

# CS 253: Web Security

# Origins of this course

- CS 241: Secure Web Programming
  - Last taught in 2011 by Dan Boneh and John Mitchell
  - My favorite class at Stanford
  - Inspired me to start looking for vulnerabilities
- I wanted to bring back the course – that's what CS 253 is!

# Story time

- `localStorage.username = 'feross' // put`
- `console.log(localStorage.username) // get`
- "5 MB per origin"

# Every origin can store 5 MB

1.filldisk.com

2.filldisk.com

3.filldisk.com

4.filldisk.com

5.filldisk.com

6.filldisk.com

7.filldisk.com

... and so on ...

Fill up your hard disk with ju x


www.filldisk.com

# The Joys of HTML5

## Introducing the new HTML5 Hard Disk Filler™ API

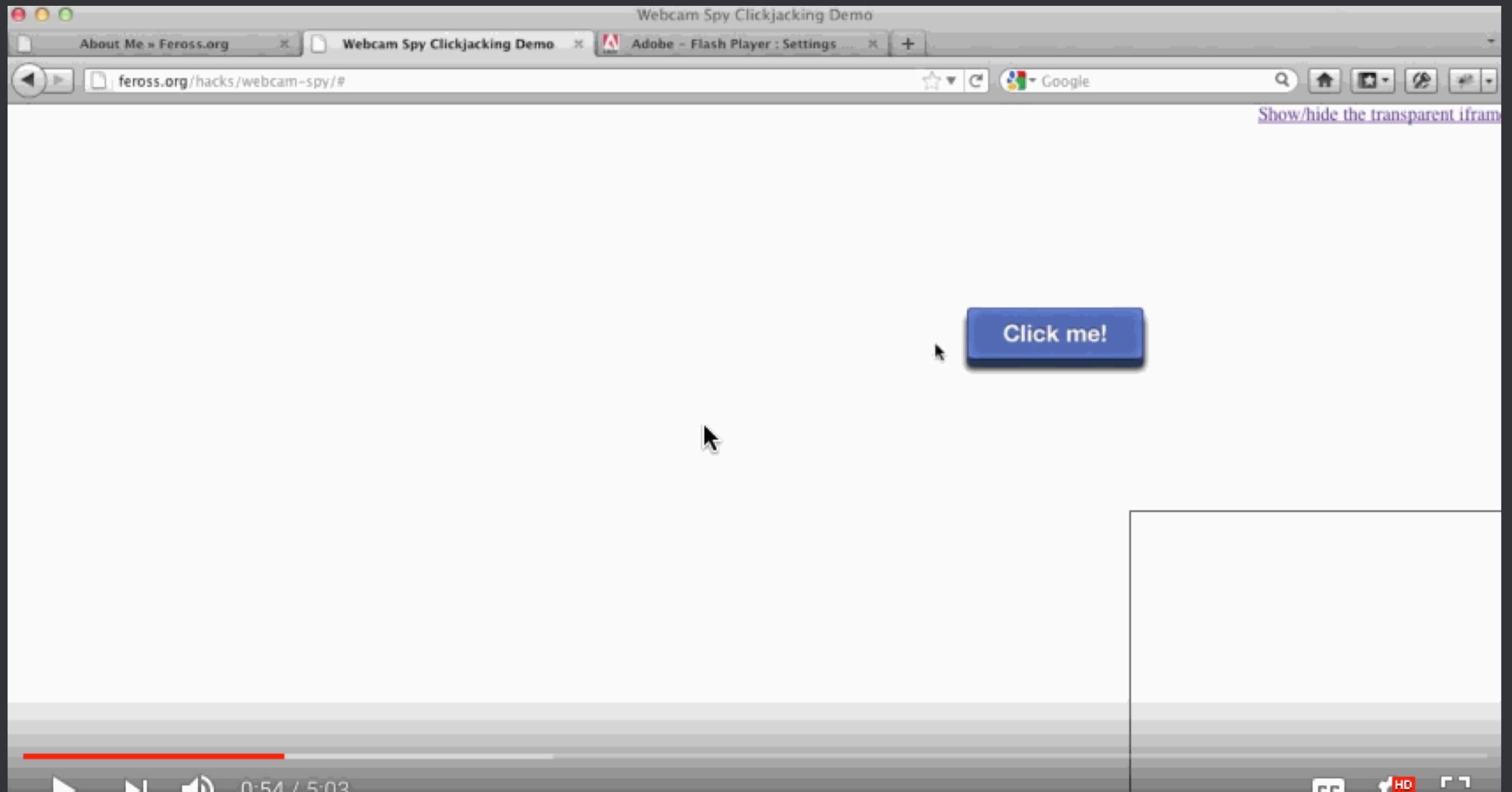
Oh hai there... Filling your hard disk with lots of cats...

