BAC

What is it

BAC: Broken Access Control

Broken Access Control is exactly as it states. It can manifest itself in both horizontal and vertical privilege escalation.

When we speak about privilege escalation, we are talking about the ability to execute functionality we are not supposed to be able to execute. But what exactly do we mean by verticale and horizontal?

If that functionality is something we can execute, but not with the given data, we are talking about horizontal priviledge escalation. A good example of this would be IDORs. If on the other hand, we can not execute that functionality at all, we are talking about Verticale priviledge escalation.

We can also give a few examples of this behaviour but first we need to define some properties. To speak of BAC, we need to have the following conditions met:

- We need to be able to create different users OR need to be assigned different users
- The users need to be of different privilege levels if we want to test for verticale privilege escalation. For example an admin and a normal user
- There need to be functions that our low privilege user is not authorised to execute, either with the given data or at all

Some examples:

- We have an admin and a normal user. We can test the admin settings with the low priv user
- We have a normal user and a prospect user. The prospect user can not execute all the functions because he only has a trial account
- We have two users of the same authorisation level: See IDOR

Attack strategy

Again we can test this manually or we can test this semi automatic. I do not believe robots have the ability to execute complex stateful scenario's where often sequential steps rely on previous input.

Manually

To test for BAC manually, we have several options at our disposal. One of the most powerful tools we have is javascript and the developer console of the browser.

JavaScript files will contain functions that can be executed. One common tactic developers use to protect their programs is to simply disable the UI elements. This means that the javascript functions might still be executable. If that is the case, say for example if we can not click on the UI element to open the invoices BUT we might be able to execute the javascript function which prints the invoice details, we have a BAC on our hands.

You may have noticed this requires some knowledge of how your target works because you need to know which levels of authorisation exist and what those types of accounts can and can't do. You will also need to be able to recognise what the different javascript functions might do.

We talked a bit about the developer console of the browser as well because there is a tab called "console" which will let you execute any javascript and it has Intellisense. This means you can type any letter and the console will automatically search through all the javascript files that contain that letter and display them to you. On top of the previously discussed advantages, we can also execute our javascript functions and directly observe the result.

We can also take a more direct approach and simply log in as a high privilege user, navigate to functionality a low priv user can not execute, copy the URL, log in as

the low priv user and paste the URL in the browser. A few caveats however are that we again need to be farmiliar with the levels of authorisation the program supports and what those types of users can and can not access.

A last strategy we can take is bordering automation and it would be the first steps we need to prepare for the next chapter.

We can also try and send all of our requests that low priv users can not execute to our repeater in the MiTM proxy such as burp and replace the authorisation method (such as JWT or session cookies, see IDOR). The point is not to grab the low priv users authorisation headers by the way, it is to test for broken access control issues.

Semi-automated strategy

We can use the same strategy we used as we did for IDORs here. Either the burp extension Authorize or match and replace can help us here. Please refer to Tools > Burp: Authorize and Burp: Match and replace