# CMSC389R

## Web

COMPUTER SCIENCE
UNIVERSITY OF MARYLAND

# recap

Any questions so far?

# agenda

- Background
  - HTTP
  - HTTP requests (GET/POST)
  - Cookies, sessions, etc
- Common vulnerabilities
  - Cross-site scripting (XSS)
  - SQL injection (SQLi)
  - Local/Remote File Inclusion (L/RFI)

# OWASP Top 10

1. Injection
2. Broken Authentication
3. Sensitive Data Exposure
4. XML External Entities (XXE)
5. Broken Access Control
6. Security Misconfiguration
7. Cross-Site Scripting (XSS)
8. Insecure Deserialization
9. Using Components with Known Vulnerabilities
10. Insufficient Logging & Monitoring

https://www.owasp.org/images/7/72/OWASP_Top_10-2017_%28en%29.pdf.pdf

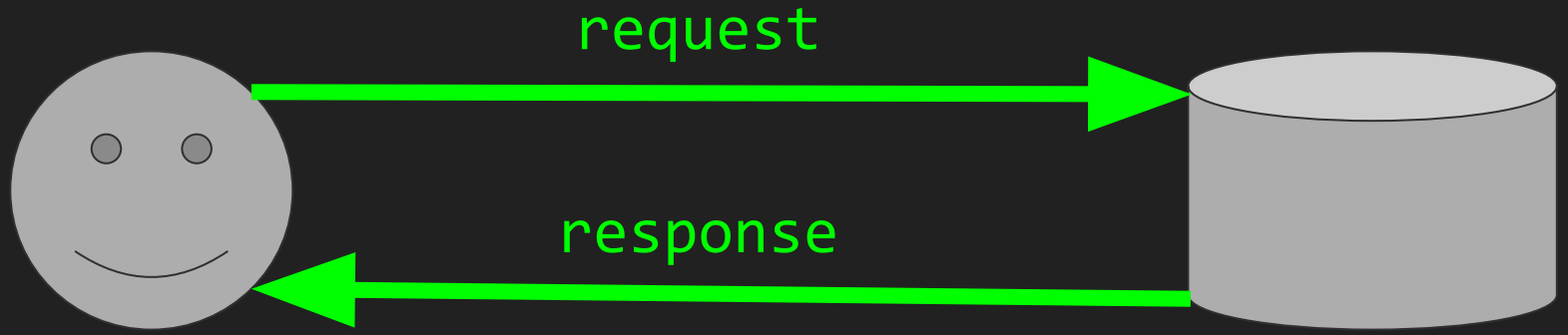
# HTTP

- 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
- ...

