Internet-Scale analysis of AWS Cognito Security

Sector.ca 2019

@AndresRiancho
Patient zero

Cloud Security Assessment
Stumble upon AWS Cognito

During a white-box **cloud security assessment** I used my **ReadOnly permissions** and the CloudTrail logs to enumerate all AWS services used in the account.

Cognito appeared in the list.

**Had no idea what it was**
Fell in love

Read the documentation and found...

Identity pools enable you to **grant your users access to AWS services**.
Full AWS account compromise

Using AWS Cognito misconfigurations I was able to compromise the AWS account in four steps.

1. Read the identity pool ID from the AWS console

```java
// Initialize the Amazon Cognito credentials provider
CognitoCachingCredentialsProvider credentialsProvider =
        new CognitoCachingCredentialsProvider()
        .getClient()
        .getSession()
        .getCredentials()
        .getProvider("us-east-1:8a7e1fc4-c70a-4ff0-83c0-fd547f988c8a", /
        Regions.US_EAST_1 // Region
);```
2. Write **boto3** code to get AWS credentials from the identity pool

```python
def get_pool_credentials(region, identity_pool):
    client = boto3.client('cognito-identity', region_name=region)
    _id = client.get_id(IdentityPoolId=identity_pool)
    _id = _id['IdentityId']

    credentials = client.get_credentials_for_identity(IdentityId=_id)

    access_key = credentials['Credentials'] ['AccessKeyId']
    secret_key = credentials['Credentials'] ['SecretKey']
    session_token = credentials['Credentials'] ['SessionToken']
    identity_id = credentials['IdentityId']

    return access_key, secret_key, session_token, identity_id
```
Privilege escalation

3. **Enumerated permissions** for the credentials

4. Escalated privileges to **full account compromise** using unnecessary Lambda Function permissions

Reported this vulnerability to my customer as: Least privilege principle not used in unauthenticated Cognito role.
Got one. Can I get them all?

During this cloud security assessment I identified and exploited one instance of Cognito misconfiguration.

A quick online search showed that there was no previous security research on AWS Cognito.
How many AWS accounts are at risk?

Which are the most common and insecure permissions granted by developers?

Is it possible to perform an Internet-Scale analysis of AWS Cognito security?
What's next

1. Understand Cognito and it's vulnerabilities
2. Grep the Internet
3. Statistics
4. Root cause analysis
Introduction to AWS Cognito

Granting end-users access to AWS
What Is Amazon Cognito?

Amazon Cognito provides authentication, authorization, and user management for web and mobile apps.

The two main components of Amazon Cognito are:

- **User pools** are user directories that provide sign-up and sign-in options for your app users
- **Identity pools** enable developers to grant end users access to AWS services
Amazon Cognito use case

**CoolCatPics mobile application** wants to allow users to upload pictures directly to S3 and the associated metadata to be stored in DynamoDB.

The application will authenticate users with a Cognito user pool and Facebook authentication.

**Authenticated** users can write to the S3 bucket and DynamoDB, **unauthenticated** users can only list and view S3 bucket contents.
Amazon Cognito from mobile apps

AWS Cognito **identity pool provides users with AWS credentials** to consume S3 and DynamoDB.

The mobile application uses the [AWS SDK for Android](https://aws.amazon.com/sdk-for-android/) or [iOS](https://aws.amazon.com/sdk-for-ios/) to interact with Cognito and once the credentials have been obtained consume the S3 and DynamoDB APIs.
CoolCatPics wants to have a web client for their application.

In this scenario the same services are used: Cognito, S3 and DynamoDB, but the mobile application is replaced by the user's browser.

AWS API calls are sent directly from the browser using the AWS JavaScript SDK.
Create new identity pool

Identity pool name: andresrancho_001

Unauthenticated identities

Amazon Cognito can support unauthenticated identities by providing a unique identifier and AWS can enable access for unauthenticated identities. Learn more about unauthenticated identities.

[ ] Enable access to unauthenticated identities

Authentication providers

Amazon Cognito supports the following authentication methods with Amazon Cognito Sign-in or Changing the application ID that your identity pool is linked to will prevent existing users from aut

- Cognito
- Amazon
- Facebook
- Google+
- Twitter / Digits
- OpenID

Facebook App ID: Optional
Example: 734624159893555
Assign IAM roles to identities

Role Summary
- **Role Description**: Your authenticated identities would like access to Cognito.
- **IAM Role**: Create a new IAM Role
- **Role Name**: Cognito_andresriancho_001Auth_Role
  - View Policy Document

Role Summary
- **Role Description**: Your unauthenticated identities would like access to Cognito.
- **IAM Role**: Create a new IAM Role
- **Role Name**: Cognito_andresriancho_001Unauth_Role
  - View Policy Document
IAM policy example

