# IDOR: A complete guide to exploiting advanced IDOR vulnerabilities

BY BLACKBIRD-EU · JUNE 25, 2024 · LAST UPDATED ON MARCH 10, 2025

IDOR —short for insecure direct object reference— vulnerabilities are one of the most commonly found web security vulnerabilities in modern web applications and APIs. It is no wonder that they are often recommended to new bug bounty hunters who are just starting as they are easy to spot and exploit and are by nature high-severity vulnerabilities.

In this article, we will go over how to identify IDOR vulnerabilities and how you can exploit them as well. We will also be covering some advanced cases as well. Let's start with defining what IDOR vulnerabilities are.

## What are IDOR vulnerabilities?

Insecure direct object reference (IDOR) vulnerabilities arise when for example a web app or an API takes user input to directly reference a data object. The data object can be anything, from sensitive fields that are stored in a database to files stored in a storage bucket.

IDOR vulnerabilities are caused due to a lack of access control validation. In other words, the web app or API fails to check if the requester is indeed the righteous owner of the data object. Because of this, the vulnerable component returns the data object even if the user is in theory not allowed to do so.

Depending on the vulnerable component, IDOR vulnerabilities can often result in exposing sensitive data and making or introducing unwanted changes such as modifying or deleting data fields in a database.

**Prefer a video instead? Check out our video that we've made just for you: IDOR in 100 seconds!**

## Identifying IDOR vulnerabilities

In order for a component, web app or API to be susceptible to IDOR vulnerabilities, we must identify a way to directly reference an object. This is often done using a unique identifier. Although it is a best practice for developers to always make use of unpredictable identifiers, you will still come across targets that do not follow these practices. Instead, these targets still make use of predictable IDs such as numerical IDs.

The component must also be performing a state-changing action or retrieve data that is not otherwise available to the public. For example, being able to view public blog posts or public comments can obviously not be considered as an IDOR vulnerability.

You may have heard this before as a new bug bounty hunter that IDOR vulnerabilities are easy to spot. However, if you have spent time on a target looking for IDOR vulnerabilities, you will notice the opposite.

You may ask yourself now where the common thought exactly came from that claims IDOR vulnerabilities are easy to find... That's more likely because experienced bug bounty hunters have obviously more

experience in identifying them and know exactly where to look for these types of vulnerabilities.

The key here is to perform better content discovery and test features that have not been tested by many others before. An example would be instead of testing the send messaging functionality, to test the auto-saving or the component that marks your message as a draft.

The more time you spend on your target, the more you will come across less-tested or untested features and functionalities that are more susceptible to IDOR attacks.

# 1) Exploiting basic IDORs

Basic IDOR vulnerabilities are where we can easily modify a predictable ID such as a numeric integer value with another numerical ID, as shown in the example below:

```
GET /api/users/1234/profile HTTP/2
Host: example.com
Authorization: Bearer eyJ...
User-Agent: ...
```

```
HTTP/2 200 OK
Server: Nginx
Content-Type: application/json
Content-Length: 23

// Data of user ID 1234
```

```
GET /api/users/1235/profile HTTP/2
Host: example.com
Authorization: Bearer eyJ...
User-Agent: ...
```

```
HTTP/2 200 OK
Server: Nginx
Content-Type: application/json
Content-Length: 23

// Data of user ID 1235
```

In this case, the API endpoint would allow us to change the email of our second test account. It can be as easy as changing IDs sometimes, but it can get more complex.

Let's go through some more advanced cases...

# 2) Exploiting IDORs via parameter pollution

In black box pentesting, we must consider every possibility. Parameter pollution should also be tested, depending on the underlying service or technology that handles input parameters. It can be that:

- both values get concatenated
- only the first value gets processed
- or, only the last value gets processed

Parameter pollution is a known method to help evade certain checks such as authorization checks.

```
GET /api/profile?userId=1234&userId=1235 HTTP/2
Host: example.com
Authorization: Bearer eyJ...
User-Agent: ...
```

```
HTTP/2 200 OK
Server: Nginx
Content-Type: application/json
Content-Length: 23

// Data of user ID 1235
```

# 3) Exploiting IDORs via JSON globbing

Similarly, just as above, we need to try out several different methods to try and evade any access control checks. If your target accepts a JSON body, we can play around with the different fields and see how the endpoint handles our input.

