Cloud Catastrophes

and how to avoid them

Who?

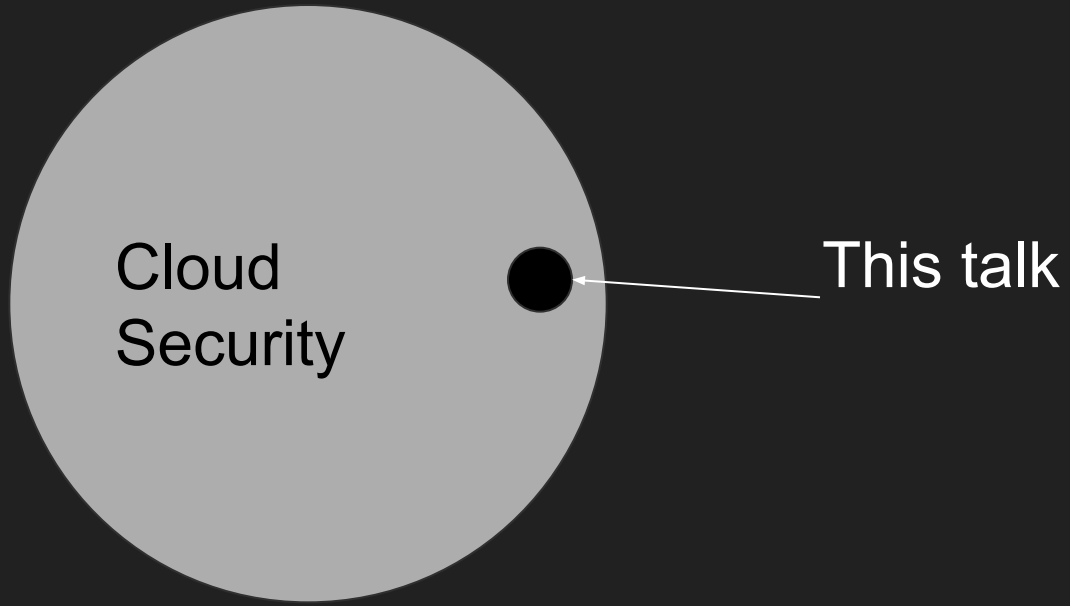
Mike Haworth

Nope

Works for: Insomnia Security

Bio: Extensive biographical information Extensive biographical information Extensive biographical information Extensive biographical information Extensive biographical information Extensive biographical information Extensive biographical information Extensive biographical information Extensive biographical information Extensive biographical information Extensive biographical information Extensive biographical information Extensive biographical information Extensive biographical information Extensive biographical information Extensive biographical information Extensive

It is a big topic

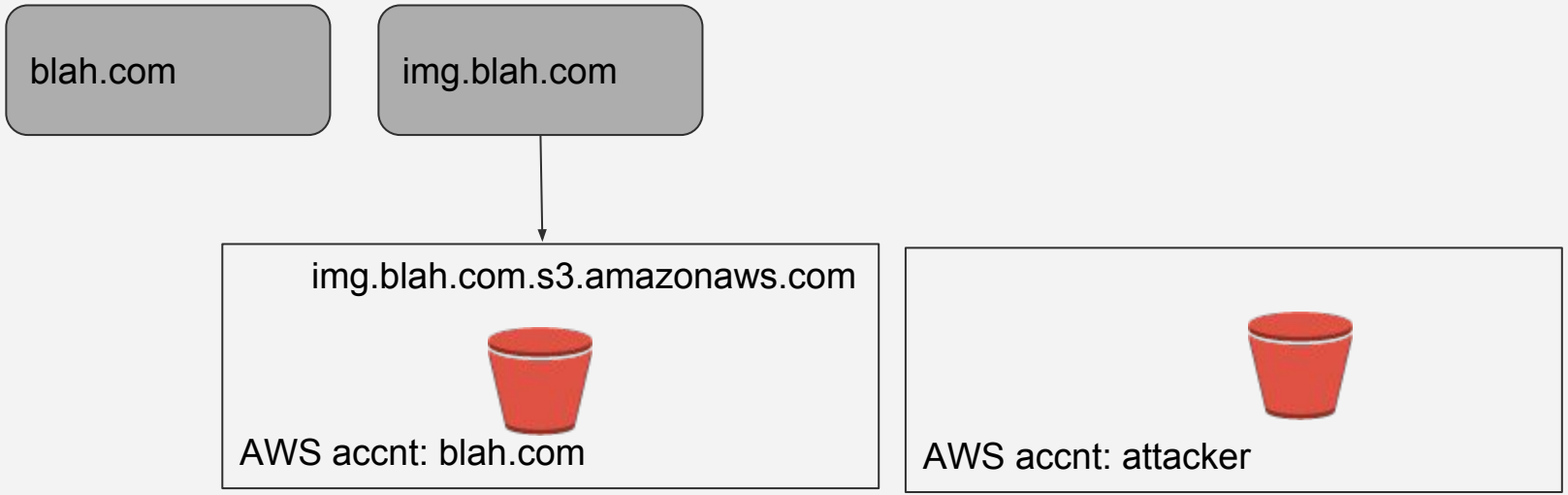


Theme

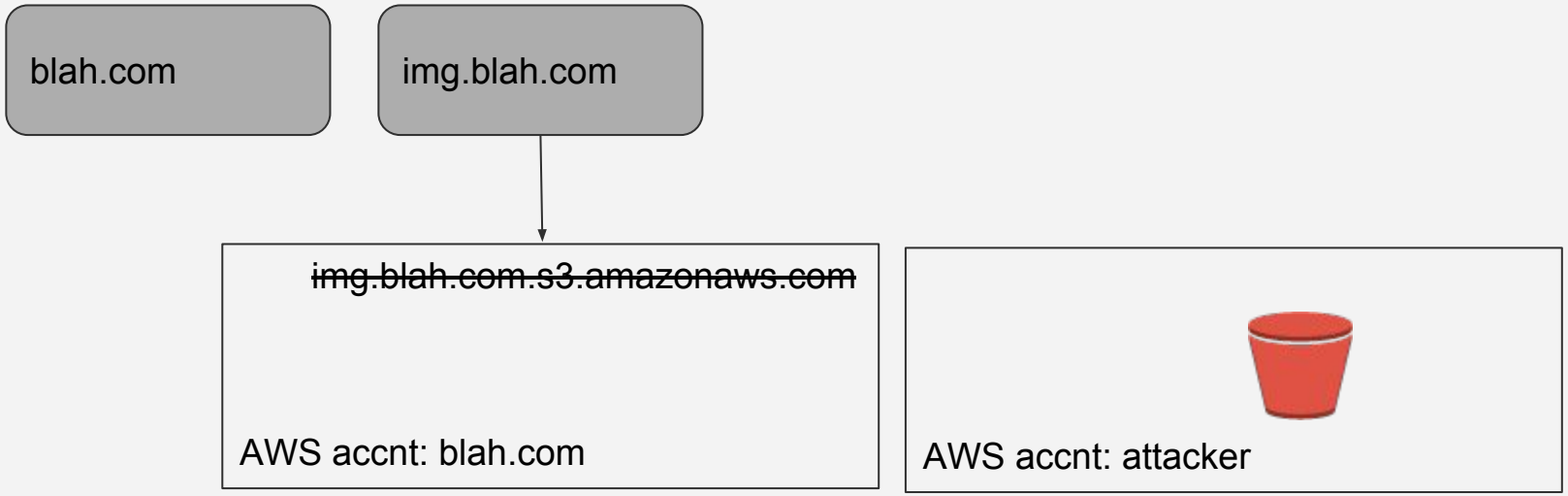
Many issues we see in Cloud environments are due to missing some of the mindshift required to do things the ‘Cloudy’ way

1. Global pools of identifiers -> Hijacking of orphaned resources
2. Cloud APIs are public -> Cred disclosure can be catastrophic
3. Gaps in knowledge of cloud auth models -> Gaps in Auth

