

MP4: Checkpoint 1



CS 461/ECE 422

What is the Web?

- Platform to deploy applications PORTABLY & SECURELY



client



server

Web Security: Two Tales

Web BROWSER: (client side):

- Attacks target browser security weaknesses
- Result in:
 - Malware installation (keyloggers, botnets)
 - Document theft from corporate network
 - Loss of private data

Web APPLICATION code: (server side)

- Runs at web site: banks, e-merchants, blogs
- Written in PHP, ASP, JSP, Ruby, ...
- Many challenges: XSS, CSRF, SQL injection

Historical Perspective

- The web is an example of “bolt-on security” : add pieces as go
- Originally, the web was invented to allow physicists to share their research papers (so, initially no security in mind when built)
 - Only textual web pages + links to other pages; no security model to speak of
- Then we added embedded images
 - Crucial decision: a page can embed images loaded from another web server
- Then, JavaScript, dynamic HTML, AJAX, CSS, frames, audio, video, ...
- Today, a web site is a distributed application

HTTP Protocol

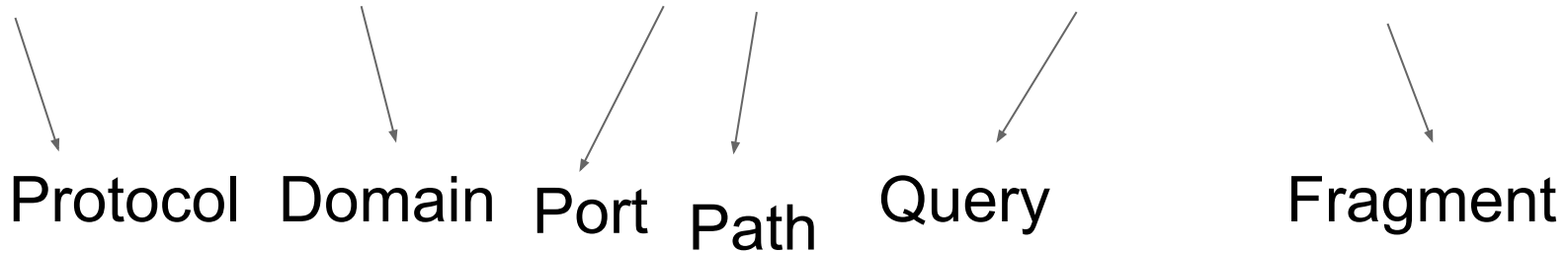
- Hypertext Transfer Protocol
- Stateless
- Unencrypted
- Widely-used & simple
- Language between client & server