Again, depending on how our input is processed, we can introduce unwanted behavior by replacing our ID with:

- an array of IDs: **[1234, 1235]**
- a boolean value: **true/false** (be careful when testing)
- a wildcard character such as an asterisk symbol (*) or percentage sign (%) ( again, be careful when testing)
- a large integer value by appending zeros in front of our ID: **00001235**
- a negative ID: **-1**
- decimal number: **1235.0**
- string value with an added delimiter: **"1234,1235"**

```
PUT /api/users/profile HTTP/2
Host: example.com
Authorization: Bearer eyJ...
Content-Type: application/json

{
   "userID": 1234,
   ...
}
```

```
PUT /api/users/profile HTTP/2
Host: example.com
Authorization: Bearer eyJ...
Content-Type: application/json

{
   "userID": [
        1234,
        1235
   ],
   ...
}
```

# 4) Exploiting method-based IDORs

Sometimes, by simply changing the request method, our request will get processed differently. If access control checks are missing, it may allow us to perform state-changing actions or retrieve sensitive data on another user's behalf.

Let's take a look at a small example to further help us understand this type of IDOR:

app.js

```
const express = require('express');
const app = express();
const port = 3000;

...

app.get('/api/users/:userId/profile', (req, res) => {
    const { userId } = req.params.userId;

    // Function call to simulate access verification
    if (verifyAccess(req.token, userId)) {
        // OK: Return Profile
        getProfile(userId);
    } else {
        // Unauthorized: Deny Request
    };

    res.send(...);
});

// New API endpoint
app.post('/api/users/:userId/profile', (req, res) => {
    const { userId } = req.params.userId;

    getProfile(userId);

    res.send(...);
});

...

app.listen(port, () => {
    console.log(`API listening on port ${port}`);
});
```

As you can see in the code snippet above, there are 2 API endpoints defined. Including a new one that is missing access control checks and is only accessible by changing our request method to POST. This would allow us to retrieve the sensitive data of any user without being authorized to do so.

To successfully exploit this case, we could simply send a POST request as displayed below:

```
POST /api/users/1235/profile HTTP/2
Host: example.com
Cookie: sess_id=...
User-Agent: ...
```

```
HTTP/2 200 OK
Server: Express
Content-Type: application/json
Content-Length: 23

// Details of user ID 1235
```

## 5) Exploiting content-type-based IDORs

Similarly, just as in the previous case, it can also be possible that our request gets processed differently by an underlying framework or library and allows us to again perform state-changing actions or retrieve sensitive data just by setting a different content-type header.

## 6) Exploiting IDORs via depreciated API versions

Targets that provide their API for the public will often make use of a versioning system. Each new version often comes with newly added capabilities and also security patches. If older versions are still accessible and have missing access control checks, it may still allow you to, for example, retrieve sensitive data from an API even though the latest version already resolved this security issue.

```
GET /api/v2/users/1235 HTTP/2
Host: example.com
Cookie: sess_id=...
User-Agent: ...
```

```
HTTP/2 401 Unauthorized
Server: Apache
Content-Type: application/json
Content-Length: 18

{"success": false}
```

```
GET /api/v1/users/1235 HTTP/2
Host: example.com
Cookie: sess_id=...
User-Agent: ...
```

```
HTTP/2 200 OK
Server: Apache
Content-Type: application/json
Content-Length: 23

// Data of user ID 1235
```

## 7) Exploiting IDORs in APIs that use static keywords

Sometimes developers make use of keywords like "current" or "me" to reference back to the current user. Now, since this sort of goes against the rules of identifying potential IDOR vulnerabilities (as they require an ID)... However, it often happens that the API or component you're testing also supports requests with your (numerical) user ID.

In other words, you can easily swap the current keyword with your user ID and test for IDOR vulnerabilities again. Just you can see in the figure below:

```
GET /api/users/me/profile HTTP/2
Host: example.com
Cookie: sess_id=...
User-Agent: ...
```

```
HTTP/2 200 OK
Server: Apache
Content-Type: application/json
Content-Length: 38

// Data of user ID 1234 (current user)
```

```
GET /api/users/1235/profile HTTP/2
Host: example.com
Cookie: sess_id=...
User-Agent: ...
```

```
HTTP/2 200 OK
Server: Apache
Content-Type: application/json
Content-Length: 23

// Data of user ID 1235
```

# 8) Exploiting IDORs that require unpredictable IDs

More developers start to use UUIDs or other unpredictable hashes simply because they aren't easily predictable and are prone to bruteforcing attacks. However, there's still hope left as there are several ways we can use it to our advantage to enumerate these IDs.

