# flaws.cloud - Level 2

minute read Published: 13 Sep, 2018 8

In a previous post, I covered level 1 of flAWS.cloud, a CTF-style cloud security game in which you have to find your way in to an AWS account by abusing common misconfigurations. This walkthrough now covers level 2, in which you discover content in another vulnerable bucket. This time, S3 ACLs are the culprit. The level demonstrates one of the most common .mistakes in S3 access control configuration

Table of Contents

In level 1, we discovered that the homepage http://flaws.cloud is hosted in an S3 bucket with Amazons static page feature. Since the creator of the bucket configured the permissions such that unauthenticated users could list it, we discovered a secret file named “secret-dd02c7c.html”. It lead us to the start page of level 2. Now, the level description says we have to do .something similar, but need our own AWS account this time

A prerequisite for this level (and some future ones as well) is thus to have your own AWS account. You will not use any paid features but a credit card is required nevertheless. AWS describes what you can do in free tier .if you want to check before creating resources. Sign-up starts here

## Level 2

### Trying to list the bucket

We could start by pulling off the same trick as last time and listing the bucket for the level 2 homepage. Since the domain is “level2-:c8b217a33fcf1f839f6f1f73a00a9ae7.flaws.cloud”, we could try this

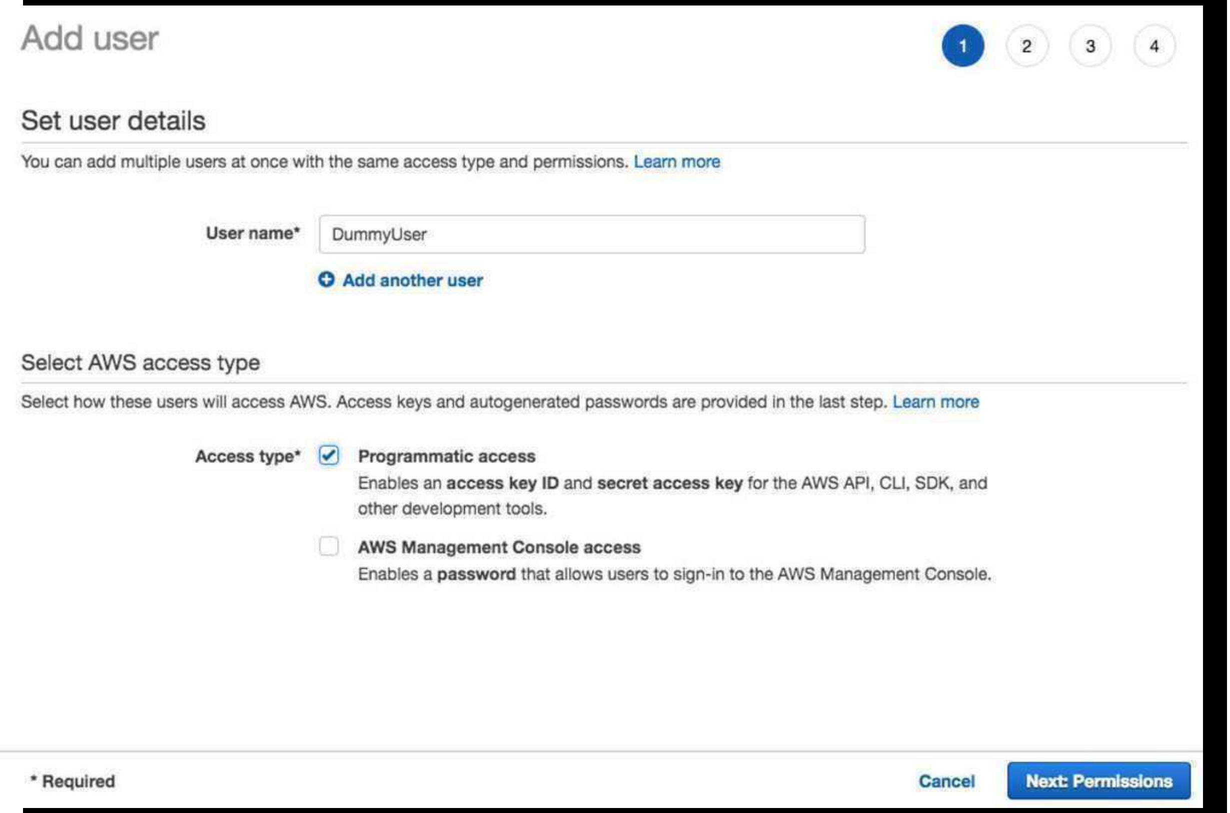
```
aws s3api list-objects-v2 --bucket level2- $
c8b217a33fcf1f839f6f1f73a00a9ae7.flaws.cloud --region us-west-2 --
no-sign-request

An error occurred (AccessDenied) when calling the ListObjectsV2
```

```
operation: Access Denied
```