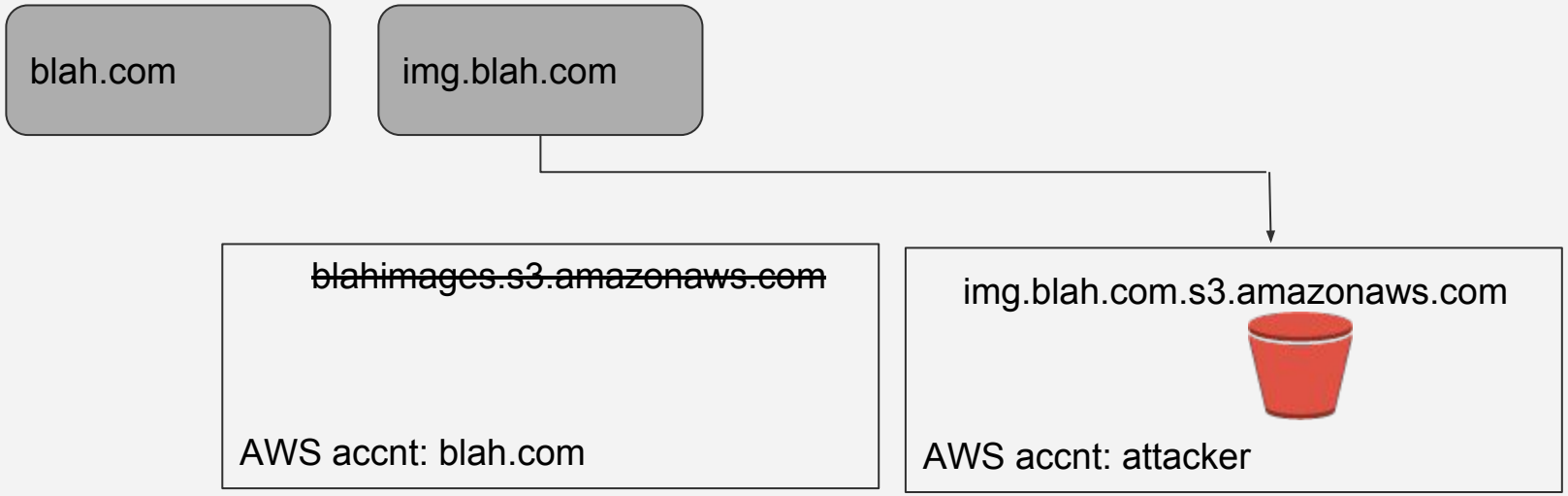
Resource Hijacking



Resource Hijacking

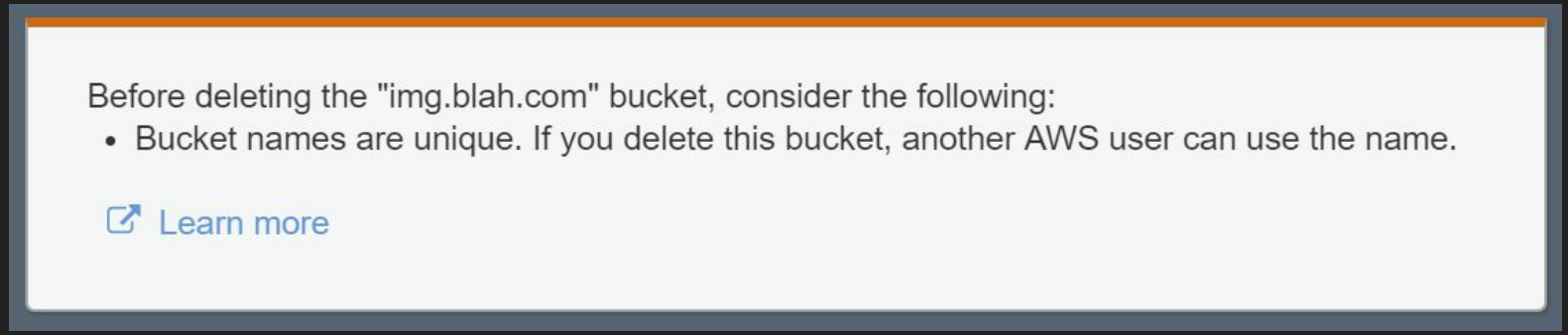


Resource Hijacking



Resource Hijacking

So removing a resource that you need is a basic error



But sometimes basic errors .. can lead to code exec

Unclaimed bucket -> code exec

Install script pulls binary from unclaimed S3 bucket

```
+ curl -fSL "https://s3.amazonaws.com/rocketchatbuild/rocket.chat-develop.tgz" -o rocket.chat.tgz
tar zxf rocket.chat.tgz  &&  rm rocket.chat.tgz
cd $ROOTPATH/bundle/programs/server
npm install
pm2 startOrRestart $ROOTPATH/current/$PM2FILE
```

So I decided to see if I could access the contents of that S3 bucket. To my surprise, I got the following error message:

```
$ aws s3 ls s3://rocketchatbuild

An error occurred (NoSuchBucket) when calling the ListObjects operation: The specified bucket does not exist
```

https://hackerone.com/reports/399166

Resource Hijacking

The namespace for many cloud resources is global

- If the identifier is user controlled AND
- Another party is able to register that name
- Then the attacker could serve their content to clients that visit that domain

Resource Hijacking

Service	Hijackable?
AWS S3	Yes
AWS Cloudfront	Edge case
Azure Webhosting	Yes
Heroku	Edge case
Google Cloud Storage	No

Source: https://github.com/EdOverflow/can-i-take-over-xyz

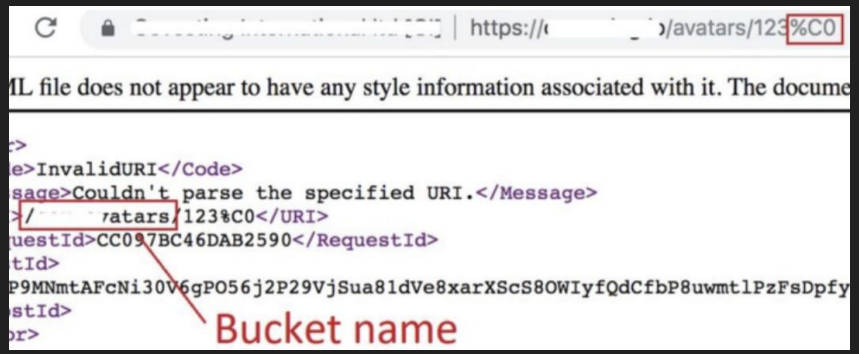
Mitigating Hijacking

1. Keep DNS and Cloud resources in sync to prevent “dangling” resources
2. Automate a mechanism of tracking DNS and cloud assets

Enumeration of S3 Buckets

Don’t assume that because a bucket is behind a CDN that the name can’t be discovered

1. Public buckets will serve a torrent file if you append “?torrent” (torrent file contains the bucket name)
2. Errors may include the name