One way is to request another endpoint to return back any references with the IDs that we're looking for. Check out this quick tip shared by one of our community bug bounty hunters!

> **"Ever got an IDOR rejected because it needed a UUID?**
>
> **There's a chance left to elevate the severity @hetmehtaa knows a trick #bugbounty #bugbountytips pic.twitter.com/n10W1eaS0X**
>
> **— Intigriti (@intigriti) February 5, 2022"**

And there are several other ways to find references to IDs, such as:

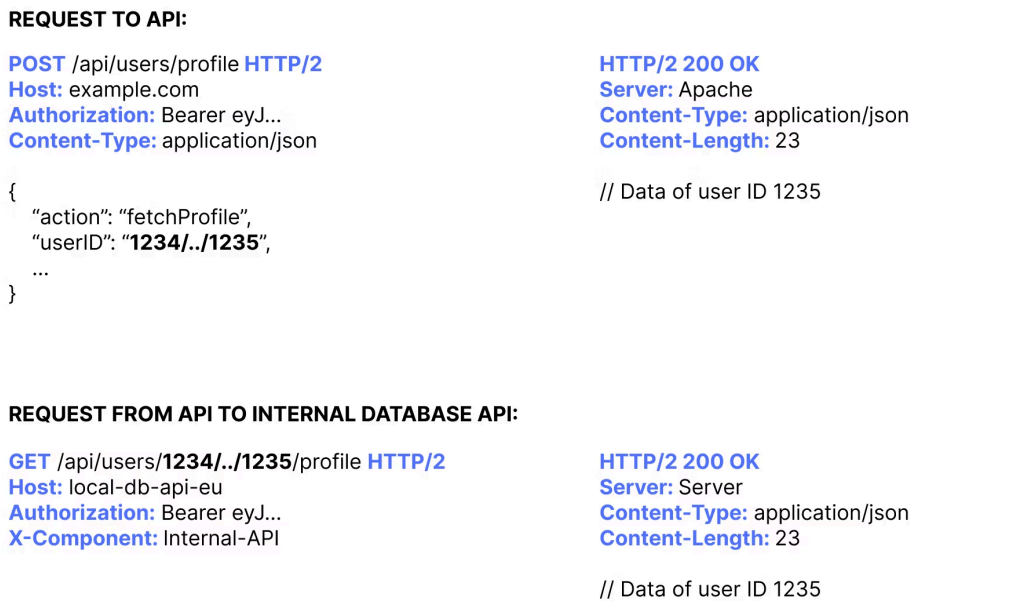
- public profiles (such as profile pictures)
- sign in/signup & password reset forms
- in-app sharing links
- email unsubscribe forms
- in-app messages
- Wayback Machine
- search engines (like Google and Bing)

# 9) Exploiting second-order IDOR vulnerabilities

Second-order IDORs are just like IDOR vulnerabilities but the only difference here is that the vulnerable component utilizes your input to indirectly reference a data object. Your ID gets stored first and then retrieved to reference an object later on.

Second-order IDOR vulnerabilities are more complex and trickier to find. An example would be a scheduled data exporting feature. First, a schedule gets created whereby your user ID gets saved in the external scheduling service. Once the schedule gets triggered, it later retrieves your user ID from the metadata to generate an export. This second step often does not go through any additional access control checks. We can make use of this exploitable behavior and try to generate an export of our second test account's data instead.

There are several ways to do so just as we've mentioned earlier in this article. In the following figure, you can see a simple example of a second-order IDOR vulnerability:




1. API failed to validate user input and forwarded it to the internal API
2. Database API receives the second test account's ID and misses authorization checks

# Conclusion

IDOR vulnerabilities can be easy to find if you know where to look for them. By nature, these are also high-severity vulnerabilities and this often goes paired with higher bounties.

So, you've just learned something new about IDOR vulnerabilities... Right now, it's time to put your skills to the test! Browse through our 70+ public bug bounty programs on Intigriti and who knows, maybe you'll earn a bounty with us today!

START HACKING ON INTIGRITI TODAY