# CMSC389R

## Web

COMPUTER SCIENCE
UNIVERSITY OF MARYLAND

# recap

Bitcamp demos!

---

Questions?

# agenda

- Background
  - HTTP
  - HTTP requests (GET/POST)
  - Cookies, sessions, etc
- Common vulnerabilities
  - Cross-site scripting (XSS)
  - SQL injection (SQLi)
- In class challenge

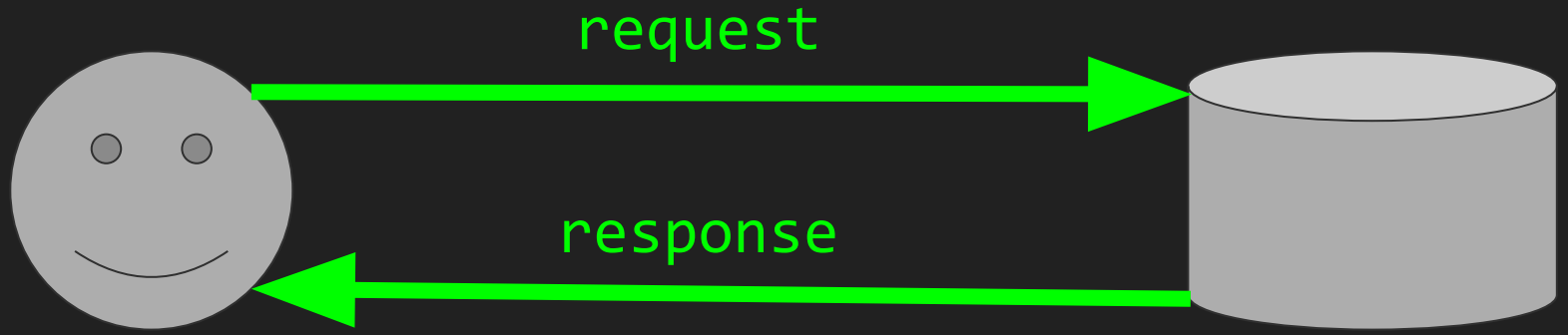
# HTTP

- Hypertext Transport Protocol
  - Usually ports 80 (HTTP), 443 (HTTPS)
  - Stateless by design
    - Stateful by usage…
      - Think cookies & sessions!
  - Built on top of TCP
- Server-side code understands HTTP and responds to requests through this protocol

# Basics of services

- What are web services built with these days?
  - Client-side
    - CSS/HTML/Javascript/etc...
  - Server-side
    - Yes, PHP is still actively used
    - Javascript/Python/Ruby/etc...
  - Databases
    - SQL, PostgreSQL, MongoDB, etc...

# web basics




- browser
- curl
- wget
- ...

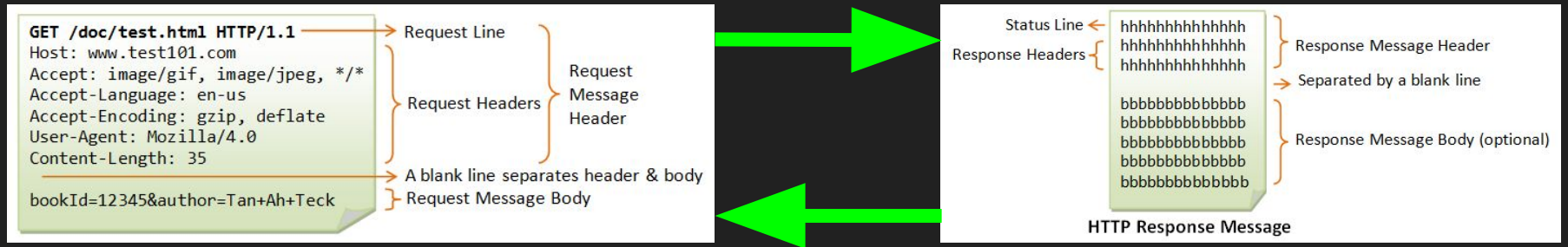
- website
- other server resources

# basics of requests/responses

- When a user triggers an action on the front-end
  - Typically send a request (GET/POST/PUT/...) to the server
  - Server receives request
    - Handles it (data processing/...)
    - Responds
- Front-end handles server’s response
  - Browser renders DOM

# HTTP request basics

## GET and POST*




*there are others, but we'll focus on these

# Cookies

- Piece of data stored client-side
  - Typically passed around in sessions
  - Completely r/w by the client
- Can be dangerous if not used correctly by server!
  - Can be modified in the browser

```
document.cookie="keyofcookie=valueofcookie"
```

Challenge!