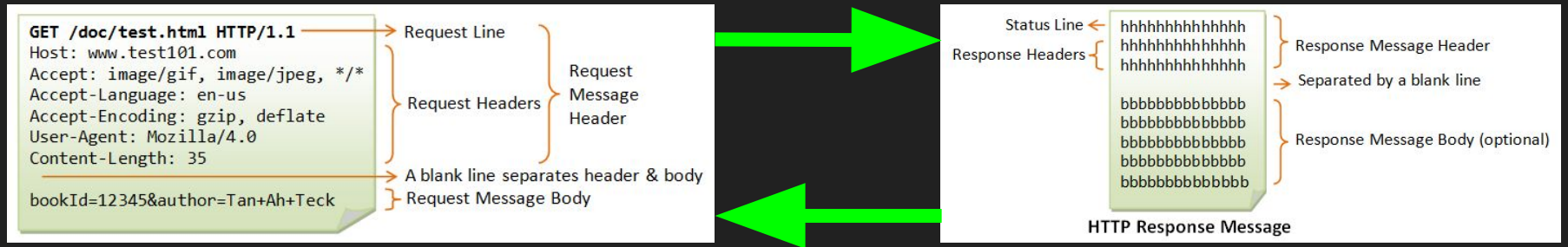
- website
- other server resources

# 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*



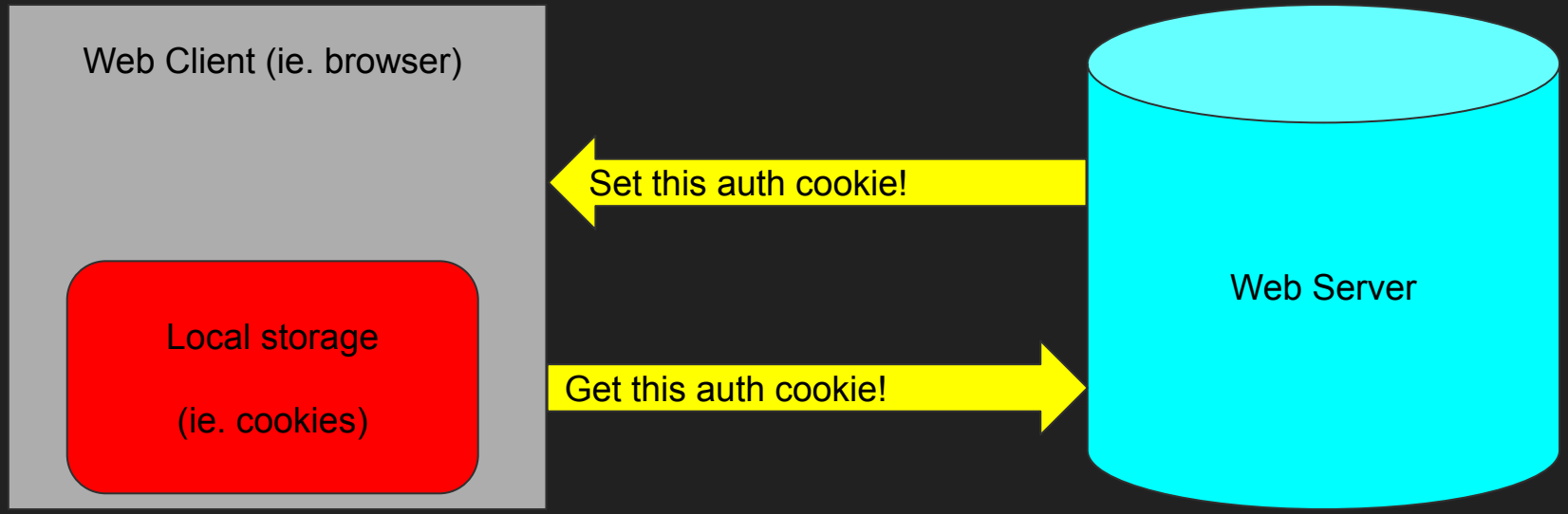

*there are others, but we'll focus on these

# 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"
```

# Cookies

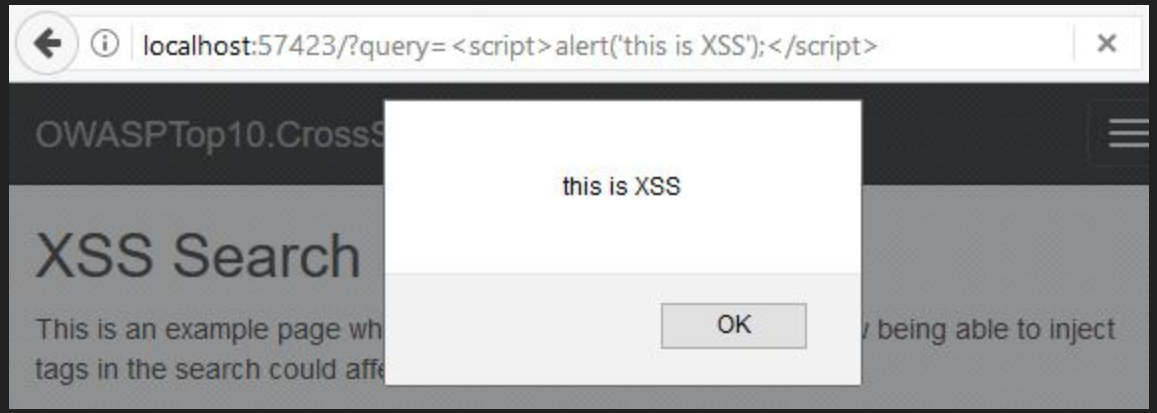

Web Client (ie. browser)
Local storage
(ie. cookies)
Set this auth cookie!
Get this auth cookie!
Web Server

# 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)



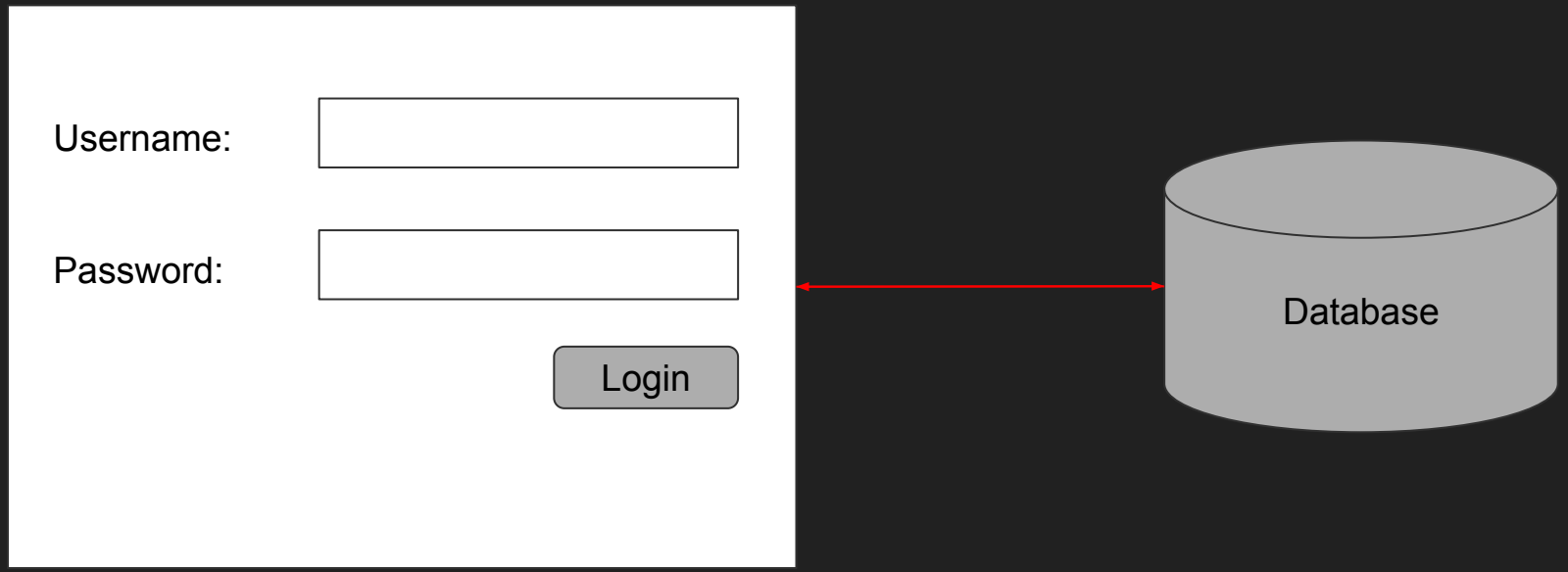
# cross site scripting (xss)

- What if there isn’t enough space in the vulnerable input to store an XSS payload?
  - Javascript can reference external scripts from third-party hosts, so you can craft a JS payload and host it via:
    - malicious.com/bad.js
    - pastebin.com
    - gist.github.com
    - etc ...

# 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 (primer)

**Table:** users

| user | password | last_login_date |
|---|---|---|
| admin | password1234 | 1542392929 |
| michael | this_is_a_bad_password | 1542392920 |

```
SELECT password from users where user = ‘admin’;  → password1234

INSERT INTO users values(‘new_user’, ‘terps’, ‘1542392927’);

