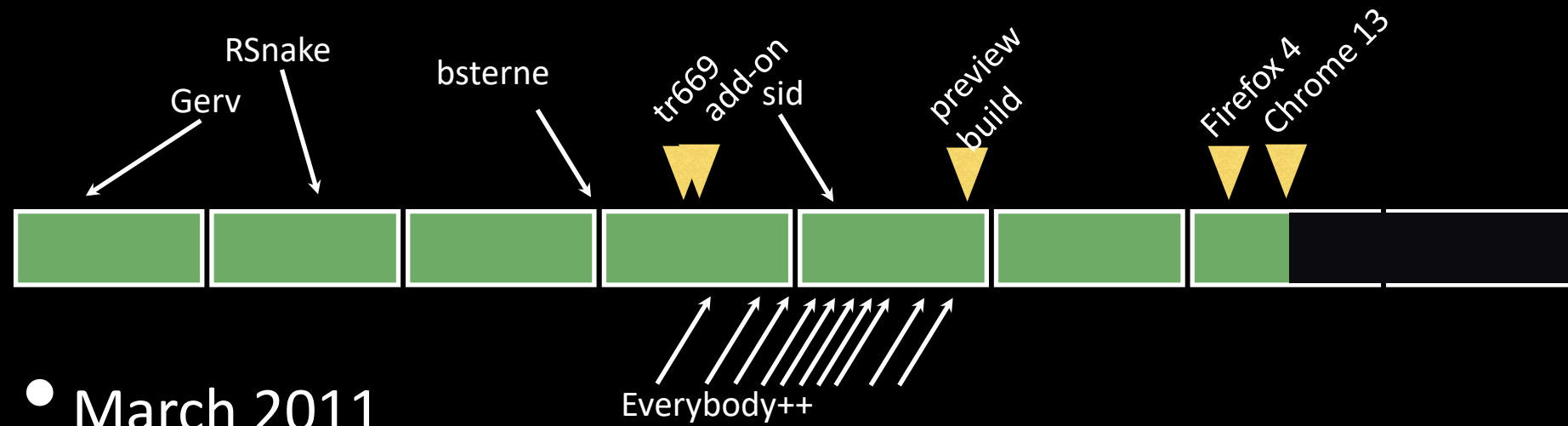


# CSP



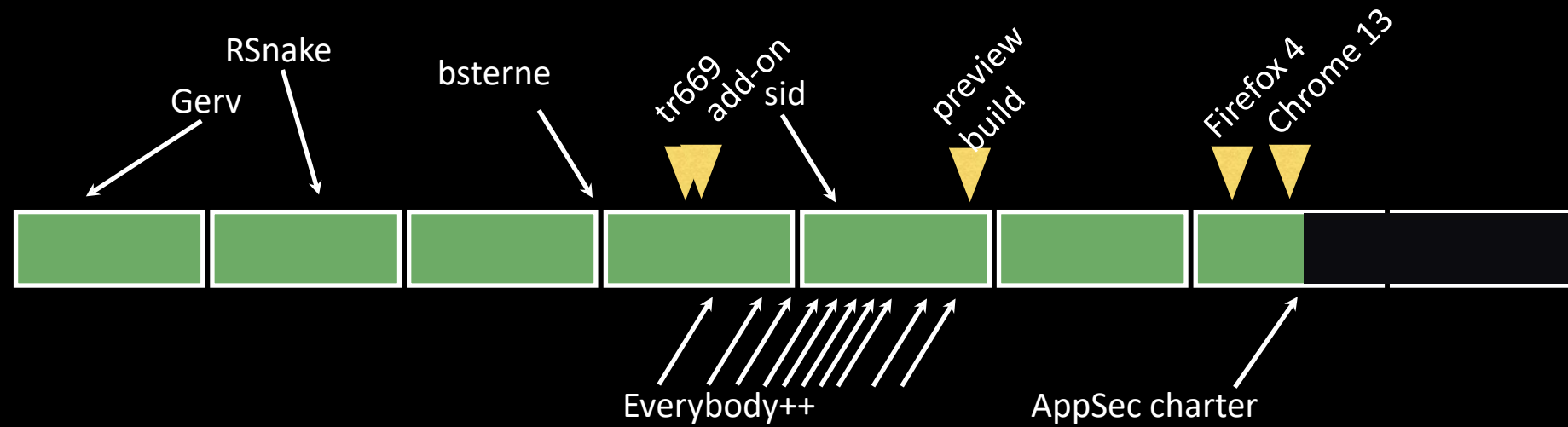
- October 2009  
First preview implementation