**Used 15 MB of disk space!**



Stop the madness! (gives your disk space back)

Works in Chrome, Safari (iOS and desktop), Opera and IE. Firefox is immune to this hackery.



{\* SECURITY \*}

# Bug in Flash Player allowed Mac webcam spying

Adobe issues patch for 'clickjacking' hole

Dan Goodin

Thu 20 Oct 2011 // 18:22 UTC

13 

**UPDATED**

Engineers on Thursday patched a hole in Adobe's ubiquitous Flash Player that allowed website operators to silently eavesdrop on visitors' webcam and microphone feeds without permission.

To be attacked, visitors needed to do no more than visit a malicious website and click on a handful of buttons like the ones in this [live demonstration](#). Without warning, the visitor's camera and microphone were activated and the video and audio intercepted. The attack closely resembled a [separate Flash-based attack on webcams](#) from 2008 using a class of exploit known as [clickjacking](#).

Adobe said on Thursday it was planning to fix the vulnerability, which stems from flaws in the [Flash Player Settings Manager](#). The panel, which is used to designate which sites may access feeds from an enduser's camera and mic, is delivered in the SWF format used by Flash. Feross Aboukhadijeh, a computer science student at Stanford University, discovered he could embed the SWF file as an invisible iframe and superimpose misleading graphics on top that tricked visitors into making changes to the underlying privacy settings.

# Goal #1

# The attacker mindset



# Goal #2

# The defender mindset

# Extra credit policy

- Anyone who finds a web security vulnerability (on any site) during the quarter will receive extra credit (1 to 50 points)
- **YOU MUST USE RESPONSIBLE DISCLOSURE**
  - You are responsible for your own actions
  - If you are unsure, come speak with us
  - If Stanford web app, must use Stanford Bug Bounty program
  - Do not attack servers you do not own, do not destroy data