SELECT * from users; -- this is a comment
```

# 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

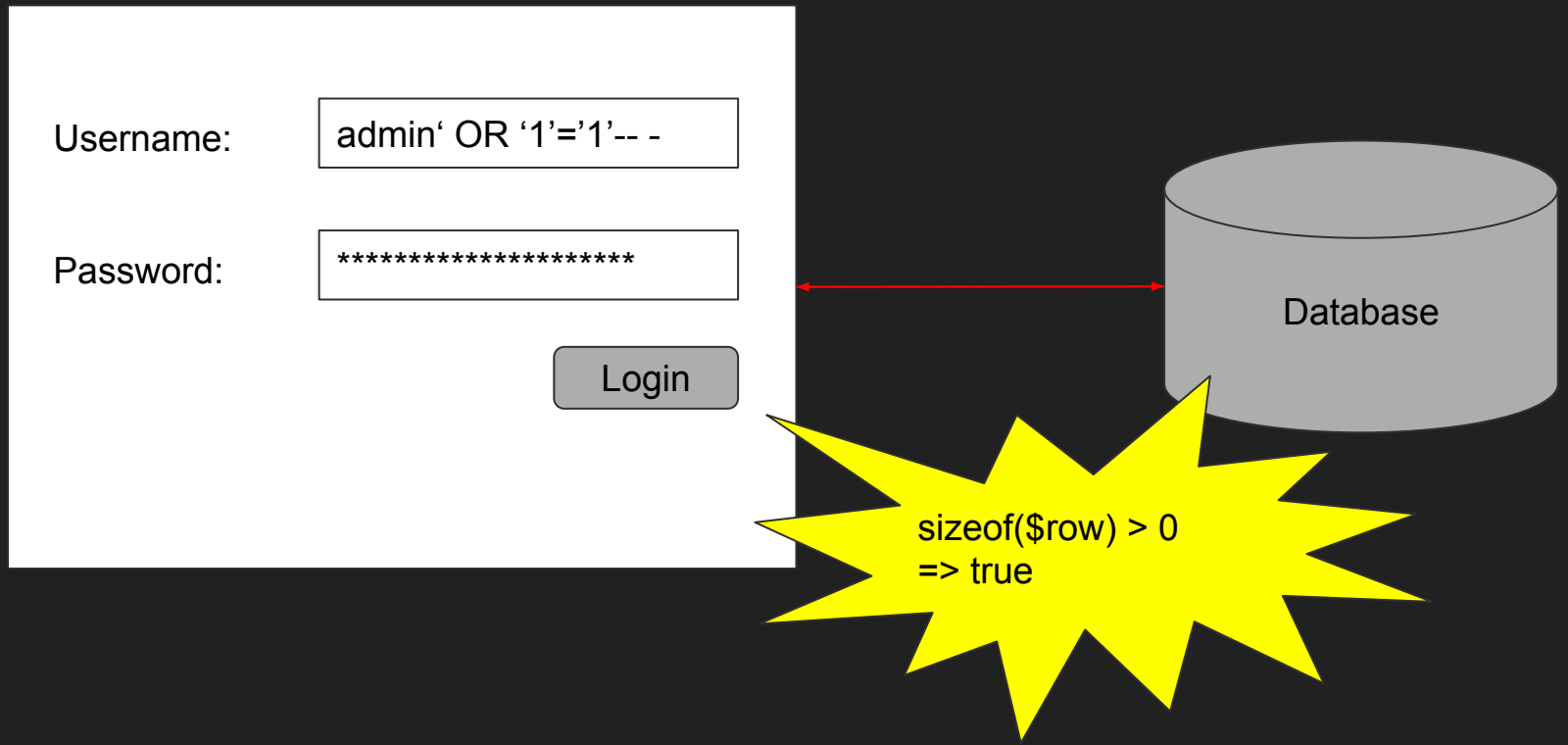
```
function can_access_feature($current_user) {
    global $db_link;

