Week 1.25 – Web 1

CS-UY 3943-G / CS-GY 9223-H

Nick Gregory

Common Web Bugs

- Most web bugs stem from just a few things:
  - SQLi
  - XSS
  - CSRF
  - File inclusion
  - Command injection
- Summarized by the OWASP Top 10

SQLi

But first: SQL

SQL

- SQL?
  - Language used to talk with databases
  - Human readable
    - Literally query strings sent to the server
  - Used **everywhere**
    - Basically any website that stores data
  - Different servers implement slightly different dialects
    - MySQL is most common and what we'll be focusing on
    - "Core functionality" is defined in SQL-99

SQL cont'd

- 3 basic "tiers" for structuring data
  - Database
  - Table
  - Column
- Very similar to Excel & related
- Each data row has a value for each column (possibly NULL)

SQL cont'd

```
SELECT id, name, password FROM users WHERE name LIKE 'nick%';
```

- Gets the ID, name, and password for each row where the name starts with 'nick'
  - LIKE statements use '%' as a wildcard
- Case insensitive by default
- Capitalization of SELECT, FROM, WHERE, etc. doesn't matter
- Strings are enclosed with single quotes
  - Quotes inside the string are escaped with backslashes

SQL cont'd

- SQL statements are typically performed "in the context" of a specific DB
  - Set at connect time or with a `USE db_name` statement
- Queries can access all databases the connecting user has access to though
  - `SELECT username FROM cs3284_demos.users WHERE id = 1;`

SQL cont'd

```
SELECT 1, 2;
```

- SELECTs can also select constants or functions
- There are also built-in functions which can be SELECTed from
  - VERSION()
  - DATABASE()
  - SLEEP(n)
  - ...

SQL cont'd

```
SELECT name, password FROM users WHERE id IN (SELECT id FROM
banned);
```

- Subqueries!
- Selects all entries from the banned table's id column
- Then uses that to filter which results are returned from `users`

SQL cont'd

- Other statements:
  - INSERT INTO {table} [(col1, col2, …)] VALUES (1,2,...)[, (3,4,...)]
    - Adds 1 or more new rows to the table
  - UPDATE {table} SET col1=1 [, col2=2, …] WHERE col3='foo';
    - Changes data in all rows where the WHERE clause matches
- Misc items:
  - Semicolon at end of statement is optional for us (but reqired in CLI)
  - Comments: `SELECT 1,2 -- Selects 1 and 2`

SQL cont'd

- SQL is human-readable on the wire
- The parameters (e.g. `'nick%'`) can either be embedded or parameterized
  - Embedded means the string is literally in the query
    - Still seen sadly
  - Parameterization means there are placeholders in the query and the arguments are sent separately
    - Greatly preferred nowadays, and is how all ORM-based systems work

SQL Injection

```
SELECT * FROM users WHERE name = '$name';
```

If $name is directly replaced with user input...

SQLi

SQL Injection

```
SELECT * FROM users WHERE name = '$name';
```

- `$name = "foo'bar";`
- `SELECT * FROM users WHERE name = 'foo'bar;`
- `` `bar` `` is now outside of the string
  - Syntax error
- But how can we make this evil...

SQL Injection

```
SELECT * FROM users WHERE name = ‘$name’ AND password =
‘$password”;
```

- `users` is a table with columns `id, name, password`
- We want to log in as admin (so name=‘admin’)
- But of course we don’t have the password!
- How can we do this?
  - There are multiple ways

SQL Injection

```
SELECT * FROM users WHERE name = ‘$name’ AND password =
‘$password”;
```

- $password = “asdf’ OR name = ‘admin’”;

- SELECT * FROM users WHERE name = ‘$name’ AND password = ‘asdf’ OR name = ‘admin’”;

SQL Injection

```
SELECT * FROM users WHERE name = ‘$name’ AND password =
‘$password”;
```

- `$name = “admin’ -- ”;`
- `SELECT * FROM users WHERE name = ‘admin’ --
AND password = ‘’”;`
- That’s be basic idea with SQLi: you’re injecting your own queries/statements into an existing query to change the result

Fixing SQLi

- Obvious solution: escape the quotes
- Transform ' INTO \'
  - Lets the SQL server know that the single quote is part of the string, not the close quote
- So `$name.replace("'", "\'")` fixes everything, right?
- …

Fixing SQLi

```
$name = “\'”;
$name = $name.replace(“'”, “\'”);
$name: “\\'”
```

- We bypassed the escaping!
- So let’s escape more things

Fixing SQLi

- Enter `mysql_escape_string()`
  - For PHP at least
- Escapes ', \, etc.
- Good now, right?

Fixing SQLi

- Databases can specify their character encoding
  - ASCII
  - UTF-8
  - Shift JIS
  - And many more…
- Different encodings may represent the same byte sequence as different text
  - e.g. 0x5c is ＼ in ASCII, but ￥ in Shift JIS

Fixing SQLi

- `mysql_real_escape_string`
  - For realz this time
- Takes both a handle to the DB so it can check the encoding as well as the string to escape
- Finally, all good.

Fixing SQLi

- Except…

SQL Injection

```
SELECT * FROM users WHERE id = $id;
```

- Now we don't have to even worry about escaping things
- `$id = "0 OR ..."`

- Of course the solution here is to `intval($id)`
  - … but of course people miss this like everything else

SQLi++

- Logging in as admin is great
- But what if we want to exfiltrate data?
  - Like user credentials!

- How can we SELECT out arbitrary data?

SQLi++

- `UNION`
- Allows you to UNION the results of 2 queries
  - *shocking*
- Funny enough, I don't think I've ever seen this used in non-SQLi situations…
- Main constraint: number of columns has to match
  - How can we figure this out (if we don't have source)?
  - Brute force the number based on response code

SQLi++

- `SELECT id, name FROM … UNION SELECT 1`

  is a syntax error
  - Odds are good this will result in a HTTP 500 from the web server
- Just keep trying different numbers of columns (usually ~15 is the most you'll see)
  - One number should eventually work

SQLi++

```
SELECT * FROM users WHERE id = $id;
```

- `users` has 3 columns (id, name, password)
- Assume the name is returned on the page
- We want to leak the MySQL server's version
- What should `$id` be?

SQLi++

```
SELECT * FROM users WHERE id = $id;
```

- `$id = 0 UNION SELECT 1, VERSION(), 3`

- The name field should now be the MySQL server version

Demos

SQLi++

```
INSERT INTO users (username, fullname, password) VALUES
        (‘$username’, ‘$fullname’, ‘$pass’);
```

- Imagine you have a registration page
  - The username is vulnerable to SQLi
    - Obviously saved in the DB when registration is successful
- INSERTs can’t return data
  - This is sort of a lie, but it’s true for our purposes here
- So what can we do?

SQLi++

```
INSERT INTO users (username, fullname, password) VALUES
('$username', '$fullname', '$pass');
```

- We can use subqueries inside the INSERT

```
$username = "sum_user', (SELECT VERSION()), 'password') -- ";
```

- Full query is now:

```
INSERT INTO users (username, fullname, password) VALUES ('sum_user', (SELECT
VERSION()), 'password') -- ', '$fullname', '$pass');
```

- N.B: The inner "SELECT" isn't actually needed here, but it makes it a bit easier to see the arbitrary query capability

SQLi++

- Now when you look at your “Full Name”, the SQL server version will be in there!
- This same principle can be applied with UPDATEs

Demos