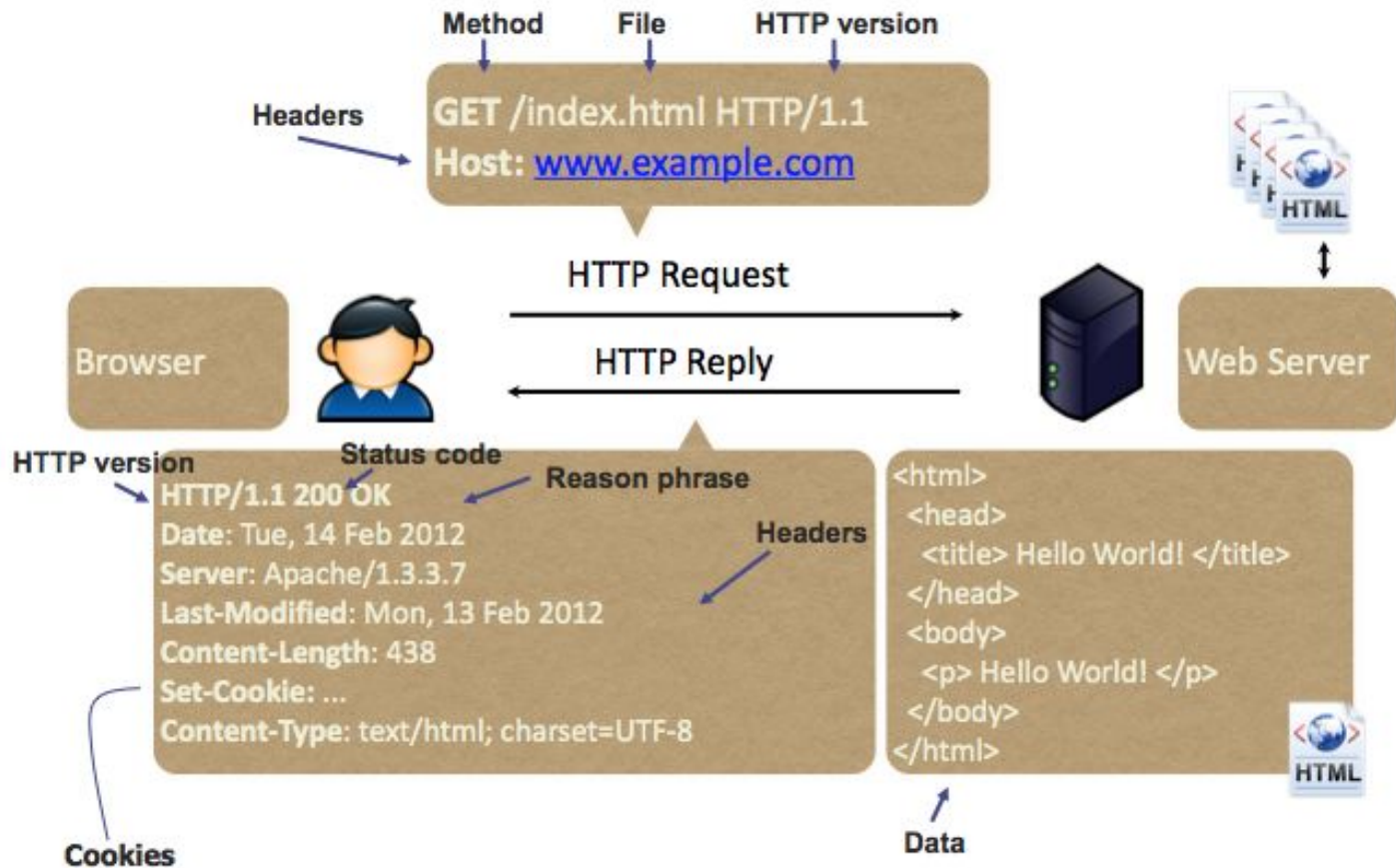
URLs

- Uniform Resource Locator
- Global identifier of network-retrievable documents
- Case-sensitive?

<http://www.unc.edu:81/class?name=cs535#homework>



HTTP Protocol



HTTP Methods

Methods to access a webpage

- GET : requests data from resource
- PUT : puts data at specific URL
- POST : submits data to be PROCESSED to a URL

Others: DELETE, OPTIONS, etc.

HTML

- Hypertext markup language (HTML)
 - Content + formatting of web pages, rendered in browser window
 - What resources required to render page
 - DOM Tree structure: huge and all-encompassing
- HTML Features that could be exploited
 - Links to other pages
 - Embed images by reference
 - Send user input to server via forms
 - Embedding programs in supported languages (i.e. Javascript, Java)
provides dynamic content that interacts with user, modifies
browser interface, access client computer environment

DEMO:

Inspect Element on Browser

Dynamic Web Pages

Add/Interact with static HTML page with code to make it dynamic.

Express web page as a program

I.e. JavaScript

In `<script>` tags

```
<title>Javascript demo page</title>

<font size=30>
Hello, <b>
<script>
var a = 1;
var b = 2;
document.write("world: ", a+b, "</b>");
</script>
```

JavaScript

Powerful web page *programming language*

Embed scripts in web pages returned by web server: `<script>...</script>`

Execute scripts by browser:

- Alter page contents
- Track events & react to them
- Read/set cookies
- Issue web requests, read replies
 - Can navigate to other web page through javascript
 - Can read other contents of replies OUTSIDE page to be loaded

JavaScript

Scripting language interpreted by browser

Define function:

```
<script type = "text/javascript">  
    Function hello() {alert("HELLO!"); }  
</script>
```

Event handlers in HTML:

```
<img src = "pic.gif" onMouseOver="javascript:hello()">
```

Built in functions can change content of window

```
window.open("http://umich.edu")
```

Click-jacking attack (appear trustworthy, acts different than intended)

```
<a onMouseUp = "window.open('http://www.evilsite.com')" href =  
"http://goodsite.com"> Click me! I'm trustworthy! </a>
```

Confining Power of JavaScript Scripts

- don't want a script sent from **hackerz.com** web server to read cookies belonging to **bank.com** ...
- ... or alter layout of a bank.com web page
- ... or read keystrokes typed by user while focus is on a bank.com page!

