CORS Kung fu

$whoami
adityabalapure

- aka Adi
- Security guy @ Grubhub
- Builder, Breaker, Defender
- Author
- Tweet Tweet @adityabalapure
- OWASP Member: aditya.balapure@owasp.org

Why are we here

- Unravel mysteries about Cross-Origin Resource Sharing (CORS)
- Common Developer Mistakes
- Attacks in the wild
- Beer of course!

CORS or did I say coarse?

“A resource makes a **cross-origin HTTP request** when it requests a resource from a different domain than the one which the first resource itself serves.”

https://developer.mozilla.org/en-US/docs/Web/HTTP/Access_control_CORS

SAME ORIGIN 101

Pages are under same origin if the protocol, port and host match

Example URL: http://abc.example.com

http://xyz.example.com - Different Origin

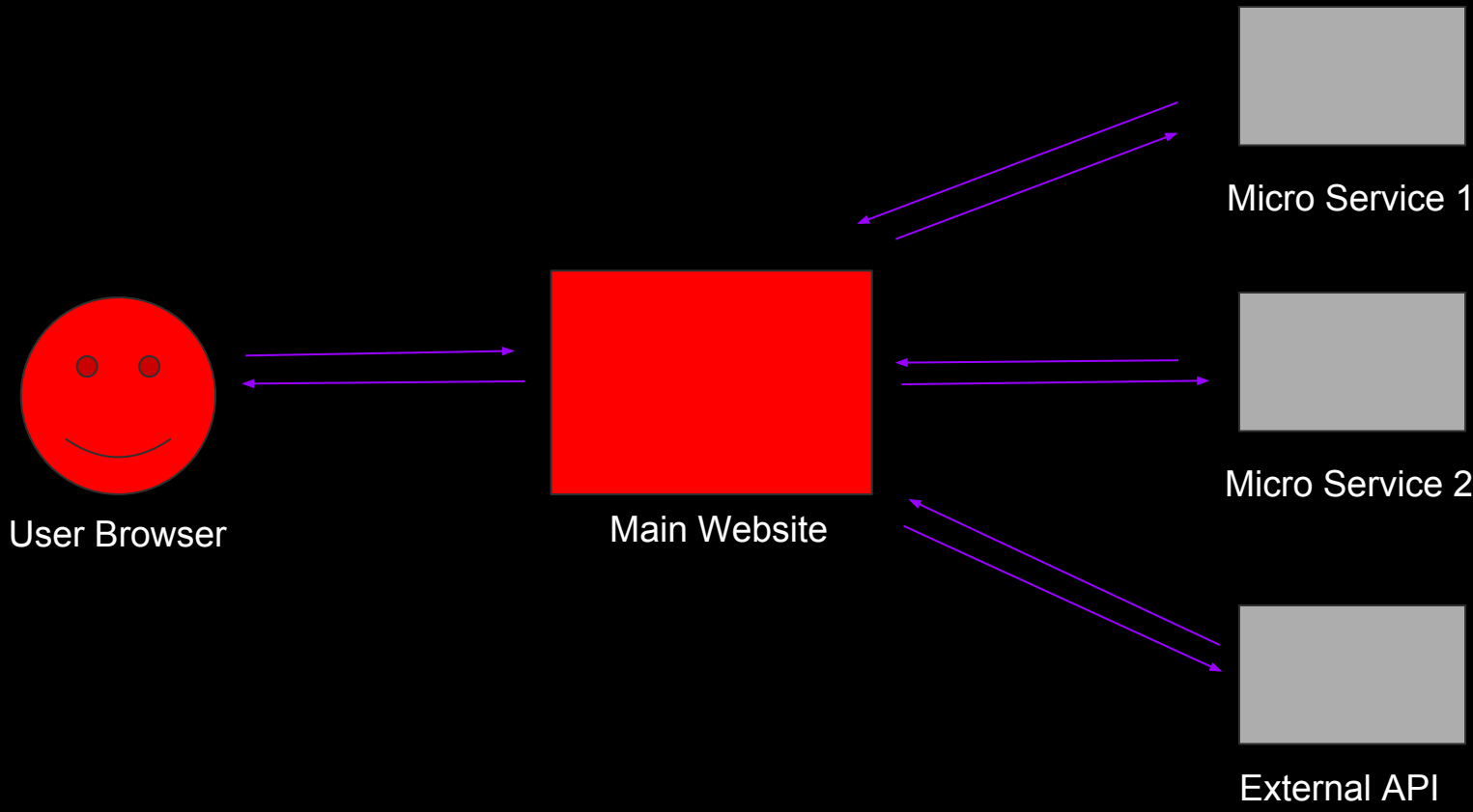
https://abc.example.com - Different Origin