    $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



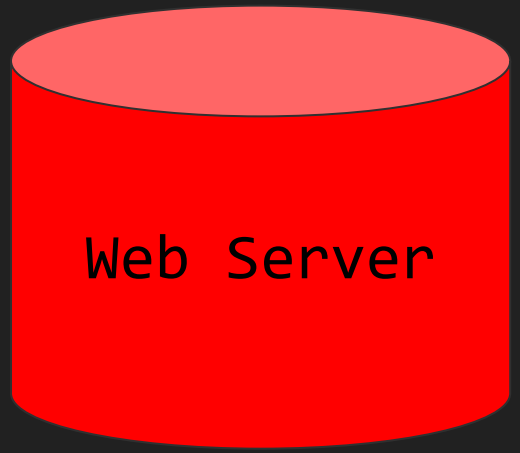

ZU 0666',0,0); DROP DATABASE TABLICE;--

# Local File Inclusion

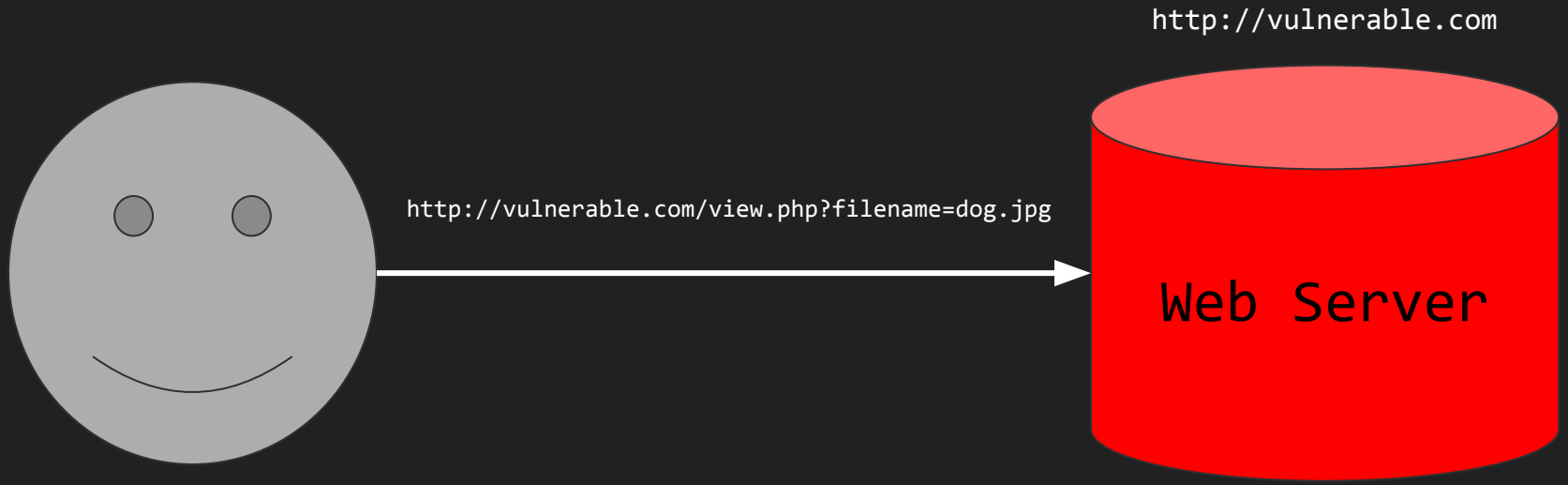
- Attacker includes a file path local to the server as input to a vulnerable field
- Can lead to:
  - Code execution (server & client)
  - DoS
  - Sensitive Information Disclosure