If the bucket name is discovered attacker can check for write access & dir listing

https://medium.com/@localh0t/unveiling-amazon-s3-bucket-names-e1420ceaf4fa

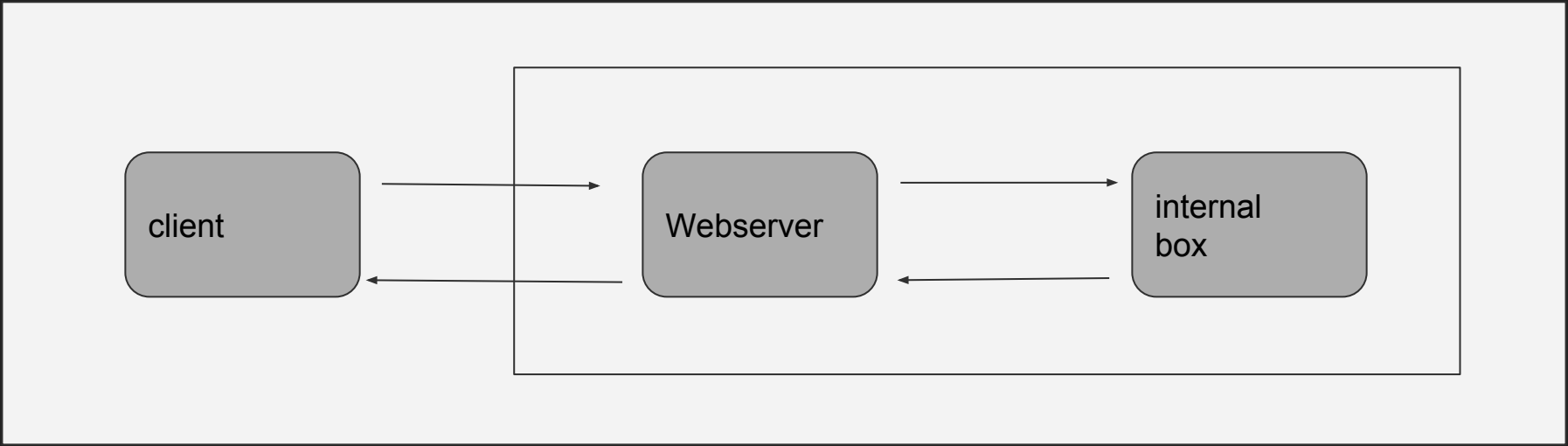
2. Credential Disclosure

One common way AWS credentials are disclosed is Server Side Request Forgery (SSRF) against the AWS metadata service

Server Side Request Forgery (SSRF)

Attacker causes server to make an HTTP request

Its most useful when the result is displayed to the attacker



Application functionality where SSRF is common

Common sources of SSRF

- XML parsing (XXE)
- PDF / page conversion functionality
- Application proxying (e.g. API gateway)

Sometimes useful

- Image uploads (may accept inline file OR a URL)
- Web hooks

What is the Metadata Service

Cloud services need a mechanism to populate instances with configuration

E.g SSH key

They use an internal service where instances can request this data

E.g. AWS uses **http://169.254.169.254/latest/meta-data**

Other services have equivalent URLs

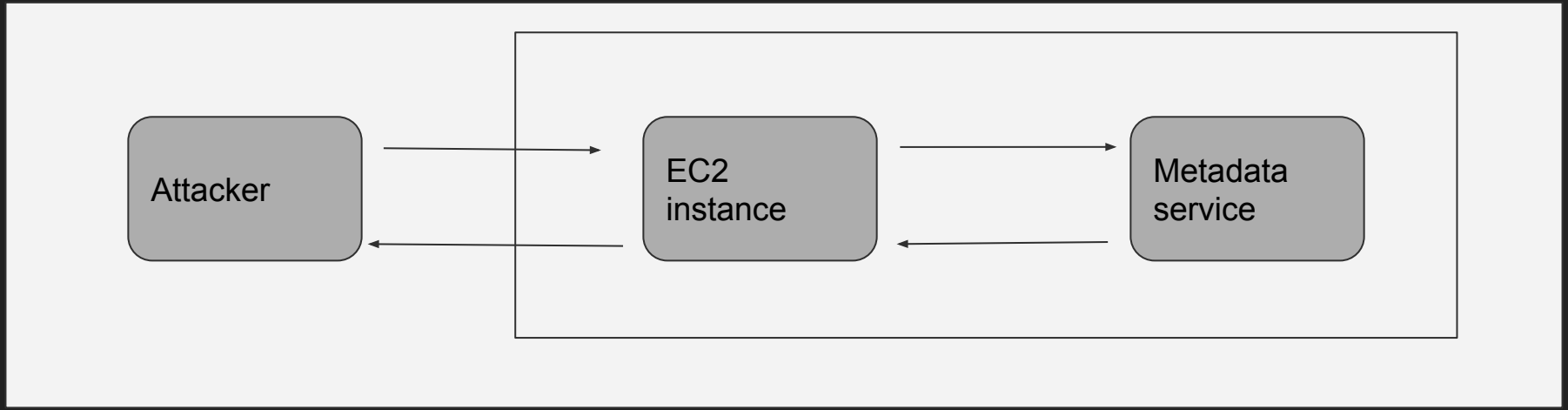
Listing: https://gist.github.com/BuffaloWill/fa96693af67e3a3dd3fb

Trivial Example

1. Attacker says: I want a PDF of this URL:

   http://169.254.169.254/latest/meta-data/iam/security-credentials/instanceRole

2. EC2 instance fetches page and returns the result as a PDF



Trivial Example cont.

PDF Contains:


```
{
        "Code"              :              "Success",
    "LastUpdated"       :       "2019-04-26T09:00:42Z",
        "Type"              :              "AWS-HMAC",
     "AccessKeyId"        :        "ASIAIB[redacted]ZZ",
   "SecretAccessKey"      :     "22oRmA[redacted]F2IJJ",
     "Token"       :      "AAoDZZdzEOv//////[redacted]",
     "Expiration"        :       "2019-04-26T16:00:16Z"
}
```


Attacker can import these credentials into the AWS CLI and perform actions with the rights of the instanceRole

Bypassing Anti-SSRF measures

An API gateway

1. Configuration step
   - Setup backend API URL
   - Setup frontend URL, so request is passed to backend URL

2. API Gateway is in service

   - Request comes in, DNS lookup for domain in backend URL
   - HTTP request to backend URL