# Lots of students have found vulnerabilities!

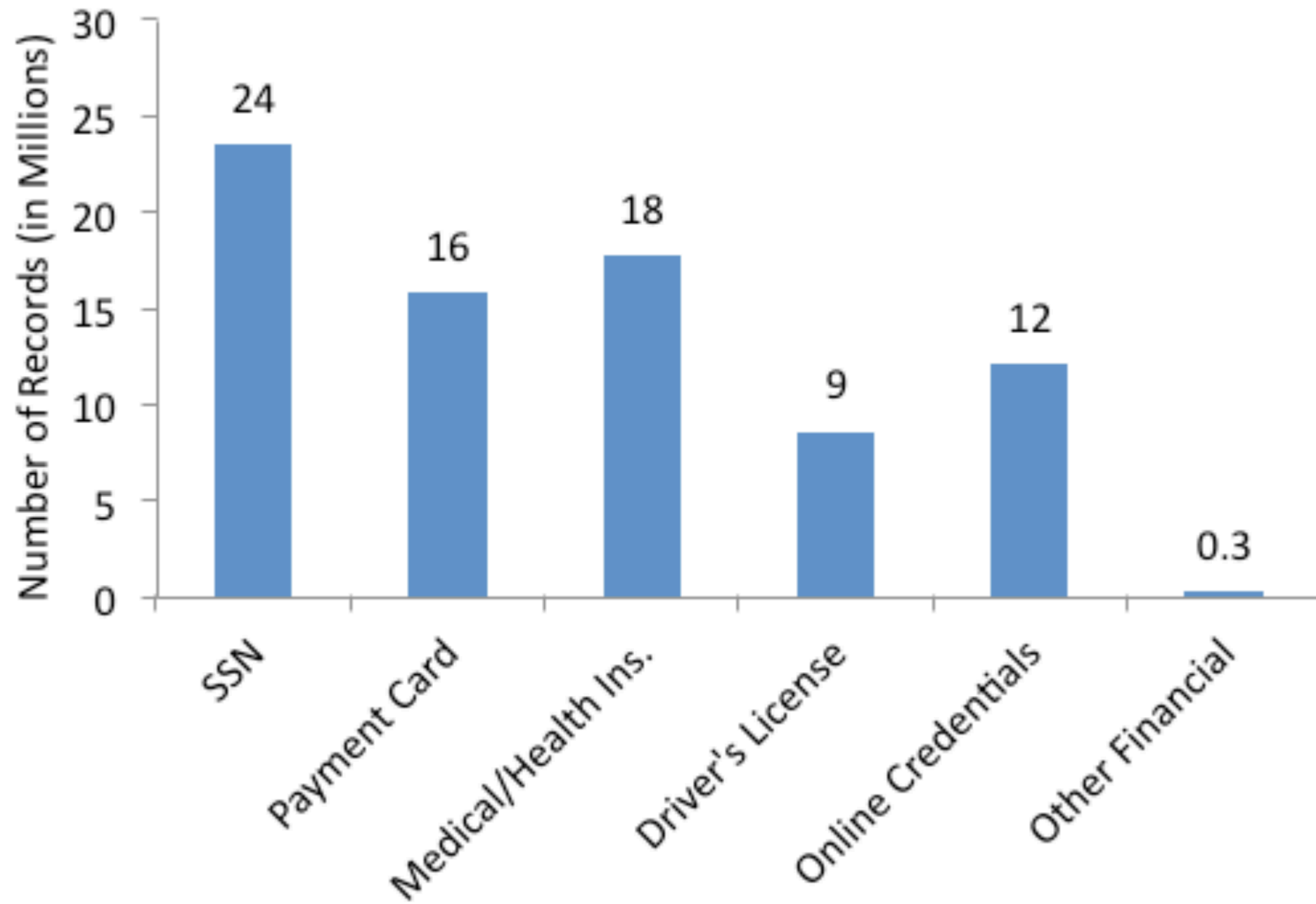
- Axxess: Cross-site scripting (\$350 bounty paid)
- CS 157 site: Cross-site scripting (change grades)
- Stripe interview site: Cross-site scripting and code injection (uncover test cases)
- React: DNS rebinding (leak development sites to network users)
- Google Search: Spammers tricked Google into thinking web spam hosted on stanford.edu
- New site: Paywall bypass

# Why is computer security hard?

- Lots of buggy code
- Social engineering is very effective
- There's money to be made by finding and exploiting vulnerable systems
  - Marketplace for vulnerabilities
  - Marketplace for owned machines / stolen data
  - Many methods to profit from owned machines / stolen data

# Why attack a computer system?

- Spam
  - Sent from legitimate IP address, less likely to be blocked
- Denial of service
  - Attack competitors, or seek ransom
- Infect visiting users with malware
  - Infect one server, use it to infect hundreds of thousands of clients
- Data theft
  - Steal credentials, credit card numbers, intellectual property



# Why attack a computer system in 2021?

- RANSOMWARE
- Mine cryptocurrency
- Geopolitical motivations

# What is web security?

- Browser security
  - e.g. Same Origin Policy – Isolate sites from each other, while running in the same browser



The image shows a browser window with the Wikipedia homepage. The browser's address bar shows 'wikipedia.org'. The page features the Wikipedia logo and the text 'The Free Encyclopedia'. Below this, there are several language options, each with a count of articles or items. A central graphic of a globe made of puzzle pieces is also visible.