Security on the Web

As covered in lecture...

RISK: Don't want malicious site accessing, modifying or trashing files/programs on my computer, infecting with malware, etc. when browsing

DEFENSE: Javascript is sandboxed

- Avoid security bugs in browser code

- Privilege separation

- Automatic Updates

Security on the Web

RISK: Don't want malicious site spying/tampering with information or interactions with OTHER websites

I.e. Browse to evil.com should not let it spy on email in Gmail, or buy stuff with my Amazon account open on another tab

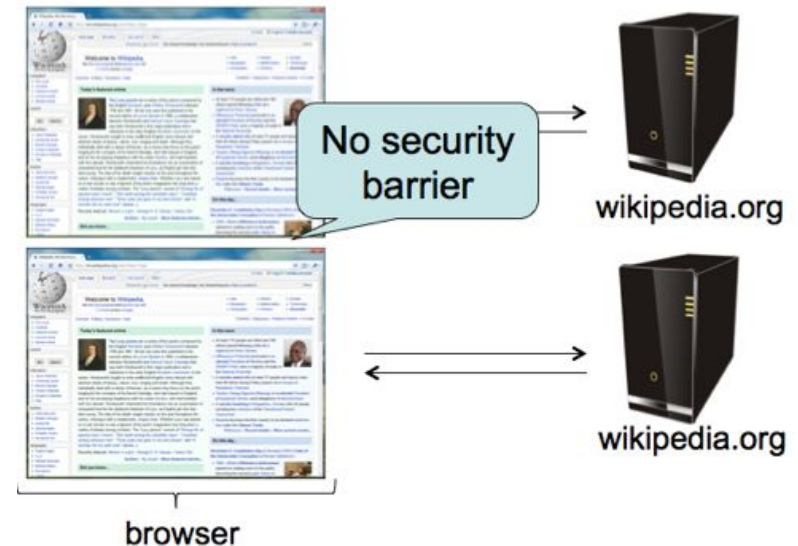
DEFENSE: **same-origin policy** (isolate web sites from others)

- Is a policy on the browser to not access DOM Tree of other sites

Same-Origin Policy

Isolate each site from others

Multiple pages on SAME site (domain/origin) aren't isolated



Same-Origin Policy

- Origin = protocol + hostname (+ port) must ALL be same
- Javascript on one page can read, change, interact with all pages with same origin
- XSS: Subverts SOP

I.e. Which can communicate?

http://coolsite.com:80

https://coolsite.com

http://coolsite.com/tools/info.html

<http://coolsite.com/users>

Security on the Web

RISK: Want data stored on web server protected from unauthorized access

DEFENSE: server-side security

I.e. sql injection

Intro to Checkpoint

CP1: Bungle

<http://trurl.cs.illinois.edu/>

This web application which has following capabilities.

Search: makes a query through GET request

Login: makes a POST request

Logout (enabled when logged in): makes a POST request

Create account: makes a POST request

DIFFERENT FROM BEFORE: we are defending against vulnerabilities, not attacking for this checkpoint.

Components of Checkpoint 1

4.1.1: Setup Bungle

- Setup mysql, access to Bungle site
- Will be implementing Bungle's components

4.1.2: SQL

- Create database (more setup)
- Users and History (search history) table

ANNOUNCEMENT: CP2 will not be autograded as with previous MPs.

Components (cont.)

4.1.3: Prepared Statements

- Implement API to process user input to SQL queries (connect frontend to backend)
- protects **SQL injection**

4.1.4: Input Sanitation

- Filter **XSS** by modifying inputs to remove executable code (simple version)

4.1.5: Token Validation

- Protect **CSRF** by checking client's cookie

SQL

- Structured Query Language: interact with database systems
- CRUD: create, read, update, delete operations on data
- On Web, user doesn't execute SQL directly, is done in backend. Takes input and use in part of SQL query on remote server

I.e. Web Form asks "First Name": _____

Back end: INSERT [value from form] INTO Users

- SELECT [*, fields] FROM [Table(s)] WHERE [query specifications]

SQL Injection

Use poorly written/secured SQL to inject and execute arbitrary SQL code



What does this code do?

```
SELECT first_name, last_name FROM users  
WHERE user_id = '$id'
```

```
SELECT first_name, last_name FROM users  
WHERE user_id = ' ' OR '1'='1 '
```

SQL Injection Protection

Webpage which escapes each ' from input to \' using mysql_real_escape()

<http://www.sqlinjection.net/advanced/php/mysql-real-escape-string/>

- Only helps when input parameter enclosed in quotes
- `SELECT * FROM table WHERE name = ' $_GET['name'] '`
- `SELECT * FROM table WHERE id=$_GET['id']`

Which one works and why??