# Local File Inclusion

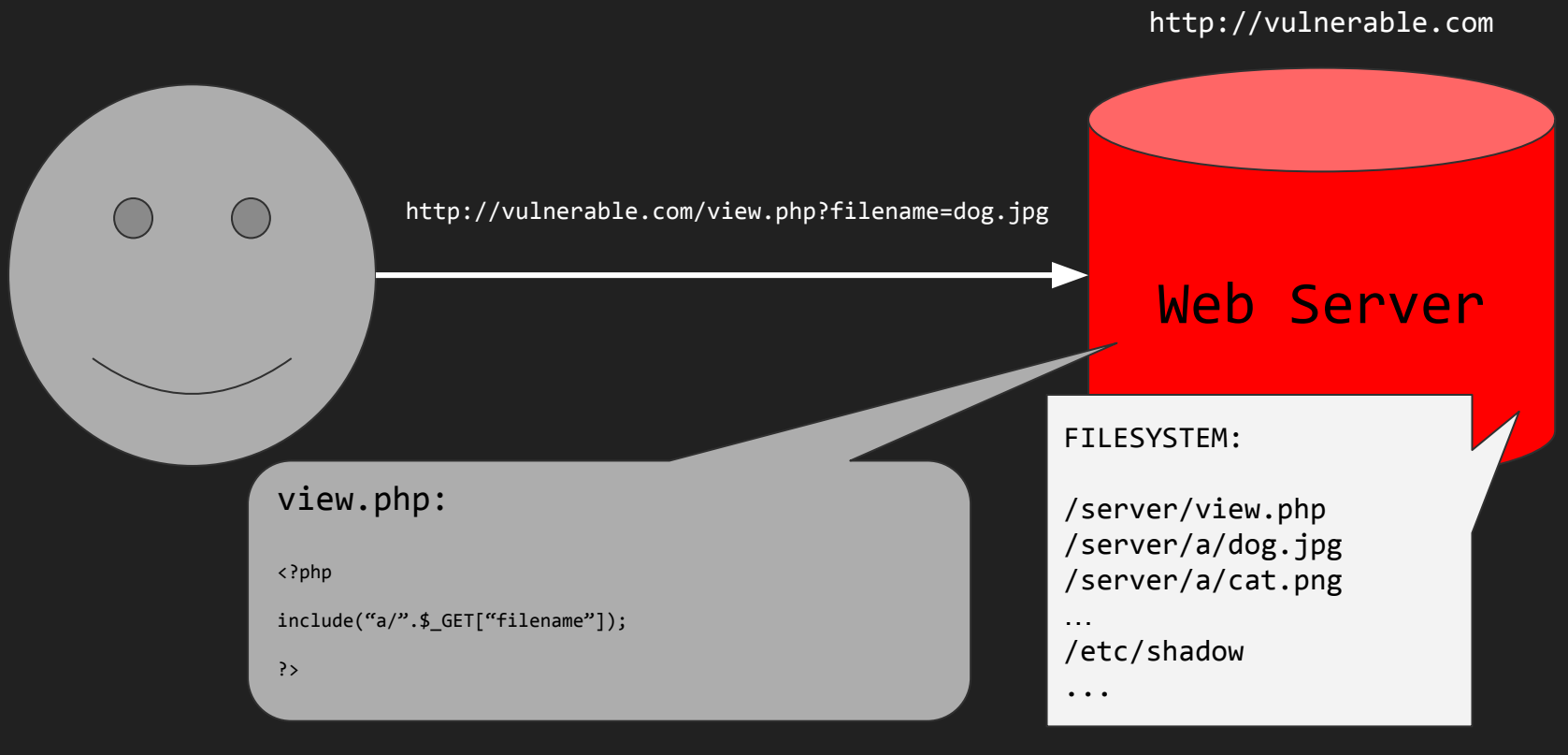
http://vulnerable.com



# Local File Inclusion



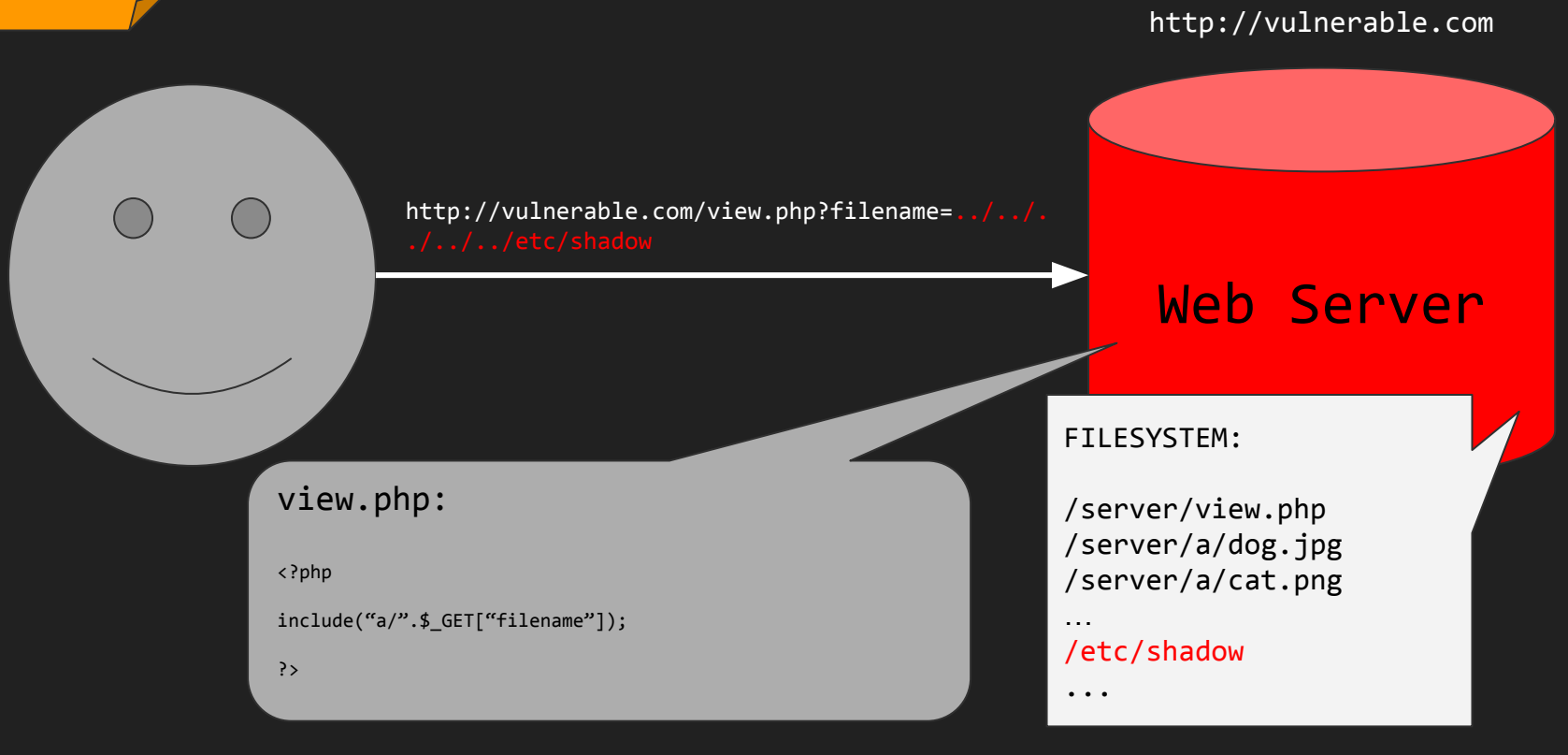
# Local File Inclusion



# Local File Inclusion

Directory Traversal Attack