Language	Count
English	5 935 000+ articles
Español	1 546 000+ artículos
日本語	1 169 000+ 記事
Русский	1 569 000+ статей
Italiano	1 554 000+ voci
Português	1 014 000+ artigos
Deutsch	2 345 000+ Artikel
Français	2 141 000+ articles
中文	1 074 000+ 條目
Polski	1 360 000+ haseł

javascript - Why does Google p X +

stackoverflow.com/questions/2669690/why-does-google-prepend-while1-to-their-json-responses

stackoverflow Products Customers Use cases Search... Log in Sign up

Home PUBLIC Stack Overflow Tags Users Jobs TEAMS What's this? First 10 Free

## Why does Google prepend while(1); to their JSON responses?

Asked 9 years, 5 months ago Active 6 months ago Viewed 509k times

3910



Why does Google prepend `while(1);` to their (private) JSON responses?

For example, here's a response while turning a calendar on and off in [Google Calendar](#):

```
while(1);[['u',[['smsSentFlag','false'],['hideInvitati
['remindOnRespondedEventsOnly','true'],
['hideInvitations_remindOnRespondedEventsOnly','fals
['Calendar ID stripped for privacy','false'],['smsVe
```

1762

I would assume this is to prevent people from doing an `eval()` on it, but all you'd really have to do is replace the `while` and then you'd be set. I would assume the `eval` prevention is to



# What is web security?

- Server-side security
  - Attackers can run arbitrary HTTP clients; can send anything to server

**curl**

```
-d '{"user": "Alice", "permission": "admin"}'  
-H "Content-Type: application/json"  
-X POST http://example.com/data
```

# What is web security?

- Client-side security
  - Prevent user from being attacked while using web app locally

# Stealing passwords from McDonald's users

## Reflected XSS through AngularJS sandbox bypass causes password exposure of McDonald users.

By abusing an insecure cryptographic storage vulnerability ([link](#)) and a reflected server cross-site-scripting vulnerability ([link](#)) it is possible to steal and decrypt the password from a McDonald's user. Besides that, other personal details like the user's name, address & contact details can be stolen too.

### Proof of Concept

#### Reflected XSS through AngularJS sandbox escape

McDonalds.com contains a search page which reflects the value of the search parameter (q) in the source of the page. So when we search on for example `*****-test-reflected-test-*****`, the response will look like this:

116,228 views | Aug 31, 2019, 03:41am

# Critical 'Backdoor Attack' Warning Issued For 60 Million WordPress Users



**Davey Winder** Senior Contributor ⓘ

[Cybersecurity](#)

*I report and analyse breaking cybersecurity and privacy stories*



# Creates a new admin user with these credentials:

- Username: `wpservices`
- Password: `wordpr3ss`



# What is web security?

- Protect the user
  - From social engineering
  - From trackers, private data being leaked

BIZ & IT —


# How a college student tricked 17k coders into running his sketchy script

Infecting military and government software engineers is easier than you may think.

DAN GOODIN - 6/14/2016, 7:10 AM



# Collusion for Chrome

 [nik.io](#)

This site is informed when you visit the following sites:

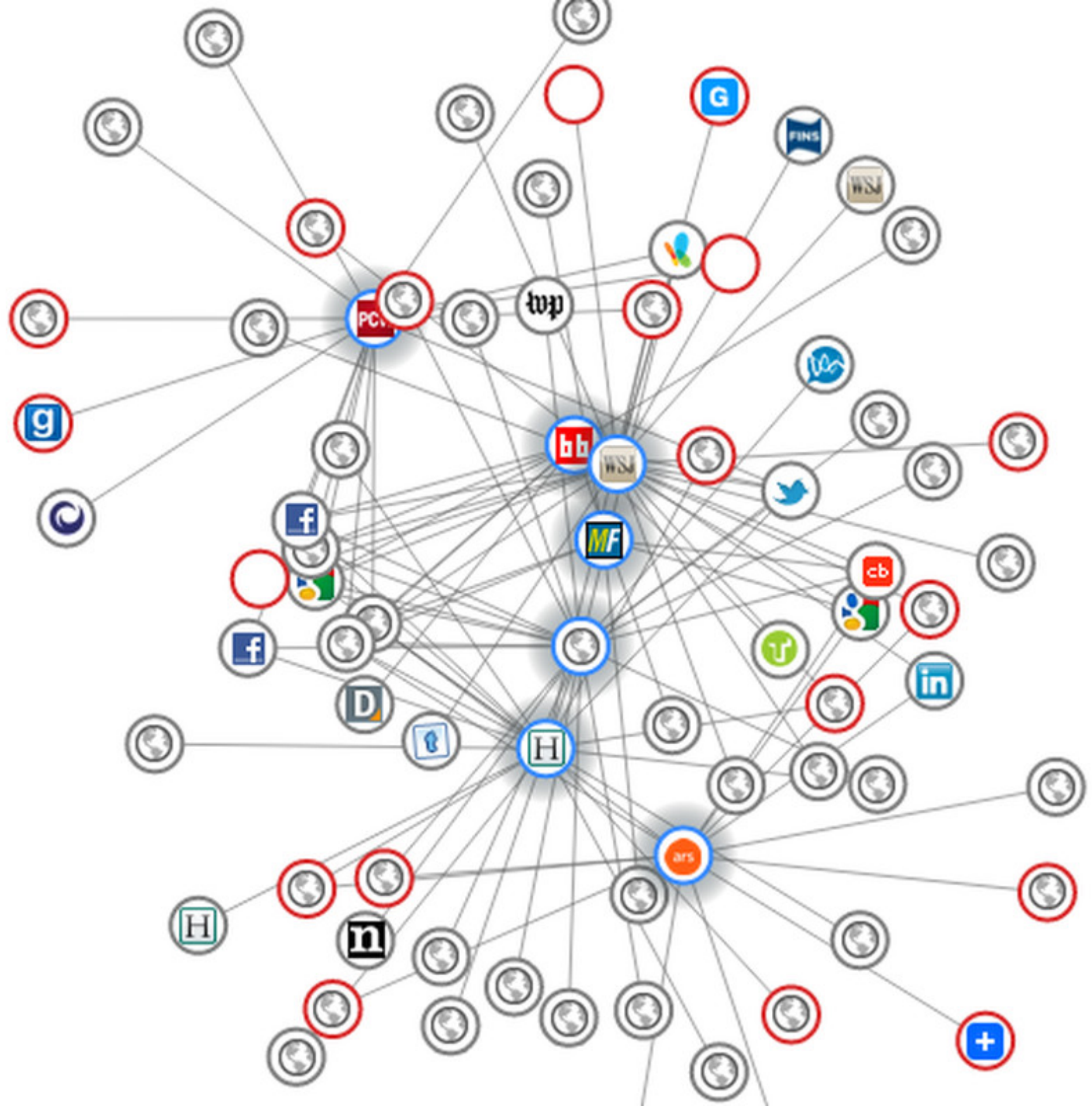
- [wsj.com](#)

Reset the graph

Hide the sidebar

Show the instructions

**Privacy:** We collect info about the sites you go to only to illustrate how they're connected. This info is stored locally on your computer and can be deleted anytime by resetting the graph or quitting your browser.



# Why is web security hard?

- Extremely ambitious goal – Run untrusted code securely
- Different sites interacting in the same tab ("mashups")
- Low-level features; hardware access
- Desire for high performance
- APIs were not designed from first principles; evolved
- Strict backwards compatibility requirements
  - "Don't break the web"

"Modern web applications are built on a tangle of technologies that have been developed over time and then haphazardly pieced together. Every piece of the web application stack, from HTTP requests to browser-side scripts, comes with important yet subtle security consequences. To keep users safe, it is essential for developers to confidently navigate this landscape."

– Michal Zalewski, from Tangled Web

# The browser has a seemingly impossible task

Sites – even malicious ones – can:

- Download content from **anywhere**
- Spawn worker processes
- Open sockets to a server, or even to another user's browser
- Display media in a huge number of formats
- Run custom code on the GPU
- Save/read data from the filesystem

# Differing visions for the web

- Simple document viewer?
- Powerful application platform?

# The web is robust

"It's is all too easy to criticize, lament, and create paranoid scenarios about the 'unsound security foundations' of the web. Truth is, all of that criticism is true, and yet the web has proven to be an incredibly robust platform."