http://abc.example.com:8080 - Different Origin

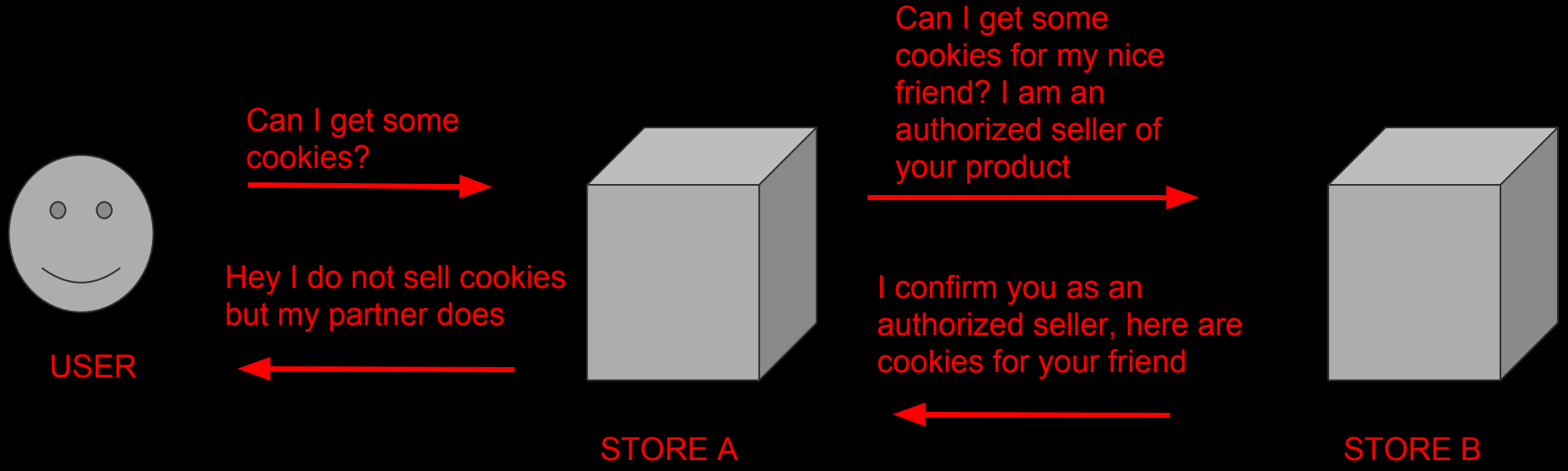
http://abc.example.com/abc/xyz.html - Same Origin

Why Change Origins?

- Document.domain restrictions
- Companies moving to micro services architecture
- More and more dependence on external APIs


Micro Service 1
Micro Service 2
User Browser
Main Website
External API

So how does it work?



CORS Request

- Simple
- Complex

Simple

Request

GET /getmystuff HTTP/1.1
Host: abc.example.com
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.11; rv:49.0) Gecko/20100101 Firefox/49.0
Accept: text/html
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Referer: https://www.example.com/
Content-Length: 134
Origin: https://www.example.com
Connection: keep-alive

Response

HTTP/1.1 200 OK
Access-control-allow-origin:*
Date: "Fri, 11 Nov 2016 21:41:54 GMT"
Content-Length: 5
Connection: Close

Why Preflight?

- Complicated requests usually containing methods other than GET, HEAD, POST
- Different Content-Type let’s say application/json
- If any custom headers need to be sent

CORS Jargon

Access-Control-Allow-Methods

Access-Control-Allow-Credentials

OPTIONS

Access-Control-Allow-Headers

Access-Control-Allow-Origin

Preflight

Access-Control-Request-Method

Access-Control-Request-Headers

Time to dive in!

REQUEST

```
OPTIONS /authenticate HTTP/1.1
Host: abc.example.com
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.11; rv:49.0) Gecko/20100101 Firefox/49.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Access-Control-Request-Method: POST
Access-Control-Request-Headers: authorization,cache-control,content-type
Origin: https://www.example.com
Connection: close
```

Response from abc.example.com

HTTP/1.1 200 OK
Access-control-allow-credentials: true
Access-control-allow-headers: Allowed headers
Access-control-allow-methods: HEAD,DELETE,POST,GET
Access-control-allow-origin: https://www.example.com
Access-control-max-age: 1800
Date: "Fri, 11 Nov 2016 21:41:54 GMT"
Content-Length: 0
Connection: Close

Request

POST /authenticate HTTP/1.1
Host: abc.example.com
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.11; rv:49.0) Gecko/20100101 Firefox/49.0
Accept: application/json
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Authorization: Bearer 33434-234hj3-434jjk-2p0r33
Content-Type: application/json
Cache-Control: no-cache
Referer: https://www.example.com/
Content-Length: 134
Origin: https://www.example.com
Connection: close

Hey Please authenticate me with these credentials (username,password)