```json
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ListObjectsInBucket",
            "Effect": "Allow",
            "Action": ["s3:ListBucket"],
            "Resource": ["arn:aws:s3:::bucket-name"]
        },
        {
            "Sid": "AllObjectActions",
            "Effect": "Allow",
            "Action": "s3:*Object",
            "Resource": ["arn:aws:s3:::bucket-name/*"]
        }
    ]
}
```
AWS Cognito identity pool is **ready to use**.

Notice that the identity pool identifiers are **randomly generated UUID4**.

This was one of the main problems to solve when trying to perform an Internet-Scale analysis of AWS Cognito security because **only users that know the UUID can interact with the identity pool**.
Internet Scale

Automation, automation, automation
Internet Scale analysis

data = []

for identity_pool_id in get_identity_pools(the_internet):
    credentials = get_pool_credentials(identity_pool_id)
    permissions = enumerate_permissions(credentials)
    score = score_permissions(permissions)

    data_i = (identity_pool, credentials, permissions, score)
    data.append(data_i)

pretty_graphs_and_stats(data)
Challenge #1: Identity Pool UUID4

Identity pool IDs are randomly generated and with enough entropy to discourage brute-force attacks. The solution is to extract these IDs from the client applications:

- Google Play Store
- Web applications
  - Common Crawl
  - GitHub
  - Shodan
  - Zoomeye
  - Google
  - Yandex
The initial plan was to download all apps from Google Play, decompile them and extract identity pool IDs. But Google play has ~2.6M apps and multiple protections against crawlers.

Mobile apps which use the AWS SDK for Android were the most important ones, found a paid service that allows users to search Google play and filter by the libraries inside the APK.
The search results contained 13000 application names (eg. com.whatsapp).

Decided to use alternative sites such as apkpure and apkmirror which are less restrictive with crawlers, this made the download process easier (no bypass of Google Play protections).
Google only indexes text

The goal was to obtain all sites that use the **AWS JavaScript SDK**:

```html
<script src="/path/to/amazon-cognito-identity.min.js"></script>
<script src="/path/to/aws-sdk-2.6.10.js"></script>
```

```javascript
window.AWS.config.region = 'us-east-1';
window.AWS.config.credentials = new 
window.AWS.CognitoIdentityCredentials({
    IdentityPoolId: 'us-east-1:d917fd38-4...
});
```
Google only indexes text

Identity pool IDs, AWS JavaScript SDK methods and classes have very specific patterns if only it would be possible to... grep the web...

egrep -r -i -e '(...)'/huge-network-fs/internet/
I CAN
DO THAT

HOLD
MY BEER
Common Crawl

Common crawl is an open repository of web crawl data that can be accessed and analyzed by anyone.

Stats from the April crawl: 2.5 billion web pages or 198 TB of uncompressed content
**Common Crawl**

`cc-mrjob` is an existing tool that uses AWS Elastic MapReduce and spot instances to query the common crawl data.

Was unable to get good results from it: Stability issues, hard to debug, poor logging, etc.
Created **cc-lambda**, a tool that uses AWS Lambda to parallelize the process of searching through common crawl data.

- 1000 concurrent AWS Lambda functions
- Download warc archive, decompress, search using Python regular expression engine
- Store matches in S3