– Ilya Grigorik, Google web performance engineer



# Goal #3

**Learn to architect  
secure systems**

# This course

- Part 1: Browser security model: Same origin policy
- Part 2: Client security: attacks, defense
- Part 3: Server security: attacks, defense
- Part 4: Authentication
- Part 5: Real world security, Writing secure code

# Administrative Stuff

- Website: `cs253.stanford.edu`
- ~5 assignments
- ~5 guest lectures
  - Brave, Google, GitHub, Dan Boneh, and more
- Use Ed for questions
- Share anonymous feedback (use form on website)
- Assignment 0 released tonight

**LEARN YOU THE NODE.JS FOR MUCH WIN!**

*Select an exercise and hit Enter to begin*

---

**» HELLO WORLD**

**» BABY STEPS**

**» MY FIRST I/O!**

**» MY FIRST ASYNC I/O!**

**» FILTERED LS**

**» MAKE IT MODULAR**

**» HTTP CLIENT**

**» HTTP COLLECT**

**» JUGGLING ASYNC**

**» TIME SERVER**

**» HTTP FILE SERVER**

**» HTTP UPPERCASERER**

**» HTTP JSON API SERVER**

---

**HELP**

**CHOOSE LANGUAGE**

**CREDITS**

**CHECK FOR UPDATE**

**EXIT**

# HTML

## **Introduction**

This article is a review of the book *Dietary Preferences of Penguins*, by Alice Jones and Bill Smith. Jones and Smith's controversial work makes two hard-to-swallow claims about penguins:

- First, that penguins actually prefer tropical foods such as bananas and pineapple to their traditional diet of fish
- Second, that tropical foods give penguins an odor that makes them unattractive to their traditional predators

...

## Introduction

This article is a review of the book *Dietary Preferences of Penguins*, by Alice Jones and Bill Smith. Jones and Smith's controversial work makes three hard-to-swallow claims about penguins:

First, that penguins actually prefer tropical foods such as bananas and pineapple to their traditional diet of fish

Second, that tropical foods give penguins an odor that makes them unattractive to their traditional predators

# <h1>Introduction</h1>

<p>

This article is a review of the book Dietary Preferences of Penguins, by Alice Jones and Bill Smith. Jones and Smith's controversial work makes three hard-to-swallow claims about penguins:

</p>

<ul>

<li>

First, that penguins actually prefer tropical foods such as bananas and pineapple to their traditional diet of fish

</li>

<li>

Second, that tropical foods give penguins an odor that makes them unattractive to their traditional predators

</li>

</ul>



## **Introduction**

This article is a review of the book *Dietary Preferences of Penguins*, by Alice Jones and Bill Smith. Jones and Smith's controversial work makes two hard-to-swallow claims about penguins:

- First, that penguins actually prefer tropical foods such as bananas and pineapple to their traditional diet of fish
- Second, that tropical foods give penguins an odor that makes them unattractive to their traditional predators

...

```
<!doctype html>
<html lang='en'>
  <head>
    <meta charset='utf-8' />
    <title>Dietary Preferences of Penguins</title>
  </head>
  <body>
    <h1>Introduction</h1>

    <p>
      This article is a review of the book Dietary Preferences...
    </p>
  </body>
</html>
```

# Uniform Resource Locators (URLs)

<https://example.com:4000/a/b.html?user=Alice&year=2019#p2>

Protocol

Hostname

Port

Path

Query

Fragment

# Ways to specify a URL

- Full URL: `<a href='http://stanford.edu/news/2021/'>2021 News</a>`
- Relative URL: `<a href='september'>September News</a>`
  - Same as `http://stanford.edu/news/2021/september`
- Absolute URL: `<a href='/events'>Events</a>`
  - Same as `http://stanford.edu/events`
- Fragment URL: `<a href='#section3'>Jump to Section 3</a>`
  - Scrolls to `<a name='section3' />` within page
  - Same as `http://stanford.edu/events#section3`

# Lots of HTML tags

- `<img>`
- `<video>`, `<audio>`
- `<canvas>`
- `<link>`, `<style>`
- `<script>`

# Include CSS in a page

```
<!-- External CSS file -->
```

```
<link rel='stylesheet' href='/path/to/styles.css' />
```

```
<!-- Inline CSS -->
```

```
<style>
```

```
  body {
```

```
    color: hot-pink;
```

```
  }
```

```
</style>
```