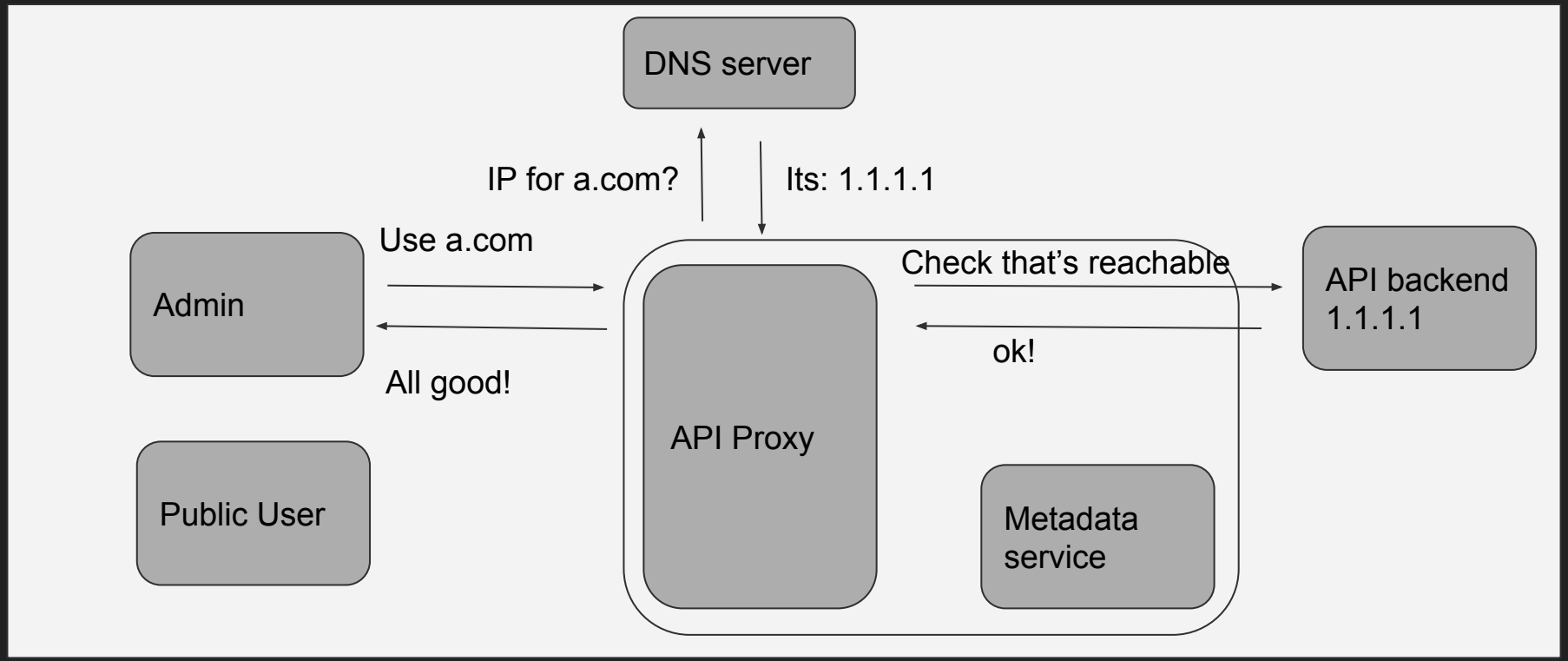
Bypassing Anti-SSRF measures

In step one there’s a sanity check of the backend URL

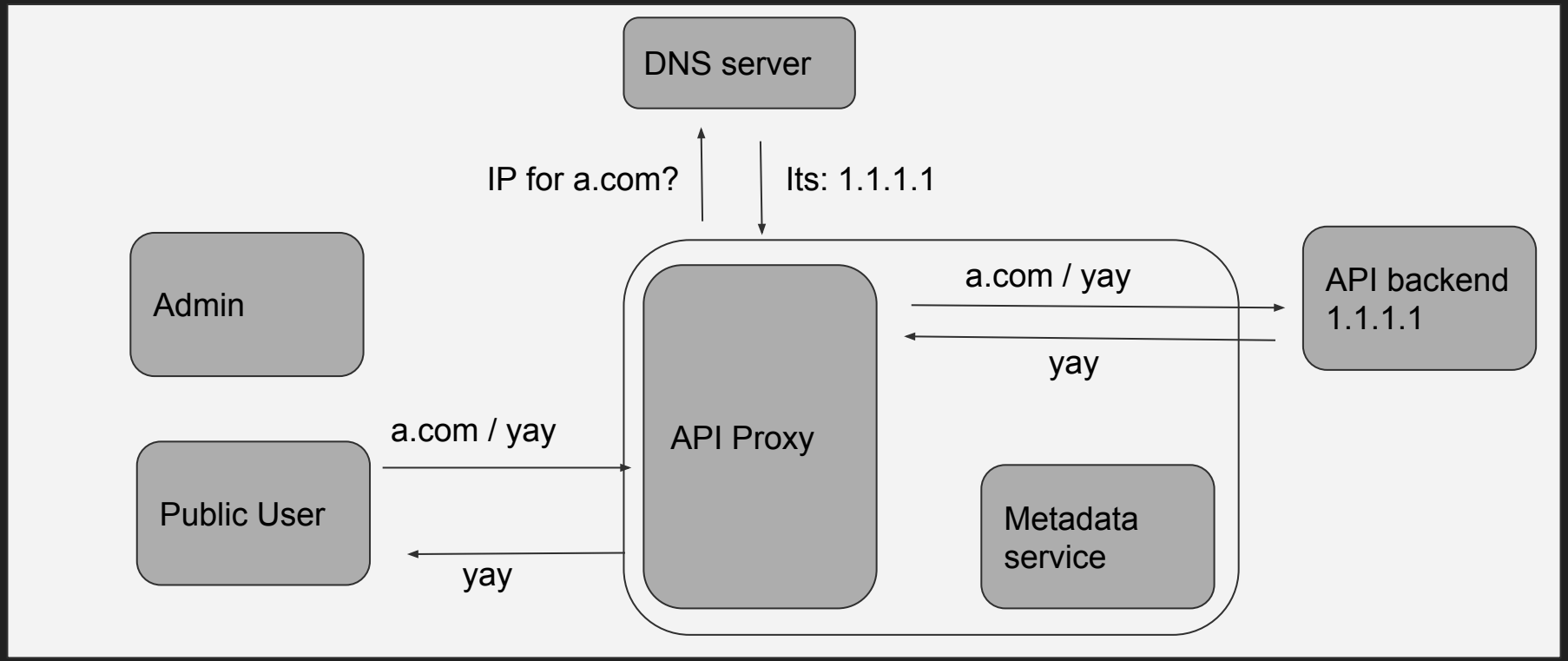
169.254.169.254 is not permitted

Any host the resolves to 169.254.169.254 is also not permitted :(

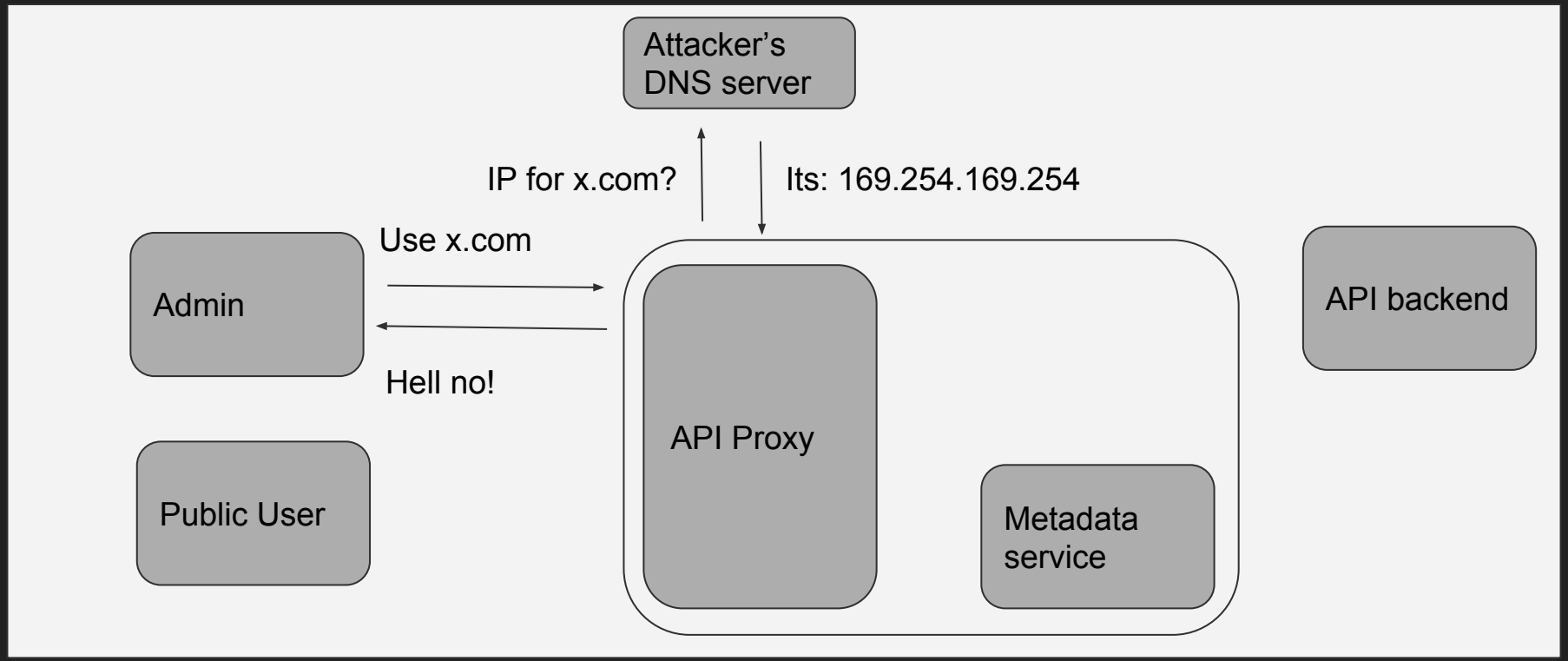
Normal use 1


DNS server
IP for a.com?
Its: 1.1.1.1
Use a.com
Admin
All good!
Check that's reachable
ok!
API backend
1.1.1.1
API Proxy
Public User
Metadata
service