Prepared Statements (PHP)

```
$conn = new mysqli($servername, $username,  
$password, $dbname);
```

```
//prepare and bind
```

```
$stmt = $conn->prepare("SELECT * FROM table  
WHERE name=?");
```

```
$stmt->bind_param("s", $name);
```

```
//execute
```

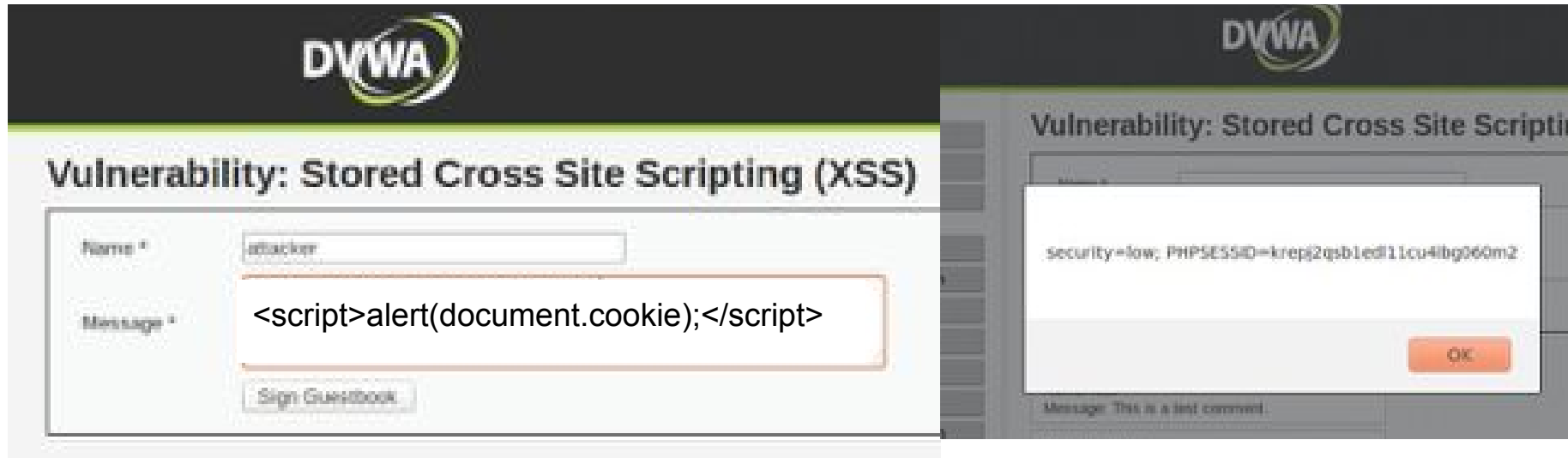
```
$stmt->execute();
```

<http://www.guru99.com/learn-sql-injection-with-practical-example.html>

XSS

Cross-site Scripting

Bypass SOP: attack uses web app to inject client side script into browser



The image displays two screenshots of the DVWA (Damn Vulnerable Web Application) interface, illustrating a Stored Cross Site Scripting (XSS) attack.

Left Screenshot: The page title is "Vulnerability: Stored Cross Site Scripting (XSS)". The "Name" field contains the text "attacker". The "Message" field contains the malicious script: `<script>alert(document.cookie);</script>`. A "Sign Guestbook" button is visible below the message field.

Right Screenshot: The page title is "Vulnerability: Stored Cross Site Scripting". A JavaScript alert box is displayed, showing the session ID: `security=low; PHPSESSID=krepj2qsb1edf11cu4ibg060m2`. An "OK" button is present on the alert box. Below the alert box, a message states: "Message: This is a test comment."

CSRF

Cross-Site Request Forgery

- Malicious entity causes user's browser to perform unauthorized action on webpage

I.e. Clicking link on external webpage send post request changing Facebook password

Protection: check cookie of accessor, some “token” that must match



`.../?password_new=<password>&password_conf=<password>&Change=Change#`

Questions?

