# CS461 MP2 Part 1 Web Security

JOSHUA REYNOLDS

## Web Security

Be careful with this one

Very easy to break CFAA accidentally by testing what you learn here

Don’t try techniques on any site you don’t own without explicit, written permission

Check out bug bounty programs if this interests you

## The Web – Uniform Resource Locators

http://www.google.com:80/signin/v2/account.php?xid=039285&m=20#Login

## The Web – Communication

1. Server machine listens to a port
2. Client (browser) looks up server’s domain name
3. Client makes a TCP connection to server
4. Client makes a request following the HTTP protocol
5. Server responds, also using the HTTP protocol

## HyperText Transfer Protocol

Client makes a request for a resource specified by a URL

Server sends a response

# HyperText Transfer Protocol

Requests:

- GET – (May include a query string)
- POST – (May include form data)

Response Codes:

- 2?? – Success
- 3?? – Look Elsewhere
- 4?? – Client problem
- 5?? – Server problem

## Web Cookies

Small persistent storage on the client browser

Protected from access by code from other servers

One of many add-ons to HTTP

Helps servers be stateless

## Seeing inside your browser

Demo

## Web Frameworks - Examples

Play - Scala

Zend - PHP

Bottle - Python

Spring - Java

ASP.NET – C#

https://en.wikipedia.org/wiki/Web_framework

## Structured Query Language (SQL) Database

Web servers need databases

4 Things you will do in this MP:

- Create MySQL users and manage permissions
- Create MySQL tables
- Query a MySQL database
- Inject malicious queries developers never intended

## Web Development – Front and Back End

**“Back End”**

- Server Code
- Database

**“Front End”**

- HTML/CSS
- JavaScript

## Cross-Site Scripting

Trick a server into serving extra code to other users.

ex:

Hi, my name is <script>alert(“Hi”);</script>

## Cross-Site Request Forgery

Impersonate a request from a client they never intended

## SQL Injection

SELECT (username, email) FROM Users WHERE username = “bob” and password = Password(“123456”);

## SQL Injection

```
 username = “bob”;

 passwd = “123456”;

 query = sprintf(“SELECT * FROM Users
WHERE username = ‘%s’ AND password =
Password(‘%s’)”, username, passwd);
```

## SQL Injection

```
username = "' -- ";

passwd = "123456";

query = sprintf("SELECT * FROM Users
WHERE username = '%s' AND password =
Password('%s')", username, passwd);
```

## SQL Defense - Prepared Statements

Just like a format string, but use a library implementation to escape all SQL badness.

## Auto-Grader Environment

For consistency, all tests will be run in the MP1 VM

If you develop solutions elsewhere, be sure to test them thoroughly in the VM to make sure there is no unexpected behavior.

- (\r\n problems, python versions, browser versions, etc)

## 2.1.1: Setup Bungle

Get your website running using the template code in the mp2 repo

## 2.1.2: SQL Setup

Setup the MySQL database for your website

Follow the specs for every column’s requirements – we’ll check!

Save your SQL for creating tables to 2.1.2.txt

## 2.1.3: Prepared Statements

Defend against SQL injections using Prepared Statements

https://dev.mysql.com/doc/connector-python/en/connector-python-api-mysqlcursorprepared.html

## 2.1.4: Input Sanitation

Filter raw user-provided input to prevent cross-site scripting

## 2.1.5: Token Validation

Protect your site against Cross-Site-Request-Forgery