

# Web Attacks

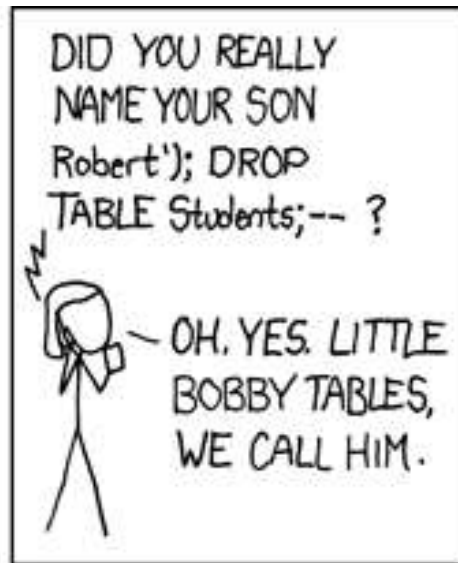
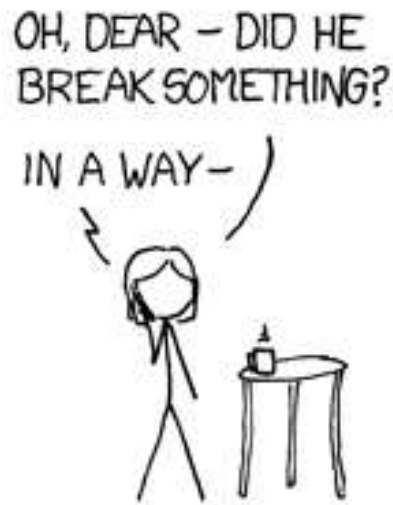
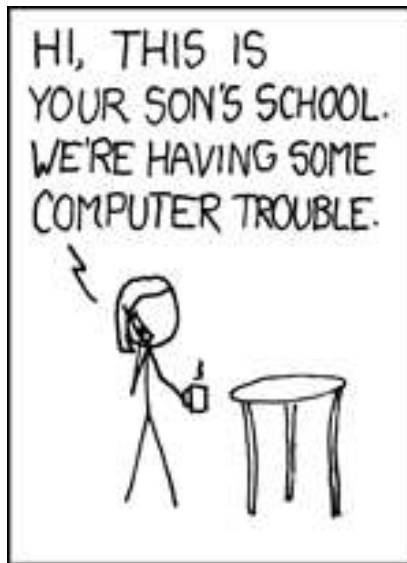


**Blase Ur, David Cash, Ben Zhao**  
UChicago CMSC 23200//33250

# SQL Injection

- Goal: Change or exfiltrate info from *victim.com*'s database
- Main idea: Inject code through the parts of a query that you define

# SQL Injection



# SQL Injection

- Prerequisites:
  - Victim site uses a database
  - Some user-provided input is used as part of a database query
  - DB-specific characters aren't (completely) stripped

# SQL Injection: How?

- Enter DB logic as part of query you impact
- Back-end query
  - `SELECT * FROM USERS WHERE USER=' ' AND PASS=' ';`
- For username & password, attacker gives:
  - `' or '1'='1`
- Straightforward insertion:
  - `SELECT * FROM USERS WHERE USER=' ' or '1'='1' AND PASS=' ' or '1'='1';`

# SQL Injection: Why Does This Work?

- Database does what you ask in queries!

# SQL Injection: Key Mitigations

- Sanitize / escape user input
  - Harder than you think!
  - Different encodings
  - Use libraries to do this!
- Prepared statements from libraries handle escaping for you!
- E.g., mysqli (in place of mysql) for PHP

# Cross-Site Scripting (XSS)

- Goal: Run JavaScript on someone else's domain to access that domain's DOM
  - If the JavaScript is inserted into a page on *victim.com* or is an external script loaded by a page on *victim.com*, it follows *victim.com*'s same origin policy
- Main idea: Inject code through either URL parameters or user-created parts of a page



# Cross-Site Scripting (XSS)

- Variants:
  - *Reflected XSS*: The JavaScript is there only temporarily (e.g., search query that shows up on the page or text that is echoed)
  - *Stored XSS*: The JavaScript stays there for all other users (e.g., comment section)
- Prerequisites:
  - HTML isn't (completely) stripped
  - *victim.com* echoes text on the page
  - *victim.com* allows comments, profiles, etc.