```
egrep -r -i -e '(...)'/internet
```
$ python cc-lambda.py
Overall progress: 1.55%
Going to process 250 WARC paths
Got futures from map(), waiting for results...
crawl-data/CC-MAIN-2019-09/.../CC-MAIN-20190215183319-20190215205319-00000.warc.gz
  - Time (seconds): 191
  - Processed pages: 44969
  - Ignored pages: 93005
  - Matches: {'aws_re_matcher': 9, 'cognito_matcher': 4}
The Common Crawl data is split in 64k gzip files.

Processing each gzip file takes ~191 seconds.

Used 1000 concurrent lambda functions.

Processed 2.5 billion web pages or 198 TB of uncompressed content in ~3 hours.
**Common Crawl**

*cc-lambda* uses the [pywren](https://pywren.readthedocs.io) library to abstract function return value storage, error handling, timeouts and retries.

The common crawl results are stored in S3 buckets which are part of the Amazon Public Datasets program, there is no cost for downloading all the data, and data transfer between S3 and Lambda is super fast.

**AWS Lambda function cost will be 95% of your bill** when running this tool, the remaining 5% is the cost associated with storing matches in S3.

~300 USD
Other (boring) sources

The research also included other sources, which were easier to consume and returned a very small number of matches:

- GitHub
- Shodan
- Zoomeye
- Google
- Yandex
In the AWS cloud there are two ways to enumerate permissions for a given credential set:

- **Use the IAM service** to get the role's permissions. In most cases this will fail because the role itself has no permission for the IAM API.

- **Call each AWS API** and analyze the response. **Brute-force**
Enumerate permissions and avoid jail time

- Enumerate Get* / List* / Describe*. Try anything else and you might change (break) the target AWS account.

- Never send API calls that disclose user-data such as S3 bucket contents, DynamoDB table contents, etc.
Enumerate Permissions / Performance

There are thousands of API calls, so speed quickly became an issue with existing tools.

Pacu and several other tools and scripts perform permission enumeration but were missing at least one of the required features.

Wrote enumerate-iam and integrated it into the main script

- Threads and AWS service connection pool for performance
- Dynamic test generation based on documentation found in the aws-js-sdk repository
enumerate-permissions.py

$ ./enumerate-iam.py --access-key AKIA... --secret-key StF0q...
[INFO] Starting permission enumeration for access-key-id "AKIA..."
[INFO] -- gamelift.list_builds() worked!
[INFO] -- sqs.list_queues() worked!
...
[INFO] -- ec2.describe_addresses() worked!
This research focuses **only on the unauthenticated roles** associated with Cognito identity pools.

Did not confirm, but common sense indicates that:

\[
\text{privileges(}\text{unauth}\_\text{role}) \leq \text{privileges(}\text{auth}\_\text{role})
\]

The results obtained from this research **would have been much worse** if Cognito's authenticated role would have been part of the analysis.
Data and statistics
Identity pool sources

<table>
<thead>
<tr>
<th>Source</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Google Play</td>
<td>2627</td>
</tr>
<tr>
<td>GitHub</td>
<td>264</td>
</tr>
<tr>
<td>Common Crawl</td>
<td>167</td>
</tr>
<tr>
<td>Yandex</td>
<td>62</td>
</tr>
<tr>
<td>Zoomeye</td>
<td>35</td>
</tr>
<tr>
<td>Google</td>
<td>4</td>
</tr>
<tr>
<td>BuiltWith</td>
<td>1</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>3160</strong></td>
</tr>
</tbody>
</table>
Usable identity pools

<table>
<thead>
<tr>
<th>State</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Research</td>
<td>2504</td>
</tr>
<tr>
<td>Only authenticated users</td>
<td>308</td>
</tr>
<tr>
<td>Does not exist</td>
<td>245</td>
</tr>
<tr>
<td>Invalid configuration</td>
<td>103</td>
</tr>
</tbody>
</table>
Insecure configurations

How many of the 2504 identity pools are poorly configured? But most importantly, how do we define “poorly configured”?

For this it is important to remember the example use-case: the mobile application reads-writes to S3 and DynamoDB, and invokes lambda functions. The mobile application never needs to list_* because it knows where to store the data, which lambda functions to call, etc.

In +80% cases anything matching list_* is not following the last privilege principle

<table>
<thead>
<tr>
<th>API Call</th>
<th>Count</th>
<th>% over total</th>
</tr>
</thead>
<tbody>
<tr>
<td>s3.list_buckets()</td>
<td>548</td>
<td>21.88%</td>
</tr>
<tr>
<td>dynamodb.list_backups()</td>
<td>96</td>
<td>3.83%</td>
</tr>
<tr>
<td>dynamodb.list_tables()</td>
<td>101</td>
<td>4.03%</td>
</tr>
<tr>
<td>lambda.list_functions()</td>
<td>98</td>
<td>3.91%</td>
</tr>