# Local File Inclusion (ATTACK)



# Remote File Inclusion

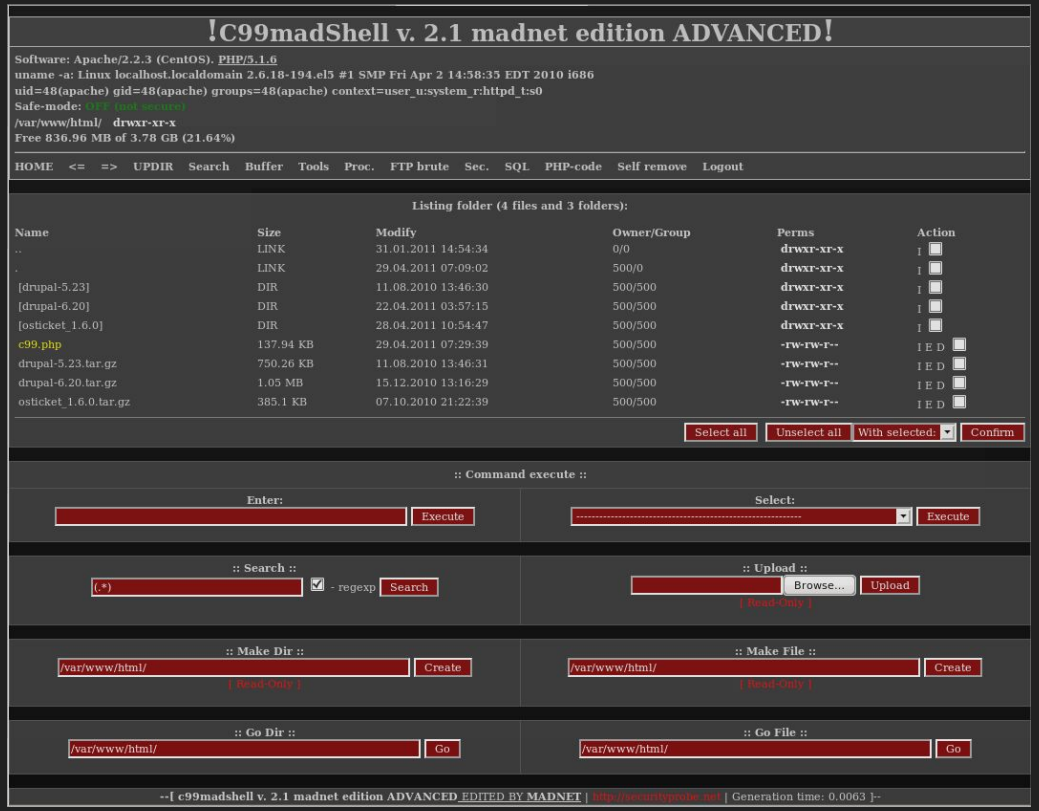
- Similar to LFI, but attacker includes a remote path as input to a vulnerable field
- Can lead to:
  - Code execution (server & client)
  - DoS
  - Sensitive Information Disclosure