# XSS: How?

- Type `<script>EVIL CODE ();</script>` into form field that is repeated on the page
- Do the same, but as a URL parameter
- Add a comment (or profile page, etc.) that contains the malicious script
- Malicious script accesses sensitive parts of the DOM (financial info, cookies, etc.)
  - Change some values
  - Exfiltrate info (load *attacker.com/?q=SECRET*)

# XSS: Why Does This Work?

- All scripts on *victim.com* (or loaded from an external source by *victim.com*) are run with *victim.com* as the origin
  - By the Same Origin Policy, can access DOM

# XSS: Key Mitigations

- Sanitize / escape user input
  - Harder than you think!
  - Different encodings
  - `<img onmouseover="EVIL CODE ();" />`
  - Use libraries to do this!
- Define Content Security Policies (CSP)
  - Specify where content (scripts, images, media files, etc.) can be loaded from
  - `Content-Security-Policy: default-src 'self' *.trusted.com`

# Cross-Site Request Forgery (CSRF)

- Goal: Make a user perform some action on a website without their knowledge
  - Trick the browser into having them do this
- Main idea: Cause a user who's logged into that website to send a request that has lasting effects

# Cross-Site Request Forgery (CSRF)

- Prerequisites:
  - *Victim* is logged into *important.com* in a particular browser
  - *important.com* accepts GET and/or POST requests for important actions
  - *Victim* encounters *attacker's* code in that same browser

# CSRF Example

- *Victim* logs into *important.com* and they stay logged in (within some browser)
  - Likely an auth token is stored in a cookie
- *Attacker* causes *victim* to load  
`https://www.important.com/transfer.php?amount=100000000&recipient=blase`
  - This is a GET request. For POST requests, auto-submit a form using JavaScript
- Transfer money, cast a vote, change a password, change some setting, etc.

# CSRF: How?!

- On *blaseur.com* have `<a href="URL">Cat photos</a>`
- Send an HTML-formatted email with ``
- Have a hidden form on *blaseur.com* with JavaScript that submits it when page loads
- Etc.



# CSRF: Why Does This Work?

- Recall: Cookies for *important.com* are automatically sent as HTTP headers with every HTTP request to *important.com*
- *Victim* doesn't need to visit the site explicitly, but their browser just needs to send an HTTP request
- Basically, the browser is confused
  - “Confused deputy” attack

# CSRF: Key Mitigations

- Check HTTP referer
  - But this can sometimes be forged
- CSRF token
  - “Randomized” value known to *important.com* and inserted as a hidden field into forms
  - Key: not sent as a cookie, but sent as part of the request (HTTP header, form field, etc.)

# Online Tracking

- Advertisers want to show you advertisements targeted to your interests and demographics

## Ads Preferences

† Ads on Search and Gmail

~ Ads on the web

Opt out

### How your ads are personalized

Ads are based on personal info you've added to your Google Account, data from advertisers that partner with Google, and Google's estimation of your interests. Choose any factor to learn more or update your preferences. [Learn more](#)

Accounting & Finance Jobs

Action & Platform Games

Android OS

Banking

Beaches & Islands

Bollywood & South Asian Film

Business & Productivity Software

Action & Adventure Films

Adventure Games

Autos & Vehicles

Bars, Clubs & Nightlife

Blues

Books & Literature

Business News

Ads on the web

**Make the ads you see on the web more interesting**

Many websites, such as news sites and blogs, partner with us to show ads to their visitors. To see ads that are more related to you and your interests, edit the categories below, which are based on sites you have recently visited. [Learn More](#)

Your interests are associated with an advertising cookie that's stored in your browser. If you don't want us to store your interests, you can opt out below. Your ads preferences only apply in this browser on this computer. They are reset if you delete your browser's cookies.

† Watch a video: [Ads Preferences on GDN explained](#)

### Your categories

Below you can review the interests and inferred demographics that Google has associated with your cookie. You can [remove](#) or [edit](#) these at any time.