Response

HTTP/1.1 200 OK
Access-control-allow-credentials: true
Access-control-allow-headers: Allowed headers
Access-control-allow-origin: https://www.example.com
Access-control-expose-headers: example,Content-Length,Location
Cache-Control: "private, no-cache, no-store, no-transform, max-age=0"
Content-Type: application/json
Date: "Fri, 11 Nov 2016 21:42:52 GMT"
Content-Length: 2113
Connection: Close

YOU ARE ALL SET! HERE IS YOUR DATA

How do I look

```
var request = new XMLHttPRequest();

var url = ‘https://abc.example.com/authenticate’;

function myFunc()

{

        if(request)

        {

                request.open('POST'',url, true);

                request.onreadystatechange = handler;

                request.send();

    }
```

How it works with Google APIs

```
var user = gapi.auth2.getAuthInstance().currentUser.get();

var oauthToken = user.getAuthResponse().access_token;

var xhr = new XMLHttpRequest();

xhr.open('GET', 'https://people.googleapis.com/v1/people/me/connections' +

  '?access_token=' + encodeURIComponent(oauthToken.access_token));

xhr.send();
```

https://developers.google.com/api-client-library/javascript/features/cors

Developer Mistakes - Attack Scenario 1

- Lesson: The Importance of * wildcard
  - Risk of an attacker accessing resources
  - Any protected resources?
  - Need of authN/authZ
  - Cross site request forgery attacks for elevated operations

Example

HTTP/1.1 200 OK
Access-control-allow-origin: *
Access-control-expose-headers: example,Content-Length,Location
Cache-Control: "private, no-cache, no-store, no-transform, max-age=0"
Content-Type: application/json
Date: "Fri, 11 Nov 2016 21:42:52 GMT"
vary: origin
Content-Length: 2113
Connection: Close

S3 Example

```
<CORSConfiguration>
 <CORSRule>
   <AllowedOrigin>http://www.example1.com</AllowedOrigin>

   <AllowedMethod>PUT</AllowedMethod>
   <AllowedMethod>POST</AllowedMethod>
   <AllowedMethod>DELETE</AllowedMethod>

   <AllowedHeader>*</AllowedHeader>
 </CORSRule>
 <CORSRule>
   <AllowedOrigin>http://www.example2.com</AllowedOrigin>

   <AllowedMethod>PUT</AllowedMethod>
   <AllowedMethod>POST</AllowedMethod>
   <AllowedMethod>DELETE</AllowedMethod>

   <AllowedHeader>*</AllowedHeader>
 </CORSRule>
 <CORSRule>
   <AllowedOrigin>*</AllowedOrigin>
   <AllowedMethod>GET</AllowedMethod>
 </CORSRule>
</CORSConfiguration>
```

http://docs.aws.amazon.com/AmazonS3/latest/dev/cors.html

Developer Mistakes - Attack Scenario 2

- Understanding credentialized requests
  - Cookies?
  - Protected resources?

Credentialized Request

POST /authenticate HTTP/1.1
Host: abc.example.com
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.11; rv:49.0) Gecko/20100101 Firefox/49.0
Accept: application/json
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Content-Type: application/json
Cache-Control: no-cache
Content-Length: 134
Origin: https://www.evilhost.com
Connection: Keep-Alive
Cookie: Token=3263TE23-3232-TRW-435

Response

HTTP/1.1 200 OK
Access-control-allow-credentials: true
Access-control-allow-headers: Allowed headers
Access-control-allow-origin: https://www.evilhost.com
Access-control-expose-headers: example,Content-Length,Location
Cache-Control: "private, no-cache, no-store, no-transform, max-age=0"
Content-Type: application/json
Date: "Fri, 11 Nov 2016 21:42:52 GMT"
Content-Length: 2113
Connection: Close

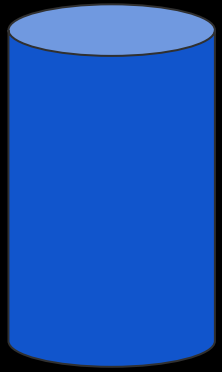
Here are your secrets! Enjoy

Here’s how I look

User Browser

Loads bad JS

Evil Host



Super Secret Server

Sends a CORS request to Super Secret Server

Hey Smiley Face, here are your secrets!

Thank You!

Developer Mistakes - Attack Scenario 3

- Origin Mess-ups
  - Specifying the right Origin
  - Always validate the Origin
  - Never blindly trust the origin header

http://www.kameronhurley.com/wp-content/uploads/2014/09/thankyou.jpg

References

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

http://blog.portswigger.net/2016/10/exploiting-cors-misconfigurations-for.html

https://ejj.io/misconfigured-cors/

https://developer.mozilla.org/en-US/docs/Web/HTTP/Access_control_CORS

https://www.eriwen.com/javascript/how-to-cors/