This time AWS blocks our request. How could we get valid credentials for ?this bucket

### Creating a new IAM user

Since the level description mentions we need our own AWS account, let's just see what we can do with that. The first thing we do is creating a new user via the AWS console. Log into your account and go to the Identity and Access Management (IAM) service (click). Click "Add User" and follow the wizard. Make sure you check "Programmatic Access" in step 1 to allow .API access



Set user details

You can add multiple users at once with the same access type and permissions. Learn more

User name* DummyUser

Select AWS access type

Select how these users will access AWS. Access keys and autogenerated passwords are provided in the last step. Learn more

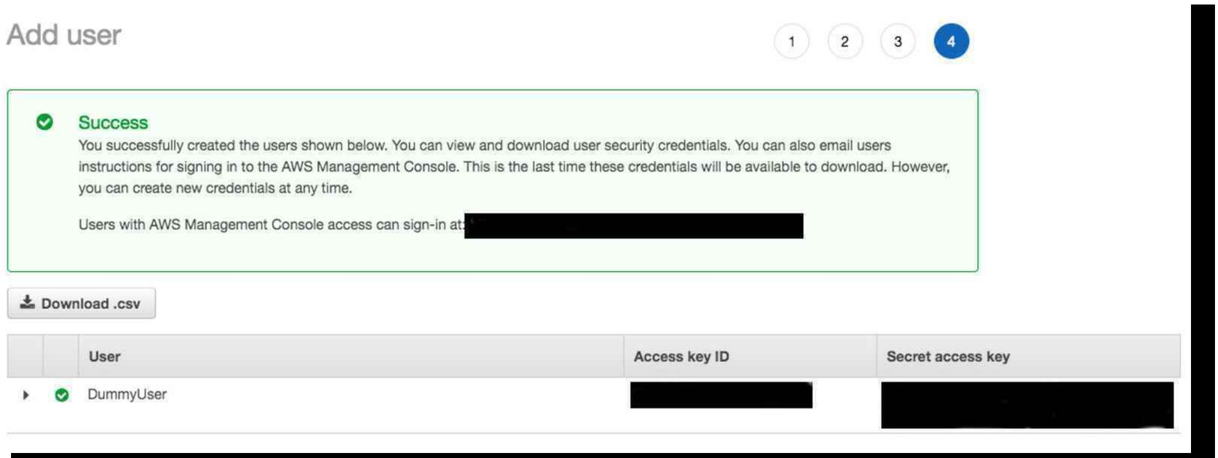
Access type* ☑ **Programmatic access**
Enables an **access key ID** and **secret access key** for the AWS API, CLI, SDK, and other development tools.

□ **AWS Management Console access**
Enables a **password** that allows users to sign-in to the AWS Management Console.

* Required

**.Check programmatic access to get API access keys**

Skip the permissions step for now and leave everything else at default. In step 4, you will get the Access Key ID and Secret Access Key, which are the two things you will need to authenticate your API requests. Copy them somewhere. After the wizard is closed, you can never see the keys again. However, if you loose them, it's no problem. You can always go back to .the user and generate new ones



Add user

**Success**

You successfully created the users shown below. You can view and download user security credentials. You can also email users instructions for signing in to the AWS Management Console. This is the last time these credentials will be available to download. However, you can create new credentials at any time.

Users with AWS Management Console access can sign-in at: [redacted]

| User | Access key ID | Secret access key |
|---|---|---|
| DummyUser | [redacted] | [redacted] |

**.AWS gives you API access keys in the final wizard step**

The easiest way to use the credentials is to set them as environment :variables. The AWS CLI will pick that up and use them

```
export AWS_ACCESS_KEY_ID=AKIAJ2BU8DYZJ8UMSJOB $
export $
AWS_SECRET_ACCESS_KEY=cNRQHFCYELGhCxe+WUKlvaz+XxYkNVqb
w3KnH45i
```

Verify that it works by executing some request and check that the error :message mentions the user you creating and it’s lack of permissions

```
aws iam get-user $
```

```
An error occurred (AccessDenied) when calling the GetUser operation:
User: arn:aws:iam::563389642670:user/DummyUser is not authorized
to perform: iam:GetUser on resource: user DummyUser
```

Now we can try again to list the bucket. Execute the same request but :remove “–no-sign-request” to authenticate yourself this time

```
aws s3api list-objects-v2 --bucket level2- $
c8b217a33fcf1f839f6f1f73a00a9ae7.flaws.cloud --region us-west-2
```