Arts & Entertainment

Computers & Electronics

Computers & Electronics - Consumer Electronics - Gadgets & Portable Electronics - PDAs & Handhelds

Internet & Telecom

Internet & Telecom - Mobile & Wireless - Mobile Phones - Smart Phones

Law & Government

Science

### Your demographics

We infer your age and gender based on the websites you've visited. You can [remove](#) or [edit](#) these at any time.

Age: 35-44

Gender: Male

# Online Tracking

- JavaScript / images from advertising networks loaded as part of your page
  - In iframes
  - Or sometimes not
  - Why does this matter?

# Ubiquity of Online Tracking

The screenshot displays the Ghostery browser extension interface. The top bar is blue with the Ghostery logo and name. The main content area is split into two panels. The left panel shows a summary: '22 Trackers found on www.mynews.com' with a circular progress indicator showing '22' in the center and '4 Blocked' below it. Below this are three buttons: 'Trust Site' (with a shield icon), 'Restrict Site' (with a lock icon), and 'Pause Ghostery'. A link 'Map these trackers' is at the bottom of the left panel. The right panel is titled 'TRACKERS' and has a 'Block All' checkbox. It lists several trackers under the 'Advertising' category, with 10 total and 3 blocked. The blocked trackers are Advertising.com, Moat, and NetRatings-Site-Center, each with a red checkmark in a box. Other trackers listed are DoubleClick, Google Adsense, Korrelate, Polar-Mobile, ScoreCard Research Beacon, and Tacoda, each with an empty checkbox.

Tracker Name	Status
Advertising	10 TRACKERS 3 Blocked
Advertising.com	Blocked (checked)
DoubleClick	Not Blocked
Google Adsense	Not Blocked
Korrelate	Not Blocked
Moat	Blocked (checked)
NetRatings-Site-Center	Blocked (checked)
Polar-Mobile	Blocked (checked)
ScoreCard Research Beacon	Not Blocked
Tacoda	Not Blocked