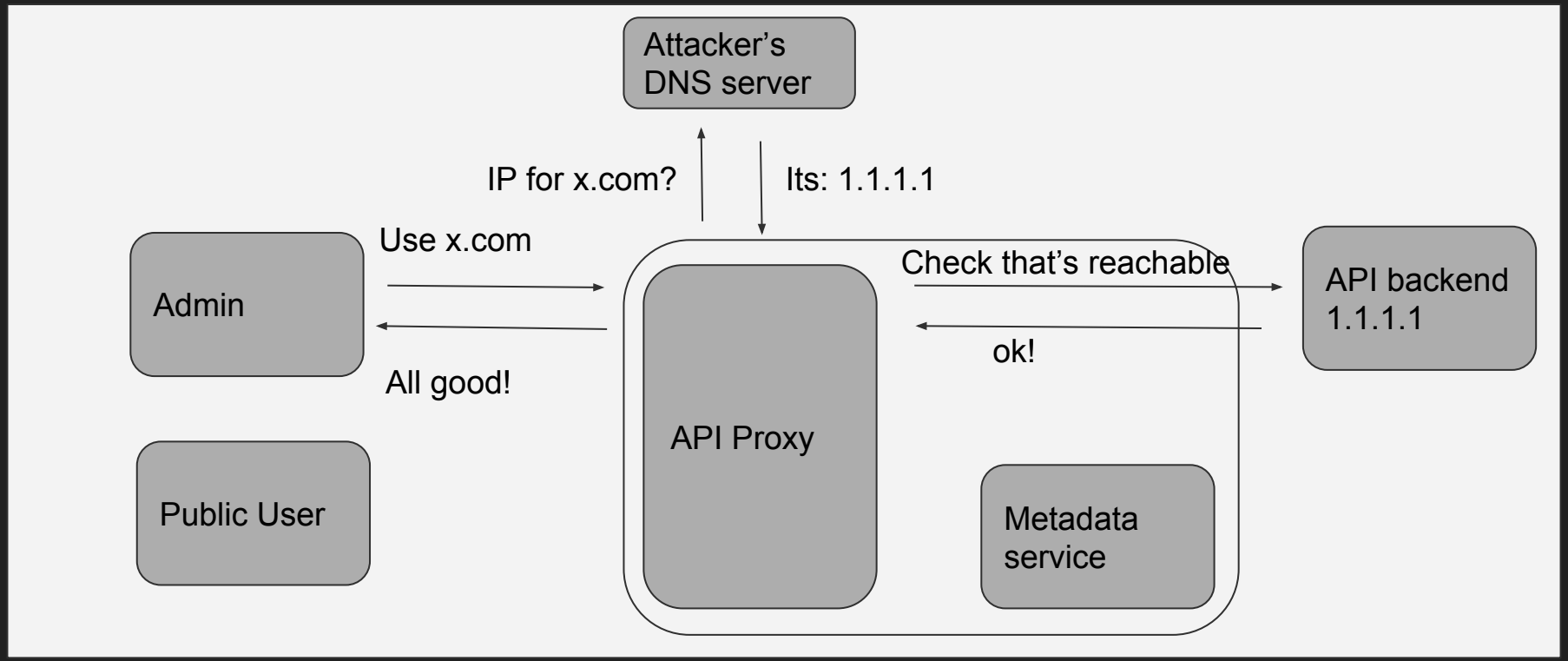
Normal use 2


DNS server
IP for a.com?
Its: 1.1.1.1
Admin
a.com / yay
API backend
1.1.1.1
yay
a.com / yay
API Proxy
Public User
Metadata
service
yay

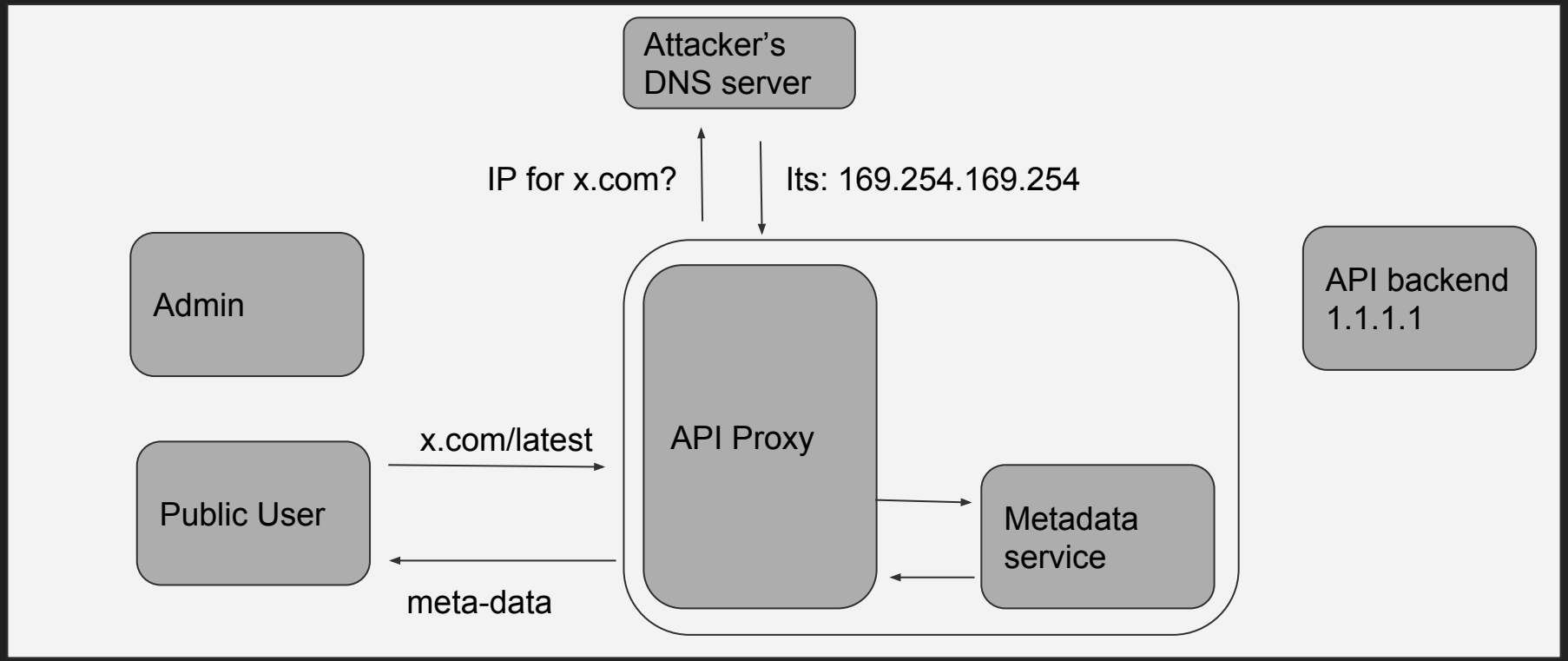
Attack attempt 1



Attack Attempt 2.



Second DNS request returns metadata IP



Mitigating Impact of SSRF in Cloud

Options:

- Avoid putting API keys in /user-data scripts etc

[1] https://www.slideshare.net/AmazonWebServices/detecting-credential-compromise-in-aws-sec389-aws-reinvent-2018

Mitigating Impact of SSRF in Cloud

Options:

- Avoid putting API keys in /user-data scripts etc
- Implement an IAM policy where use of creds is IP restricted [1]

[1] https://www.slideshare.net/AmazonWebServices/detecting-credential-compromise-in-aws-sec389-aws-reinvent-2018

Mitigating Impact of SSRF in Cloud

Options:

- Avoid putting API keys in /user-data scripts etc
- Implement an IAM policy where use of creds is IP restricted [1]
- Implement a proxy that whitelists by user agent [1]

[1] https://www.slideshare.net/AmazonWebServices/detecting-credential-compromise-in-aws-sec389-aws-reinvent-2018

Mitigating Impact of SSRF in Cloud

Options:

- Avoid putting API keys in /user-data scripts etc
- Implement an IAM policy where use of creds is IP restricted [1]
- Implement a proxy that whitelists by user agent [1]
- Trigger alerts when creds used from unknown IPs [1]

[1] https://www.slideshare.net/AmazonWebServices/detecting-credential-compromise-in-aws-sec389-aws-reinvent-2018

3. Authentication and Trust Relationships

Authentication and Trust Relationships

Scenario:

My rad dev shop starts off with just me and Joe

We both have admin access in AWS coz, we have like 1 customer

..