</tbody>
</table>
Sensitive data
S3 buckets

<table>
<thead>
<tr>
<th>Data</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sensitive</td>
<td>906</td>
</tr>
<tr>
<td>Other</td>
<td>12590</td>
</tr>
<tr>
<td>Total</td>
<td>12590</td>
</tr>
</tbody>
</table>
DynamoDB

dynamodb.list_tables()

<table>
<thead>
<tr>
<th>Data</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sensitive</td>
<td>37</td>
</tr>
<tr>
<td>Other</td>
<td>1141</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>1141</strong></td>
</tr>
</tbody>
</table>

- Sensitive: 3.1%
- Other: 96.9%
Lambda function environment variables

Was able to list 1572 lambda functions using `lambda.list_functions()`

The results show at least 78 environment variables that contain:

- API keys for third-party services
- AWS access keys
- Database credentials
- Passwords
Root cause analysis

Why * 5
Insecure by default documentation

**Uploading Photos to Amazon S3 from a Browser** is a tutorial on how to configure Cognito and AWS JS SDK to list, write, read and delete images to S3.

- Allow **unauthenticated** Cognito role to **s3:* on a specific bucket**

  **Note added after contacting AWS security:** This security posture is useful in this example to keep it focused on the primary goals of the example. In many live situations, however, tighter security, such as using authenticated users and object ownership, is highly advisable.
Restrictions on **Unauthenticated** Cognito roles

Cognito allows **only 26 services** to be associated with the **unauthenticated role**.

For example, it is **impossible** to use an IAM role with **EC2:** for **unauthenticated** access.

But the allowed services include DynamoDB, S3, IoT, Lambda, SimpleDB, SES, SNS and SQS.

**Lambda was the one I used during the initial cloud security assessment to fully compromise the account.**
Restrictions on authenticated Cognito roles

Cognito imposes no restrictions on the permissions a developer can set on Cognito authenticated role.

And in most Cognito applications users can create their own users by using the registration flow.
Developer can shoot himself in the foot

The existing restrictions are not enough.

The insecure Cognito configuration falls on the client's side of the shared responsibility model.

But there is certainly a trend, a big percentage of Cognito identity pools are insecurely configured.

AWS needs to review decisions made during the design phase. Changes in the user interface and configuration of the S3 service that prevent public S3 buckets is a good example of AWS revisiting their decisions.
Developers

Least privilege principle
Least privilege principle and more...

These are the most important tips for developers using Cognito:

- **Always follow the least privilege principle** when configuring the IAM roles associated with Cognito. In other words, if the IAM policy contains * you are doing it wrong.

- **Remember object level permissions.** This was not even discussed in this talk, but keep this in mind: Not all users should be able to read all objects in the S3 bucket, not all users should be able to read all rows in a DynamoDB table.
Bonus!

Hard-coded credentials
Hard-coded credentials

Since I was going to grep the internet... I included regular expressions to identify hard-coded AWS credentials!

- Found **280 hard coded credentials**
  - 26 (9%) were root accounts
  - 38 (13.5%) had high privileges that granted access to RDS, EC2 or IAM

- Sources
  - 25 findings came from common crawl
  - 1 finding came from GitHub

Reported them to [AWS' security team](https://aws.amazon.com/security/). Customers were contacted and leaked credentials disabled.
Closing thoughts
Future work

This research could be extended as follows:

- Extract identity pools from iOS applications
- Similar services, maybe in Azure, GCP, Alibaba.
- Authenticated role analysis
- Privilege escalation analysis (danger!)

I’m most likely not going to follow-up on all this, but feel free to contact me if you want to and need guidance.
Key takeaways

These are the **three most important things** to remember:

- AWS Cognito is commonly misconfigured and easy to exploit
- It is possible to grep the internet for vulnerabilities using **cc-lambda**
- It is possible to enumerate IAM permissions in a fast, safe and in-depth manner using **enumerate-iam**

Follow me on twitter [@AndresRiancho](https://twitter.com/AndresRiancho) for more interesting cloud security research
Thanks!
For hire

Does your company or startup need these services?

- Cloud Security Assessment
- Intro to AWS Hacking training
- Application Penetration Test
- Source Code Review

Let me know, I can help you deliver secure applications.