011. What the IDOR?

Introduction

When we are hacking for BAC, we can apply some general tips, and the same goes for IDORs. These won't be specific technical things we should be doing but they do are certainly a handy guide to follow a bit.

IDORs, let's first explain what they are before deep diving into how to find them, and believe me that it will be deep. Insecure Direct Object References consist of 2 things, we have our direct object reference which means as much as id=1. We are directly pointing to an object and this can be anything. It can be an invoice, address, credit card,... The insecure part references the fact we can sometimes access objects that are not supposed to be accessed by you. If these conditions are met we speak of an IDOR. But how does the server know what you should access and what not? Let's get right into it!

Authentication vs authorization

We should make the distinction between authentication and authorization. As a user, I can either be authenticated or unauthenticated. This means that I can be logged in or not and the authentication part refers to that I authenticated myself with my username and password or any other security system such as biometrics or a PIN code. I can also be authorized which means that the server will allow me to perform an action (like grabbing an object's details).

Attack scenario's

The reason I want to talk about the attack scenarios is that they can be pretty diverse as you saw above! First of all, we can try to access an object unauthenticated, which means not logged in, and try to go to a URL like GET /invoices?id=432. Please note i only talk about GET parameters here but it can also be a POST request or a PUT or DELETE and even PATCH request as well. as long as an object identifier can be used, it might be vulnerable to IDORs.

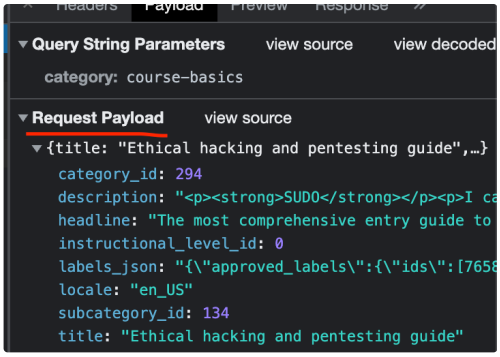
Besides this, we might also be plagued by authenticated IDORs where we are logged in and trying to actually grab a resource we should be able to access like /invoices but we are trying to grab an object on the resource that is not accessible to us normally.

Now that we are logged in however we need to define another attack scenario that has been rising due to companies sharing a server between multiple clients. In some applications like HR applications or invoicing applications that are marketed to businesses (b2b). These applications have two different attack scenarios:

- IDORs between employees of the same company
- IDORs between two companies

GET vs POST vs cookies vs headers IDOR

Of course, we have to be mindful of the fact and ID parameter can be hidden anywhere within the application and as stated before, not just in the GET parameters of a request. Those are the URL parameters, marked by the question mark: ?param1=value&param2=value. We can also have our luck in other parts of the application such as the POST parameters



Or even in the headers such as cookies.

Tip: Go look for IDOR in POST, GET parameters, headers of a request, and file imports! For example, you might not be able to edit my products, but while importing you may be able to include an ID parameter for the products which edits mine anyway.

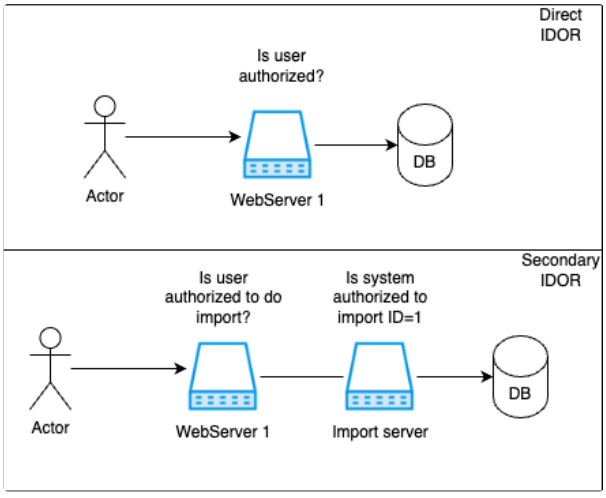
objectID or userID based

We also have to make a distinction between object and user-based IDs. What we mean by that is that sometimes we can reference an object by its ID directly (invoiceID=1) or we can reference the user to which the object should belong. (invoices.php?owner=123)

These two don't require any different way of hunting but it is important to realize whether we are referencing an object directly or whether we are indirectly referencing the object. This will come in handy in one of the next chapters: Multi-factor IDORs!

Secondary vs primary IDOR

This is one of the most overlooked aspects of IDORs. You might be wondering what a second order IDOR is. To explain this, we first need to explain you how applications pass along authorization. Sometimes we have a direct reference to a system that automatically checks the correct authorization and sometimes a second system comes into play:



This matters a lot because developers might forget to pass over your authorization tokens and inherently trust the system. That is a good striking point for us as you can imagine!

Multi-tenancy

Now that we have a grip on these concepts, the last one I need to introduce during this course is multi-tenancy. It sometimes just does not make economic sense to have separate server space for every customer and sometimes, multiple customers are put onto 1 system. This can help keep costs lower but increases any security requirements as you might be able to imagine. After all, if we have multiple clients on

1 system, the possibility of IDORs between the clients appears. This is very bad as it can enable company espionage and nobody wants to work with a company that has had a severe breach like that without extreme restructuring which is why it is vital to keep the different clients separated.