4 years later

Things are a bit more hectic, 10 developers, lots of customers

So we enabled MFA on all developer accounts, good to go!

API Keys

Those developer API keys have full access to the environment

Developers need the access to troubleshoot, so can’t just segment per customer

What if a developer laptop is compromised or creds are accidentally disclosed?

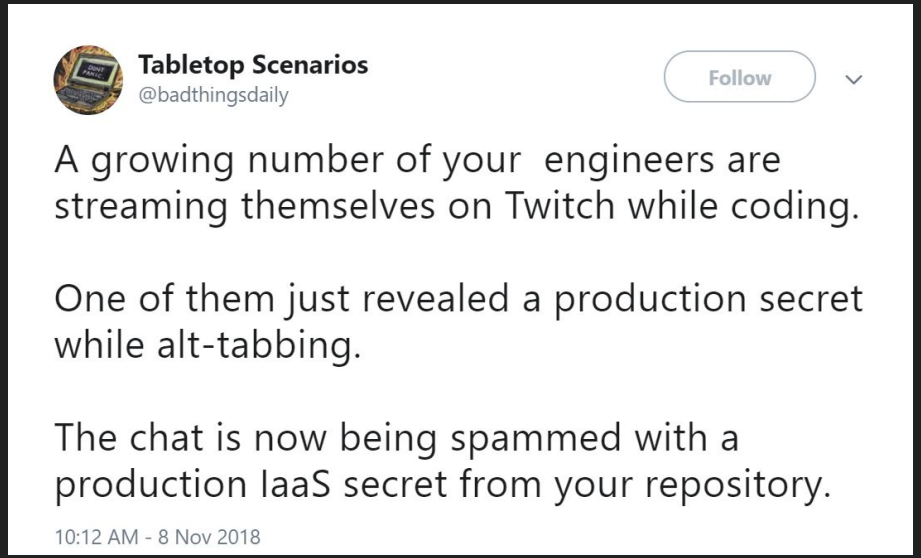
API Keys

There are a lot of ways an API key could be exposed

API Keys

There are a lot of ways an API key could be exposed

You just don't want it to be a "game over" event.

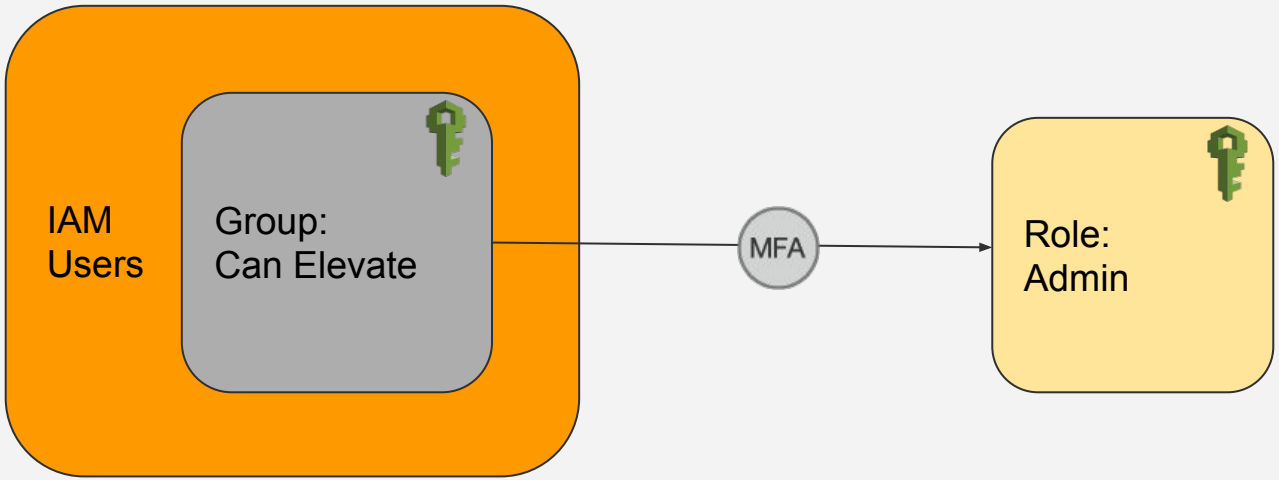


Using AssumeRole to enforce MFA

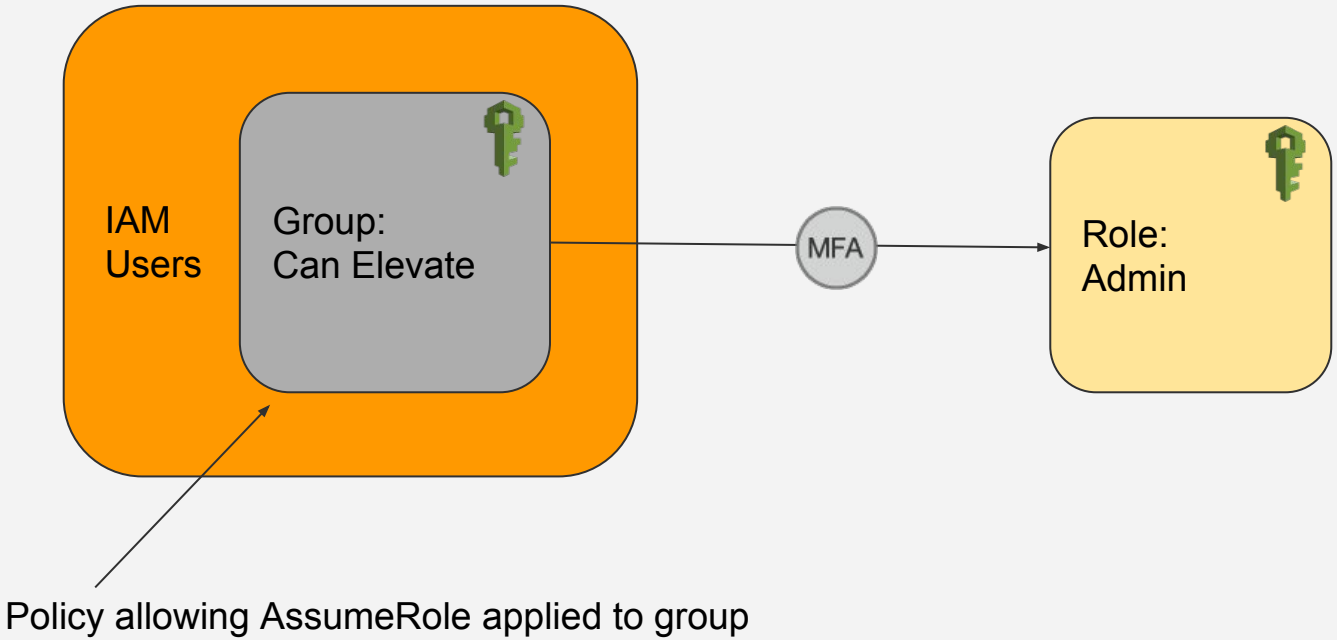
Admin starts in a “bastion” account with no rights