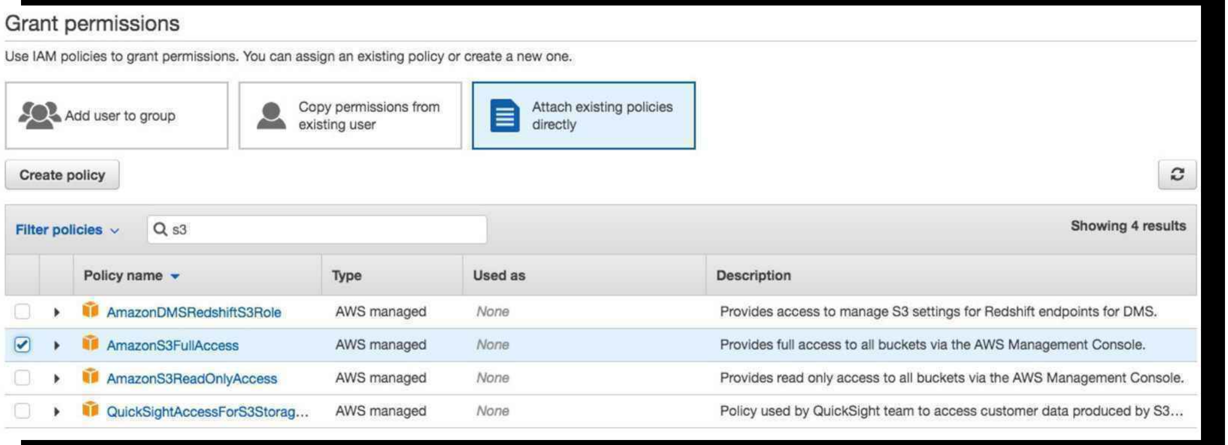
```
An error occurred (AccessDenied) when calling the ListObjectsV2
operation: Access Denied
```

...Still the same error message

## Granting access to S3

It should not be a surprise that bucket listing failed because the user does not have any permissions. Maybe it works better if we give it access to S3. .We can do that by attaching an IAM policy to use user

AWS comes with numerous managed policies for all sorts of use cases. To be on the safe side, you can just give a user full access to S3, which means it can do everything with all the resources (unless denied by another policy on the user or the bucket). Since we are in the Console already, just click on your user and then on “Add permissions”. In the following screen, click on “Attach existing policies directly” and then search for “AmazonS3FullAccess”. Check this policy and click “Review”, .”then “Add permissions



Grant permissions

Use IAM policies to grant permissions. You can assign an existing policy or create a new one.

Add user to group
Copy permissions from existing user
Attach existing policies directly
Create policy
Filter policies
s3
Showing 4 results

| | | Policy name | Type | Used as | Description |
|---|---|---|---|---|---|
| □ | ▸ | AmazonDMSRedshiftS3Role | AWS managed | None | Provides access to manage S3 settings for Redshift endpoints for DMS. |
| ☑ | ▸ | AmazonS3FullAccess | AWS managed | None | Provides full access to all buckets via the AWS Management Console. |
| □ | ▸ | AmazonS3ReadOnlyAccess | AWS managed | None | Provides read only access to all buckets via the AWS Management Console. |
| □ | ▸ | QuickSightAccessForS3Storag... | AWS managed | None | Policy used by QuickSight team to access customer data produced by S3... |

**.Granting your user full access to S3**

:List again to see if adding the policy helped

```
                              aws s3api list-objects-v2 --bucket level2- $
c8b217a33fcf1f839f6f1f73a00a9ae7.flaws.cloud --region us-west-2
                                                                         }
                                                          ] :"Contents"
                                                                    ...
                                                                     }
                                         ,"Key": "index.html"
                    ,"LastModified": "2017-02-27T02:02:14.000Z"
              ,""\ETag": "\"bbc2900889794698e208a26ce3087b6f"
                                               ,Size": 2786"
                                   "StorageClass": "STANDARD"
                                                                    ,{
                                                                    ...
```

```
                                                      }
                         ,"Key": "secret-e4443fc.html"
          ,"LastModified": "2017-02-27T02:02:15.000Z"
  ,""\ETag": "\"8207323f2b9dcfc5983421452f91ad5f"
                                       ,Size": 1051"
                         "StorageClass": "STANDARD"
                                                      {
                                                        [
                                                          {
```

Very nice! Just by creating a user and giving it access to S3 in general, we were able to access the bucket of somebody else. Among the objects, we .find “secret-e4443fc.html”, which contains the link to the next level

## The flaw

Many companies using AWS do not just have one AWS account. Rather, you often set up multiple accounts for different teams for improved security, isolation, and to keep teams independent and flexible click. Thus, sharing S3 buckets between accounts is a commonly used feature. For users of a single AWS account not thinking about such more advanced use .cases though, some of the configuration options are easily misinterpreted

Access management for S3 buckets is rather complex. Multiple different ways to define access rights coexist. One is through IAM policies attached to IAM users, the way we used above to get access. Other mechanisms are to attach IAM policies to the buckets as well as using the S3 ACL feature. All these access rights are combined and evaluated on each request, as described in the flow chart on the bottom of this page. The full .gory details are in the AWS docs