# CSP



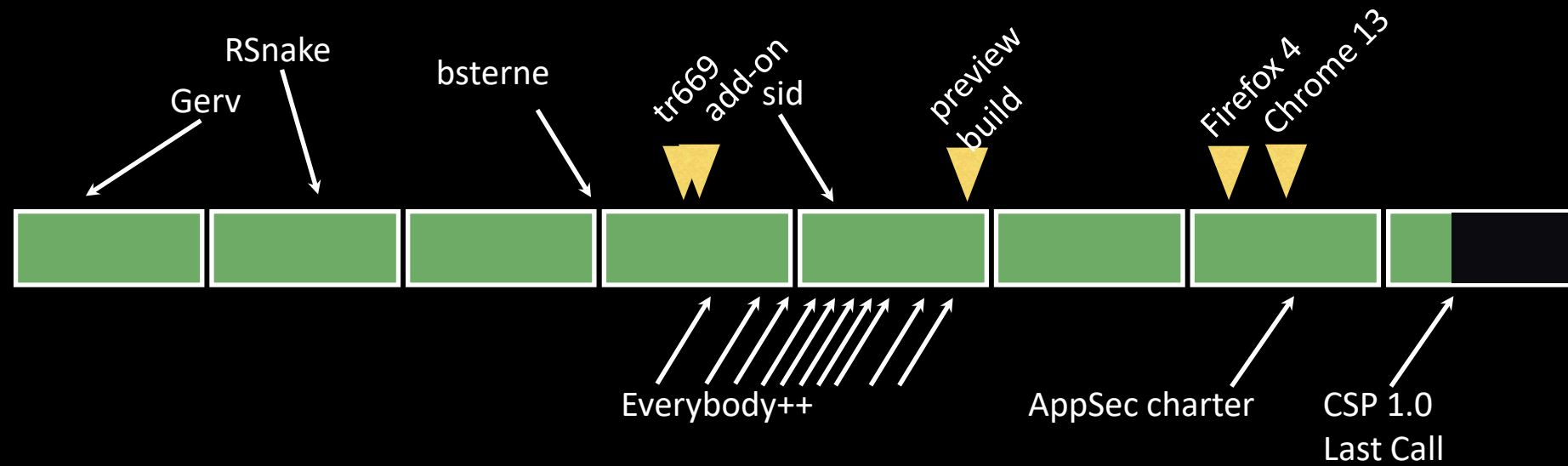
- March 2011  
Firefox 4 Launch
- August 2011  
Chrome ships CSP

# CSP



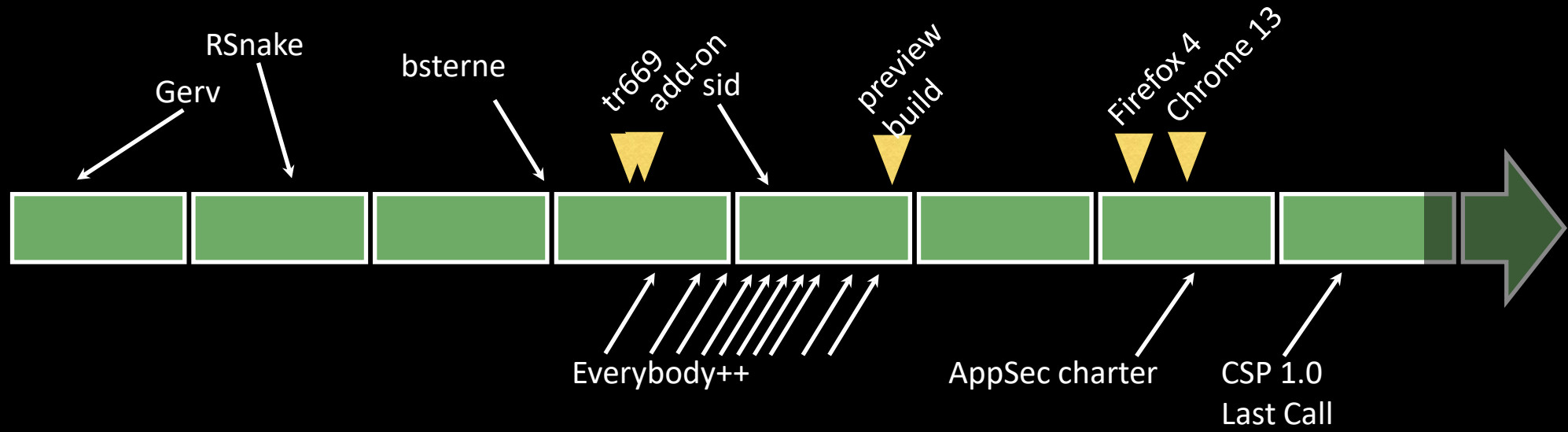
- August 2011  
W3C AppSec Working Group chartered

# CSP



- June 2012  
CSP 1.0 Draft Last Call (stable spec)

# CSP



CSP in a small amount of detail

# Step 1: Smooth Edges

- Scripts served in files (not inline)
  - “javascript:” URIs
  - <tag on\*=...> event registration
  - text nodes in <script> tags
- Establish Code / Data Separation
  - eval(“foo”) and friends

## Step 2: Content Restrictions

- Block requests for all resources  
... unless explicitly allowed by a policy!