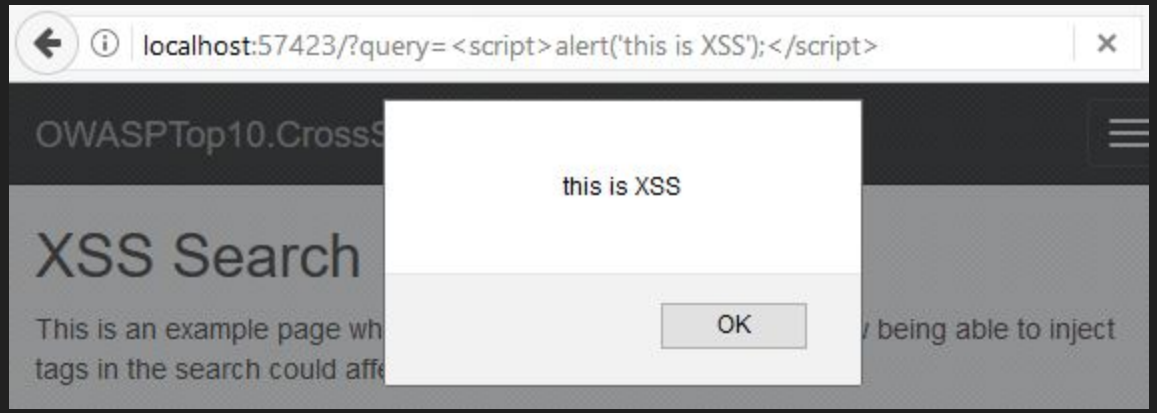
http://159.89.236.106:4567

# Injection Attacks

# cross site scripting (xss)

- Attacker sends malicious code rendered on the victim’s browser
  - Stored: attacker forces malicious code to be stored on database
    - ie) user sets username to injection code; rendered each time victim visits profile
  - Reflected: injected script is reflected off of server to victim
    - Typically sent via email/links/...

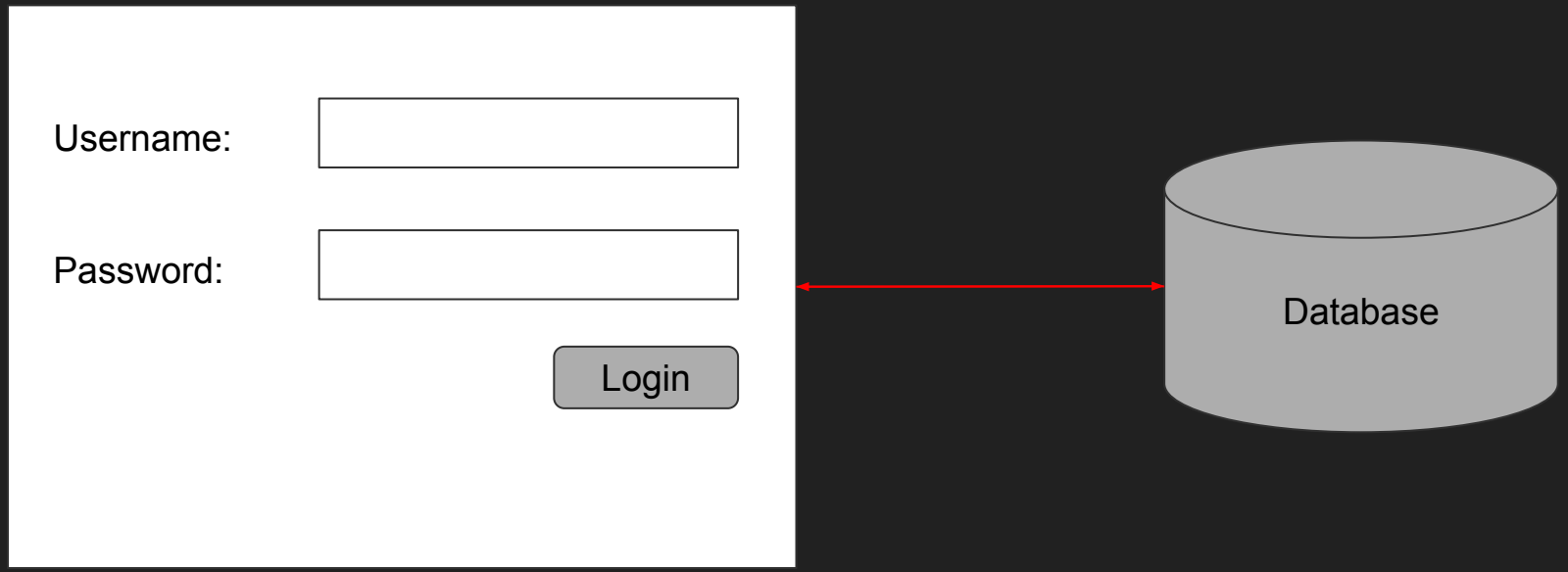
# cross site scripting (xss)