In our case here, the situation is that an IAM user of account A (ours) tries to perform a bucket operation (listing objects) on a bucket in account B. Thus, AWS will evaluate access rights as described here First, it will check if we gave our IAM user rights to read S3 buckets, which we have done with the IAM policy “AmazonS3FullAccess”. Second, it will check if our AWS account has been given access. Unless account B gave access explicitly, AWS will block us. So, how could the owner have given that ?access

## Sharing via bucket policies

One way is with a bucket policy. With bucket policies, you allow principals to perform actions, either on the bucket or on a given set of objects in the bucket. If you happen to specify “*” as the principal, everyone would get :rights to perform the action

```
}
,"Version":"2012-10-17"
]:"Statement"
}
,"Sid":"grantPublicAccess"
,"Effect":"Allow"
,"*" :"Principal"
,Action":["s3:ListBucket"]"
Resource": ["arn:aws:s3:::level2-"
c8b217a33fcf1f839f6f1f73a00a9ae7"]
{
[
{
```

Instead, you should always specify the principal like so when you allow actions: {"AWS": ["arn:aws:iam::111122223333:root"]}. This way, you would limit access to only the AWS account with ID 111122223333. Many sample AWS policies from the docs use “*” and if you forget to replace it .after copy&pasting, you may leave your bucket wide open

In our case, such a bucket policy is likely not responsible for the flaw. As far as I know, there is no way to use wildcards for principals apart from just “*” itself (docs). Thus, you could either give anonymous access or grant access to specific accounts, but not grant access to authenticated .users of any account

## Sharing via S3 ACLs

The other way to grant access is via S3 ACLs, which are described here. They allow specifying access rights on a broader level, distinguishing between read and write access for objects and/or ACLs. Under they hood, they get transformed into something similar to an additional bucket policy, according to rules specified here. For example, granting READ access via an S3 ACL is the same as granting “s3:ListBucket” and .“s3:GetObject” actions (plus their variations) with a bucket policy

S3 ACLs may grant rights only to specific AWS accounts or to any of a set :of predefined groups. These groups include

- Authenticated Users: all AWS accounts (access only via authenticated requests, but can be authenticated by any AWS account in the world)
- All Users: anyone in the world (access via unauthenticated requests, could also be achieved with a bucket policy with principal “*")

If you read the group names carelessly, you mights be tempted to believe “Authenticated Users” grants access to all IAM users of your account, where in fact it effectively grants access to all accounts. “User” means .”“AWS Account” in this case, not “IAM User

:For example, the bucket in this level could have an ACL similar to this one

```
<?"xml version="1.0" encoding="UTF-8?>
AccessControlPolicy xmlns="http://s3.amazonaws.com/doc/2006-03->
<"/01
<Owner>
ID>d70419f1cb589d826b5c2b8492082d193bca52b1e6a81082c36>
<c993f367a5d73</ID
<DisplayName>0xdabbad00</DisplayName>
<Owner/>
<AccessControlList>
...
<Grant>
Grantee xmlns:xsi="http://www.w3.org/2001/XMLSchema->
"instance
<"xsi:type="Group
URI>http://acs.amazonaws.com/groups/global/AuthenticatedUse>
<rs</URI
<Grantee/>
<Permission>READ</Permission>
<Grant/>
...
<AccessControlList/>
<AccessControlPolicy/>
```

By granting READ access to the Authenticated Users group, anyone with an AWS account can list the bucket to which the ACL is attached. To make such an ACL more safe, you could change the grantee from a group to a :dedicated account

```
  "Grantee xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance>
                                         <"xsi:type="CanonicalUser
     <ID>long_canonical_user_id_of_target_account_goes_here</ID>
   <DisplayName>account_name_of_target_account</DisplayName>
                                                                <Grantee/>
```

As a side note: each AWS account does not only have an “Account ID” but also a “Canonical User ID” (docs). While ARNs, used in the IAM user and bucket policies, rely on Account IDs to specify accounts, S3 ACLs use the .Canonical User IDs

An ACL granting access to the “Authenticated Users” group seems to be .precisely what the creator of level 2 has done

**Conclusion**

This level demonstrated a misconfiguration which is likely one of the most common things admins got wrong in recent times. In fact, AWS even removed the option altogether from the console (previously, is was named “any AWS user” in the UI (see here), which is obviously easily confused with “any of my IAM users” if you are a person using only a single AWS account. Now, you can only grant access to specific accounts, or make it “Public”, i.e., open to the world. Moreover, the UI is littered with colorful .warnings whenever you set something public

This level illustrated an important issue but many more can arise if S3 ACLs are not set properly. For a deep dive into configuration options and .their consequences, check out this post