# Include JavaScript in a page

```
<!-- External JS file -->
```

```
<script src='/path/to/script.js'></script>
```

```
<!-- Inline JS -->
```

```
<script>
```

```
    window.alert('hi there!')
```

```
</script>
```

# JavaScript

- Fun
- Flexible
- Immediate feedback
- Pre-installed on every device in the world
- Dev environment is pre-installed too, so easy to start writing code



# Node.js

- JavaScript on the command line
- Adds built-in functions for filesystem, raw sockets, DNS
  - Stuff that belongs in a scripting language, but not a browser
- Also adds module system, binary data support (Buffer)
  - Less necessary because JavaScript has improved rapidly

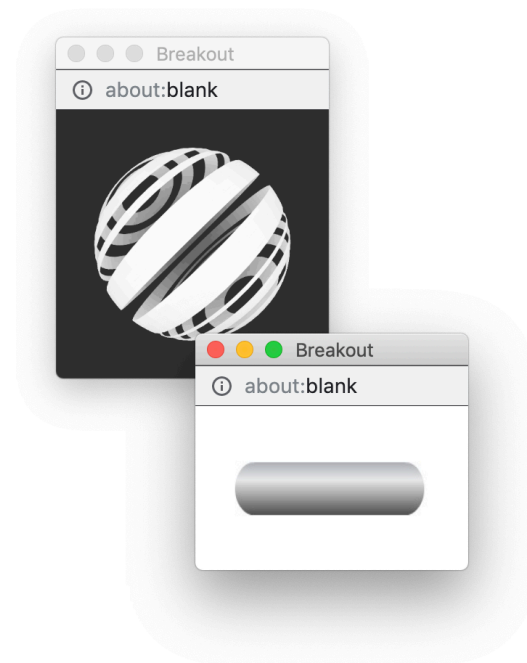
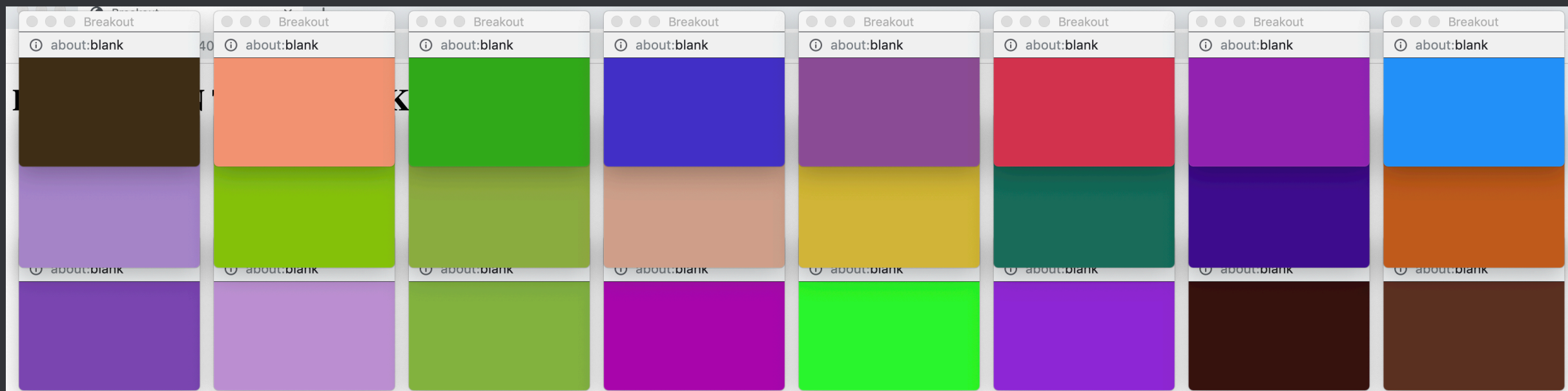
# "JavaScript APIs"

- "JavaScript APIs" can come from:
  - JavaScript language specification
  - Document Object Model specification (browser)
  - Node.js built-ins
- Examples:
  - **Array**
  - **document.createElement**
  - **fs.readFile**