# sql injection

- Website utilizes SQL database
  - Does not sanitize input
  - Query is interpreted as code rather than data
  - Mitigated with prepared statements
- Potentially leads to:
  - Leaking tables
  - Deleting tables
  - Command execution

# sql injection



# sql injection

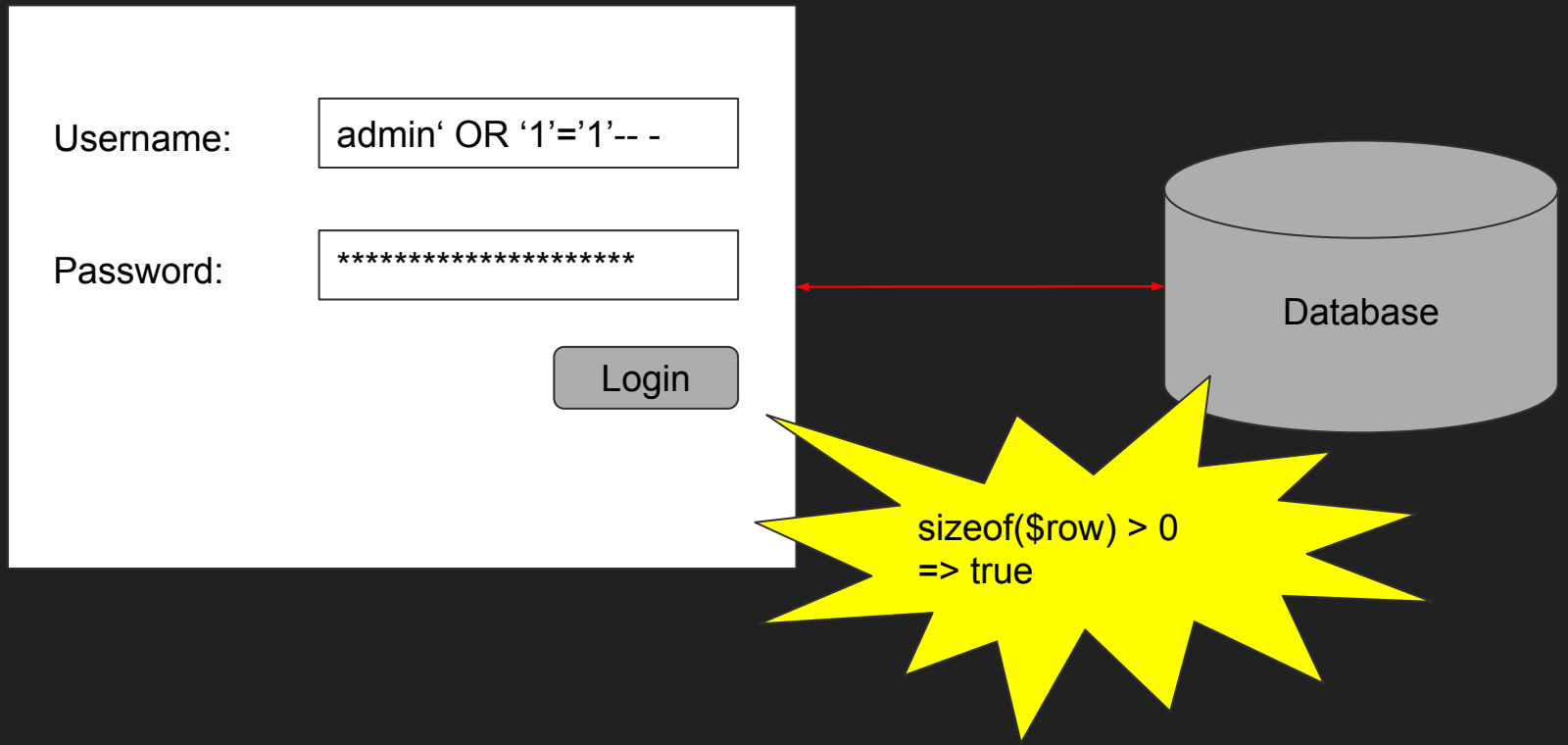
```
function can_access_feature($current_user) {
     global $db_link;

     $db_link = mysqli_connect('localhost', 'dbuser', 'dbpassword', 'dbname');
     $username = $_POST['username'];
     $password = $_POST['password'];

     $res = mysqli_query($db_link, "SELECT * FROM users WHERE username = '".
$username . "' AND password = '" . $password. "';");

     $row = mysqli_fetch_array($res);
     if (sizeof($row) > 0) {
         return true;
     } else {
         return false;
     }
}
```

# sql injection



# resources

- Natas OverTheWire
- JuiceShop
- Gruyere
- Ringzer0team
- OWASP Top 10

# homework #9

Will be posted tonight.

Let us know if you have any questions!

This assignment has two parts.

It is due by 4/19 at 11:59 PM.