Then use AssumeRole to gain admin rights

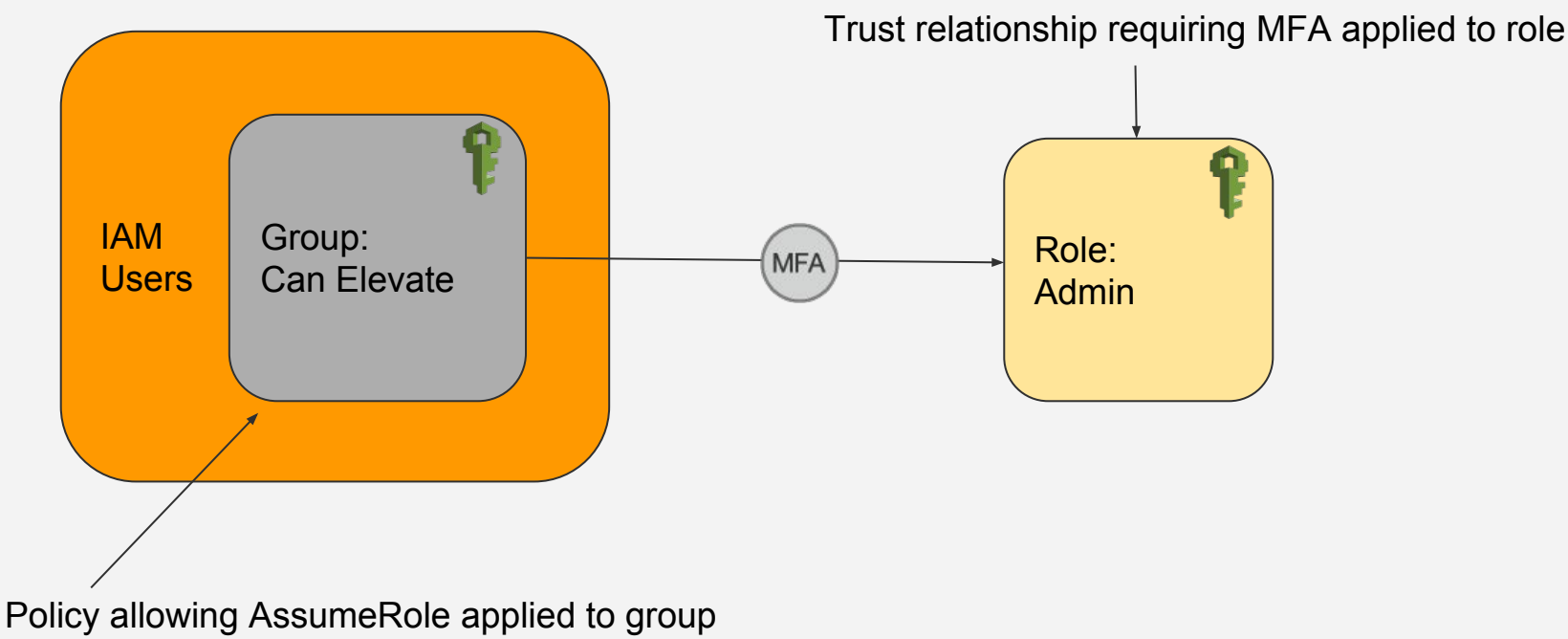
Using AssumeRole to enforce MFA



Using AssumeRole to enforce MFA



Using AssumeRole to enforce MFA



How to 2FA API access overview

1. Enable MFA in AWS console for admins

How to 2FA API access overview

1. Enable MFA in AWS console for admins
2. Create a group ‘can-elevate’, put your admins in this group

How to 2FA API access overview

1. Enable MFA in AWS console for admins
2. Create a group ‘can-elevate’, put your admins in this group
3. Grant ‘can-elevate’ the right to AssumeRole

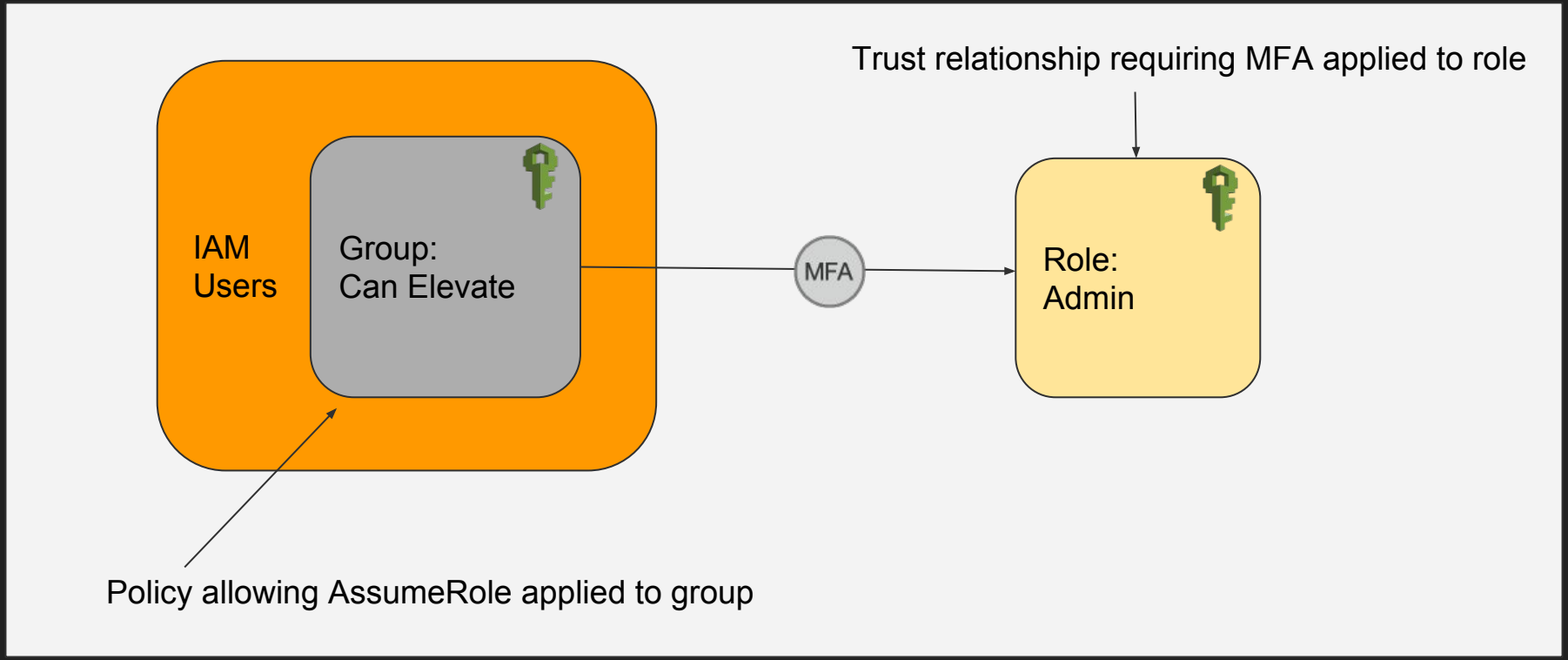
How to 2FA API access overview

1. Enable MFA in AWS console for admins
2. Create a group ‘can-elevate’, put your admins in this group
3. Grant ‘can-elevate’ the right to AssumeRole
4. Create a role ‘admin’ that trusts members of ‘can-elevate’ and requires MFA

How to 2FA API access overview

1. Enable MFA in AWS console for admins
2. Create a group ‘can-elevate’, put your admins in this group
3. Grant ‘can-elevate’ the right to AssumeRole
4. Create a role ‘admin’ that trusts members of ‘can-elevate’ and requires MFA
5. Setup AWS CLI so it prompts for MFA code

Ref: https://blog.jayway.com/2017/11/22/aws-cli-mfa/


Trust relationship requiring MFA applied to role
IAM
Users
Group:
Can Elevate
MFA
Role:
Admin
Policy allowing AssumeRole applied to group

Configure CLI to prompt for MFA

In .aws/credentials theres a user mike-admin

In .aws/config we add

```
[profile admin]
role_arn = arn:aws:iam::409 [redacted] :role/admin-role
source_profile = mike-admin
mfa_serial = arn:aws:iam::409 [redacted] :mfa/mike-admin
~
```

End result is we get a prompt for MFA code

```
mike@ubuntu:~$ aws s3 ls --profile admin
Enter MFA code for arn:aws:iam::409[redacted]:mfa/mike-admin:
2019-02-18 23:40:50 img.blah.com
2019-02-06 20:39:26 mhtest-9000
mike@ubuntu:~$
```