# Crusty browser APIs

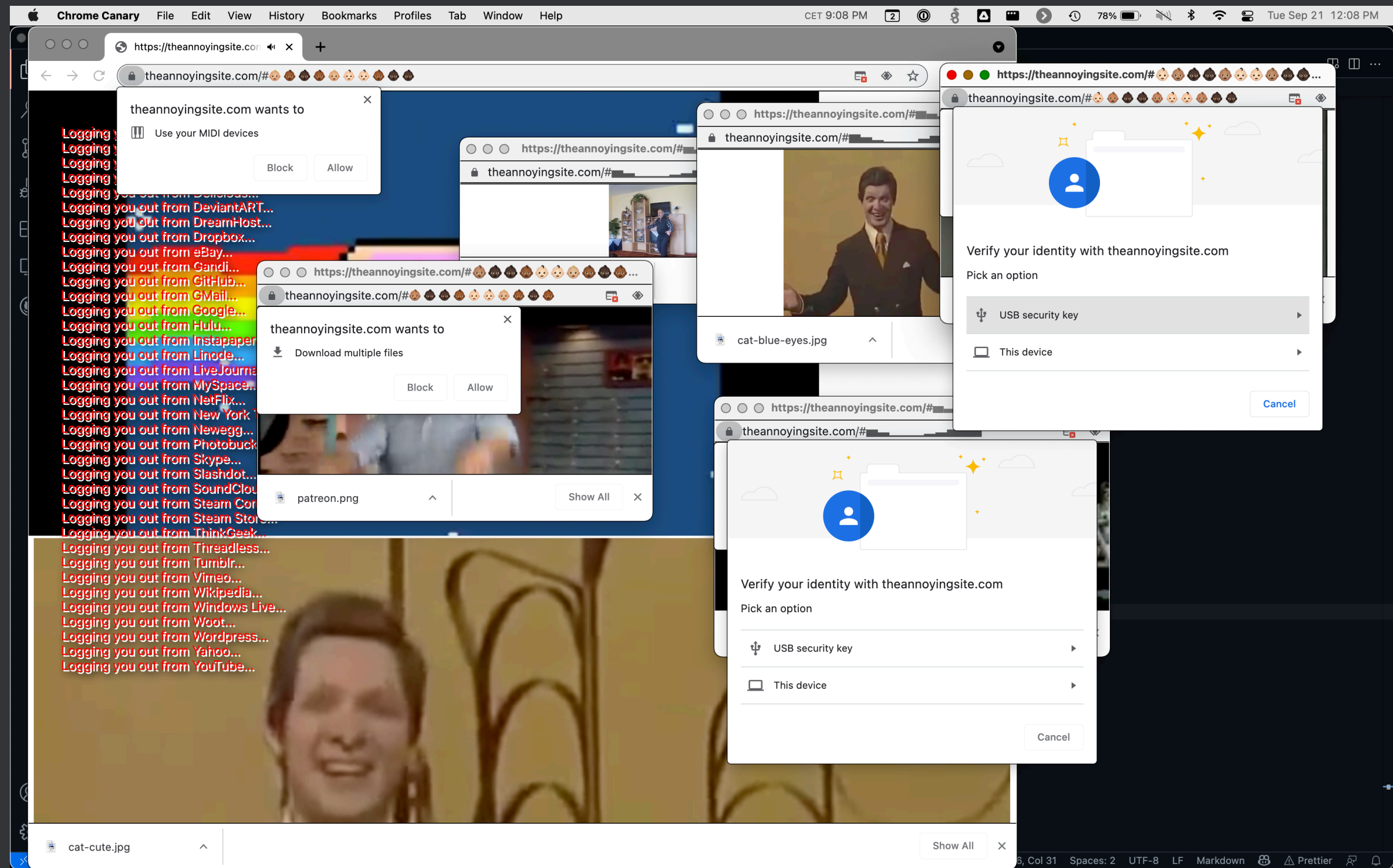
- `window.open()`
- `window.moveTo()`
- `window.resizeTo()`

# Demo: Window breakout game



# Demo: The annoying site

- Demo: <https://theannoyingsite.com>
  - WARNING: Open this in an alternate browser
- Source code available
  - <https://github.com/feross/theannoyingsite.com>



# END