# Step 3: Safety

- Sites only request explicitly allowed resources
- Injected inline scripts don't run
- Content homogenization (mixed content control)
- Violation reports = early alert

# Google's CSP

```
object-src 'none';  
base-uri 'self';  
script-src 'nonce-6VkEPITNuNAeaYj1aWDFig'  
'strict-dynamic' 'report-sample' 'unsafe-eval'  
'unsafe-inline' https: http:;  
report-uri https://csp.withgoogle.com/csp/gws/fff
```

# Twitter's CSP

- connect-src 'self' blob: https://\*.pscp.tv https://\*.video.pscp.tv https://\*.twimg.com https://api.twitter.com https://api-stream.twitter.com https://ads-api.twitter.com https://aa.twitter.com https://caps.twitter.com https://pay.twitter.com https://sentry.io https://ton.twitter.com https://twitter.com https://upload.twitter.com https://www.google-analytics.com https://accounts.google.com/gsi/status https://accounts.google.com/gsi/log https://app.link https://api2.branch.io https://bnc.lt wss://\*.pscp.tv https://vmap.snappytv.com https://vmapstage.snappytv.com https://vmaprel.snappytv.com https://vmap.grabyo.com https://dhdsnappytv-vh.akamaihd.net https://pdhdsnappytv-vh.akamaihd.net https://mdhdsnappytv-vh.akamaihd.net https://mdhdsnappytv-vh.akamaihd.net https://mpdhdsnappytv-vh.akamaihd.net https://mmdhdsnappytv-vh.akamaihd.net https://mdhdsnappytv-vh.akamaihd.net https://mpdhdsnappytv-vh.akamaihd.net https://mmdhdsnappytv-vh.akamaihd.net https://dwo3ckksxlb0v.cloudfront.net https://media.riffsy.com https://\*.giphy.com https://media.tenor.com https://c.tenor.com ; default-src 'self'; form-action 'self' https://twitter.com https://\*.twitter.com; font-src 'self' https://\*.twimg.com; frame-src 'self' https://twitter.com https://mobile.twitter.com https://pay.twitter.com https://cards-frame.twitter.com https://accounts.google.com/ https://client-api.arkoselabs.com/ https://iframe.arkoselabs.com/ https://recaptcha.net/recaptcha/ https://www.google.com/recaptcha/ https://www.gstatic.com/recaptcha/; img-src 'self' blob: data: https://\*.cdn.twitter.com https://ton.twitter.com https://\*.twimg.com https://analytics.twitter.com https://cm.g.doubleclick.net https://www.google-analytics.com https://maps.googleapis.com https://www.periscope.tv https://www.pscp.tv https://media.riffsy.com https://\*.giphy.com https://media.tenor.com https://c.tenor.com https://\*.pscp.tv https://\*.periscope.tv https://prod-periscope-profile.s3-us-west-2.amazonaws.com https://platform-lookaside.fbsbx.com https://scontent.xx.fbcdn.net https://scontent-sea1-1.xx.fbcdn.net https://\*.googleusercontent.com; manifest-src 'self'; media-src 'self' blob: https://twitter.com https://\*.twimg.com https://\*.vine.co https://\*.pscp.tv https://\*.video.pscp.tv https://dhdsnappytv-vh.akamaihd.net https://pdhdsnappytv-vh.akamaihd.net https://mdhdsnappytv-vh.akamaihd.net https://mdhdsnappytv-vh.akamaihd.net https://mpdhdsnappytv-vh.akamaihd.net https://mmdhdsnappytv-vh.akamaihd.net https://mdhdsnappytv-vh.akamaihd.net https://mpdhdsnappytv-vh.akamaihd.net https://mmdhdsnappytv-vh.akamaihd.net https://dwo3ckksxlb0v.cloudfront.net; object-src 'none'; script-src 'self' 'unsafe-inline' https://\*.twimg.com https://recaptcha.net/recaptcha/ https://www.google.com/recaptcha/ https://www.gstatic.com/recaptcha/ https://client-api.arkoselabs.com/ https://www.google-analytics.com https://twitter.com https://app.link https://accounts.google.com/gsi/client https://appleid.cdn-apple.com/appleauth/static/jsapi/appleid/1/en\_US/appleid.auth.js 'nonce-MWViM2RhODItMDJiYS00M2Y0LWJlZTAtYWVmZTdkYjU5MWE2'; style-src 'self' 'unsafe-inline' https://accounts.google.com/gsi/style https://\*.twimg.com; worker-src 'self' blob:; report-uri https://twitter.com/i/csp\_report?a=O5RXE%3D%3D%3D&ro=false

What's Next?

# Can we automatically apply CSP?

- Kind of.
- Lots of “inline” script usage.
- TONS use scripts from other origins – hard to predict.

[http://research.sidstamm.com/papers/csp\\_icissp\\_2016.pdf](http://research.sidstamm.com/papers/csp_icissp_2016.pdf)

*"Injecting CSP for Fun and Security", Christoph Kerschbaumer, Sid Stamm, and Stefan Brunthaler. 2nd [International Conference on Information Systems Security and Privacy \(ICISSP\)](#)' February 2016. Rome, Italy.*

# It's still a problem?

- Yeah, turns out web apps are complicated. 😞
- Develop CSP rules as you develop a web app!
- Look for weak CSPs to find possible security flaws.
- Security features like CSP are not a substitute for good software development practice

Thank you

---

**ROSE-HULMAN**

**ROSE-HULMAN**