IAM Policy is Very Flexible

Because IAM is very flexible there are many ways to segment trust

But there are also many ways to make mistakes

Choose the simplest model that fits with how you work

IAM = Identity and Access Management

Reviewing IAM Policies

Sometimes IAM policy is obviously too permissive

Example from a ScoutSuite[1] IAM audit finding

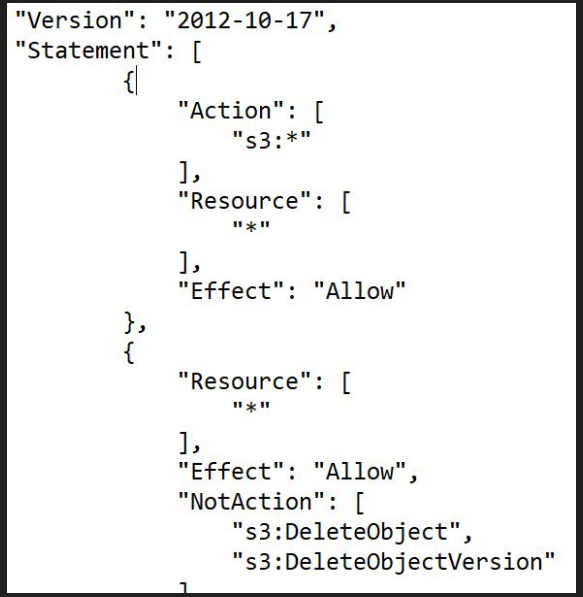
[1] https://github.com/nccgroup/ScoutSuite

IAM is complicated cont.

But it is not *always* that obvious

This policy was intended to grant access to all S3

functionality, except 2 delete perms



```
"Version": "2012-10-17",
"Statement": [
        {
            "Action": [
                "s3:*"
            ],
            "Resource": [
                "*"
            ],
            "Effect": "Allow"
        },
        {
            "Resource": [
                "*"
            ],
            "Effect": "Allow",
            "NotAction": [
                "s3:DeleteObject",
                "s3:DeleteObjectVersion"
            ]
```

Source: https://medium.com/edge-security/abusing-aws-cross-account-relationships-3b36a111b494

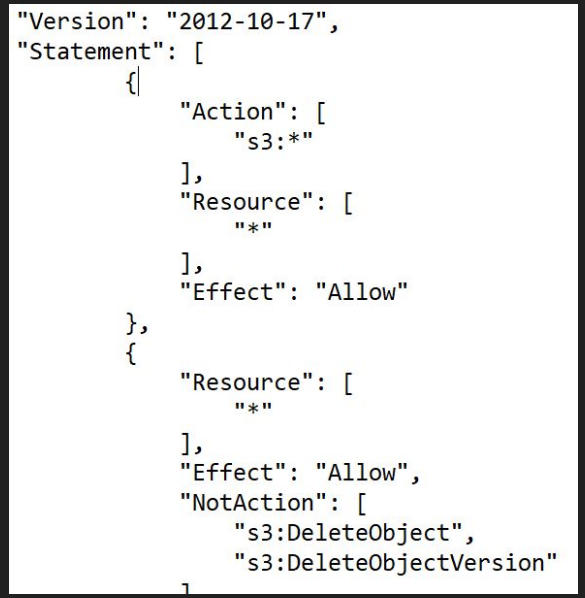
IAM is complicated cont.

But it is not *always* that obvious

This policy is intended to grant access to all S3

functionality, except 2 delete perms

Actually it grants admin rights



```
"Version": "2012-10-17",
"Statement": [
        {
            "Action": [
                "s3:*"
            ],
            "Resource": [
                "*"
            ],
            "Effect": "Allow"
        },
        {
            "Resource": [
                "*"
            ],
            "Effect": "Allow",
            "NotAction": [
                "s3:DeleteObject",
                "s3:DeleteObjectVersion"
            ]
```

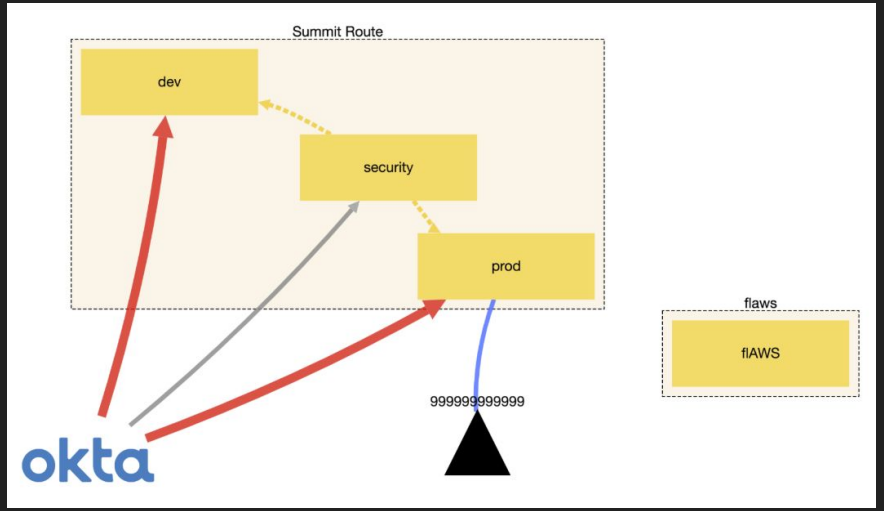
Source: https://medium.com/edge-security/abusing-aws-cross-account-relationships-3b36a111b494

Auditing IAM policy

Doing this manually in a complex environment can be daunting

Cloudmapper's web of trust feature can help visualise IAM trust relationships

https://github.com/duo-labs/cloudmapper



Summary

1. Global pools of identifiers -> Hijacking of orphaned resources

   Mitigation: monitor DNS for orphaned resources

2. Cloud APIs are public -> Cred disclosure can be catastrophic

   Mitigation: Proxy access to metadata services

3. Gaps in knowledge of cloud auth models -> Gaps in Auth

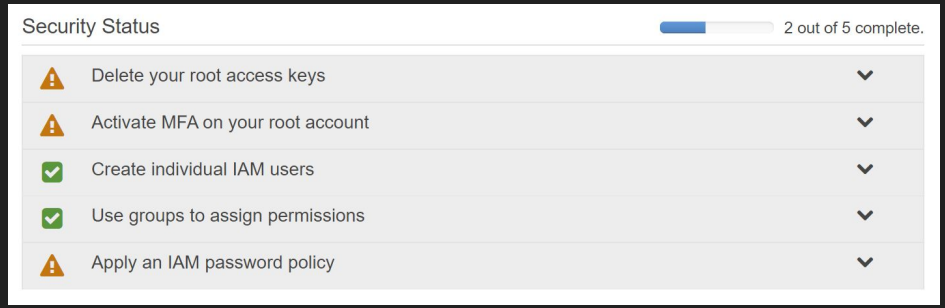
   Mitigation: MFA admin CLI users and leverage tools to analyse complex policy

Stepping back

As mentioned there is a lot to cover and this talk was just a small part

Here's a quick security todo list (Basic and boring but that's where you start)

1. Leverage the built-in tools to review IAM security
   - Use 'credential report' to find unused accounts and remove them
   - Make that security status page happy!
   - Use "access advisor" tab to sanity check access



Stepping back

2. Security groups and VPC ACLs

- Are there any ‘allow all’ rules e.g. for SSH or RDP?

3. S3 Buckets

- Do all the buckets marked ‘public’ need to be?

  New policies lets you prevent public buckets at account or per bucket level

  - Can you prevent public access account wide?
  - Can you prevent public access on this bucket?

https://aws.amazon.com/blogs/aws/amazon-s3-block-public-access-another-layer-of-protection-for-your-accounts-and-buckets/

Thanks

OWASP NZ

Everyone at Insomnia Security