# Remote File Inclusion

# Remote File Inclusion

# Remote File Inclusion (ATTACK)

# 0wned


!C99madShell v. 2.1 madnet edition ADVANCED!
Software: Apache/2.2.3 (CentOS). PHP/5.1.6
uname -a: Linux localhost.localdomain 2.6.18-194.el5 #1 SMP Fri Apr 2 14:58:35 EDT 2010 i686
uid=48(apache) gid=48(apache) groups=48(apache) context=user_u:system_r:httpd_t:s0
Safe-mode: OFF (not secure)
/var/www/html/ drwxr-xr-x
Free 836.96 MB of 3.78 GB (21.64%)
HOME <= => UPDIR Search Buffer Tools Proc. FTP brute Sec. SQL PHP-code Self remove Logout
Listing folder (4 files and 3 folders):
Name Size Modify Owner/Group Perms Action
.. LINK 31.01.2011 14:54:34 0/0 drwxr-xr-x I
. LINK 29.04.2011 07:09:02 500/0 drwxr-xr-x I
[drupal-5.23] DIR 11.08.2010 13:46:30 500/500 drwxr-xr-x I
[drupal-6.20] DIR 22.04.2011 03:57:15 500/500 drwxr-xr-x I
[osticket_1.6.0] DIR 28.04.2011 10:54:47 500/500 drwxr-xr-x I
c99.php 137.94 KB 29.04.2011 07:29:39 500/500 -rw-rw-r-- I E D
drupal-5.23.tar.gz 750.26 KB 11.08.2010 13:46:31 500/500 -rw-rw-r-- I E D
drupal-6.20.tar.gz 1.05 MB 15.12.2010 13:16:29 500/500 -rw-rw-r-- I E D
osticket_1.6.0.tar.gz 385.1 KB 07.10.2010 21:22:39 500/500 -rw-rw-r-- I E D
Select all Unselect all With selected: Confirm
:: Command execute ::
Enter: Execute
Select: Execute
:: Search ::
(.*) - regexp Search
:: Upload ::
Browse... Upload
[ Read-Only ]
:: Make Dir ::
/var/www/html/ Create
[ Read-Only ]
:: Make File ::
/var/www/html/ Create
[ Read-Only ]
:: Go Dir ::
/var/www/html/ Go
:: Go File ::
/var/www/html/ Go
--[ c99madshell v. 2.1 madnet edition ADVANCED EDITED BY MADNET | http://securityprobe.net | Generation time: 0.0063 ]--

# Web Application Firewalls (WAF)

- Used by webapps prevent attacks such as XSS, SQLi, RFI/LFI, misconfigs, etc
- Typically rule-based
- Can be implemented in:
  - Server code (Apache ModSecurity, etc)
  - Appliance between network and server (Barracuda, etc)
  - Cloud (AWS WAF, Cloudflare, etc)

# (Some) Approaches to WAF Bypasses

- How does WAF normalize input data?
  - Decode HTML entities?
  - Escape characters?
  - Enforce null byte string termination?
- Is it using regex to match signatures?
  - Blocks ‘OR ‘1’=’1’
  - Allows ‘ || 0x50 is not null

# resources

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

# homework #11

Last homework will be posted tonight!