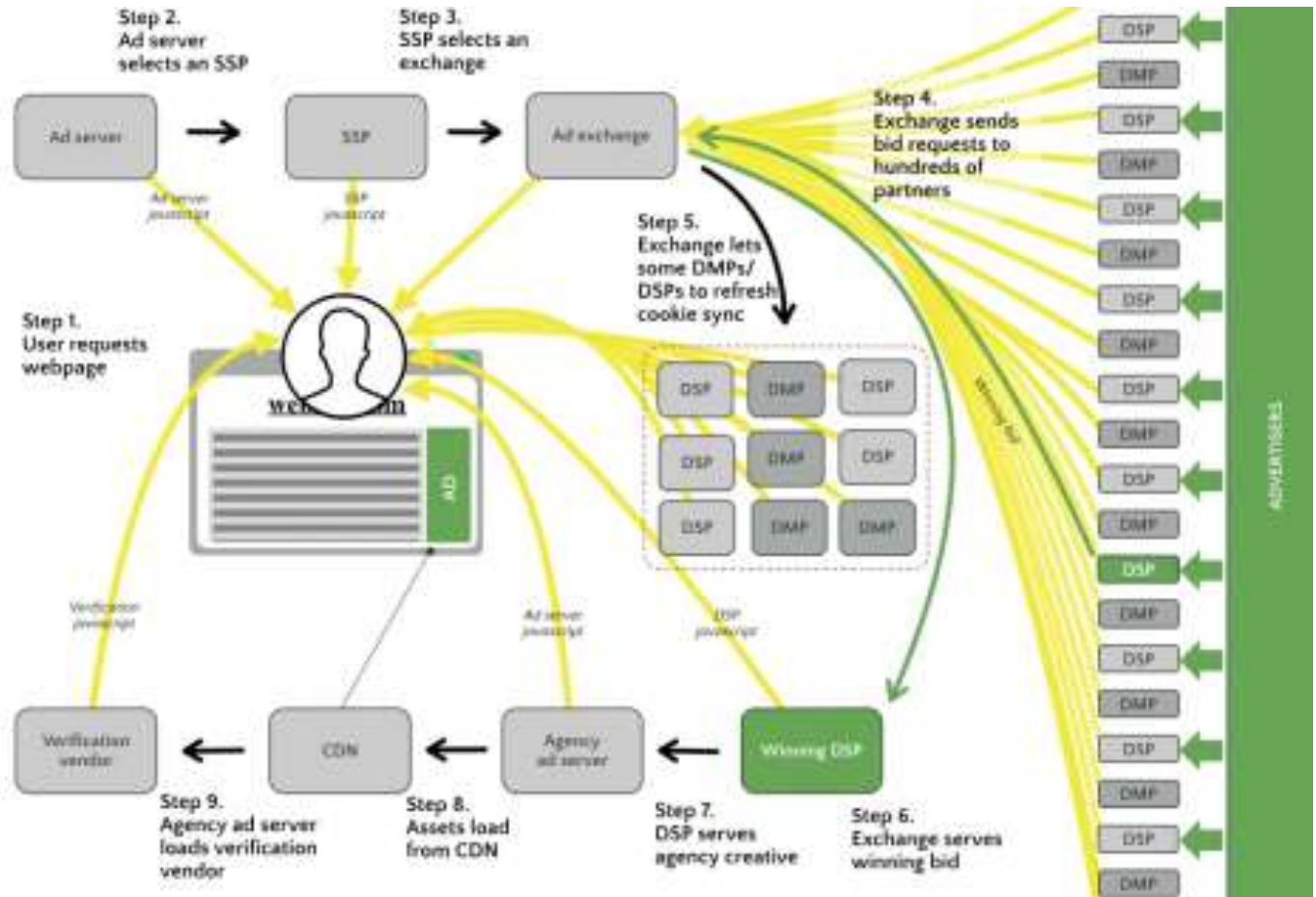
# Ad Bidding Marketplaces

## DATA LEAKAGE IN ONLINE ADVERTISING

This is the current process of real-time bidding that is used in online behavioural advertising.

### Legend

- Channel of data leakage
- Money
- Personally identifiable information



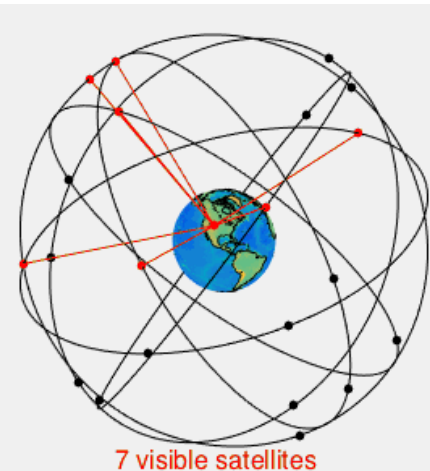
# Device Fingerprinting

- Use unique(-ish) combination of device features as an identifier
- <https://panopticklick.eff.org/>



# Location Tracking

- IP Geolocation
  - Hierarchy of IP addresses
- GPS (Global Positioning System)
  - ~31 satellites in semi-synchronous orbit in OUTER SPACE with atomic clocks
  - Always ~6 satellites in line of sight
  - Multilateration





# What Does HTTPS Hide?

- Body of the HTTP request / response is hidden
- ...So what's left to be seen / inferred?

# Side Channels

- Using metadata or outside observations to make inferences about the data



# Web Side Channels Include:

- Size of packets
  - How can this reveal what pages you are visiting?
- Timing

## Remote Timing Attacks are Practical

David Brumley  
*Stanford University*  
dbrumley@cs.stanford.edu

Dan Boneh  
*Stanford University*  
dabo@cs.stanford.edu

### Abstract

Timing attacks are usually used to attack weak computing devices such as smartcards. We show that timing attacks apply to general software systems. Specifically, we devise a timing attack against OpenSSL. Our experiments show that we can extract private keys from an OpenSSL-based web server running on a machine in the local network. Our results demonstrate that timing attacks against network servers are practical and therefore

The attacking machine and the server were in different buildings with three routers and multiple switches between them. With this setup we were able to extract the SSL private key from common SSL applications such as a web server (Apache+mod\_SSL) and a SSL-tunnel.

**Interprocess.** We successfully mounted the attack between two processes running on the same machine. A hosting center that hosts two domains on the same machine might give management access to the admins of each domain. Since both domain are