CLOUD DECEPTION IN AWS

WHAT IS DECEPTION?

Cyber deception is an active defense tactic that relies on misleading threat actors into engaging with specially crafted look-alike or fake assets (known as decoys)

DECEPTION CORE CONCEPTS (MITRE)

1. Diversion
2. Resource Depletion
3. Uncertainty
4. Intelligence
5. Proactivity

Divert adversary focus away from genuine assets and towards fake targets.

DECEPTION CORE CONCEPTS (MITRE)

1. Diversion
2. Resource Depletion
3. Uncertainty
4. Intelligence
5. Proactivity

Consume adversary resources, efforts, and infrastructure in the examination of false information.

DECEPTION CORE CONCEPTS (MITRE)

1. Diversion
2. Resource Depletion
3. Uncertainty
4. Intelligence
5. Proactivity

Instil uncertainty in the adversary regarding the authenticity of identified vulnerabilities or stolen information.

DECEPTION CORE CONCEPTS (MITRE)

1. Diversion
2. Resource Depletion
3. Uncertainty
4. Intelligence
5. Proactivity

Observe and analyse adversary actions to shape future defence strategies.

DECEPTION CORE CONCEPTS (MITRE)

1. Diversion
2. Resource Depletion
3. Uncertainty
4. Intelligence
5. Proactivity

Use deception to disrupt previously unknown attacks (zero days)

WHY DECEPTION?

Three reasons why a mature org gets breached...

WHY DECEPTION?

Three reasons why a mature org gets breached...

1. Low Visibility

WHY DECEPTION?

Three reasons why a mature org gets breached...

1. Low Visibility
2. Dynamic threat landscape

WHY DECEPTION?

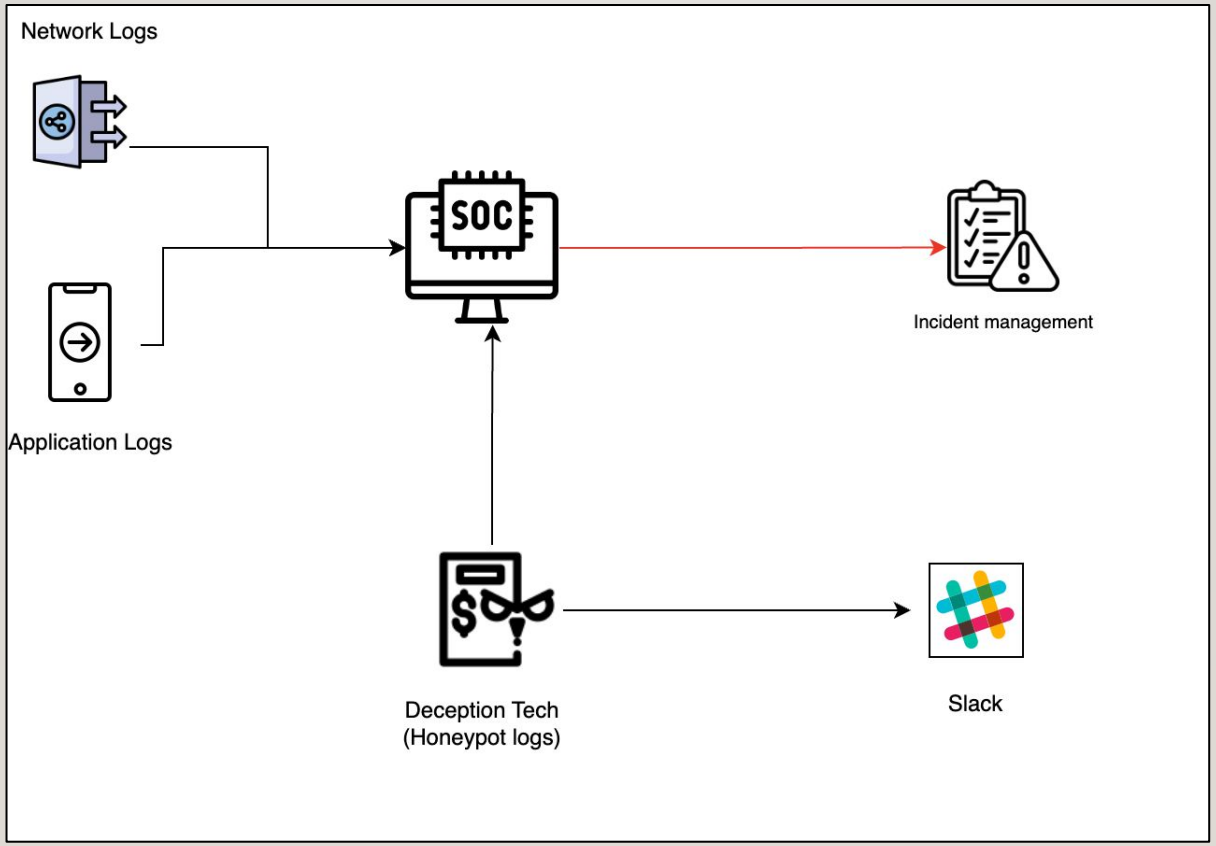
Three major reasons why a mature org gets breached...

1. Low Visibility
2. Dynamic threat landscape
3. Alert fatigue – overwhelming FPs

BENEFITS OF DECEPTION

- Attack Agnostic – Effective against zero-day and unidentified threats.
- Highly fidelity alerts – Super accurate with zero false positives
- Low Friction – Doesn't expand attack surface and neither introduce dependencies.
- Operational simplicity – Low deployment cost and straightforward to operate and scale

WHERE DOES DECEPTION FIT?



WHO SHOULD IMPLEMENT DECEPTION?

Ideally **EVERY** org but especially –

- Critical Infrastructure providers
- Financial Institutes
- Govt Agencies
- High Profile Targets

WHEN SHOULD DECEPTION BE UTILISED

AWS 101

AWS – IDENTITY AND ACCESS MANAGEMENT (IAM)

Users

- An IAM user is an identity with an associated credential and permissions attached to it.
- Each IAM user is associated with only one AWS account.

Groups

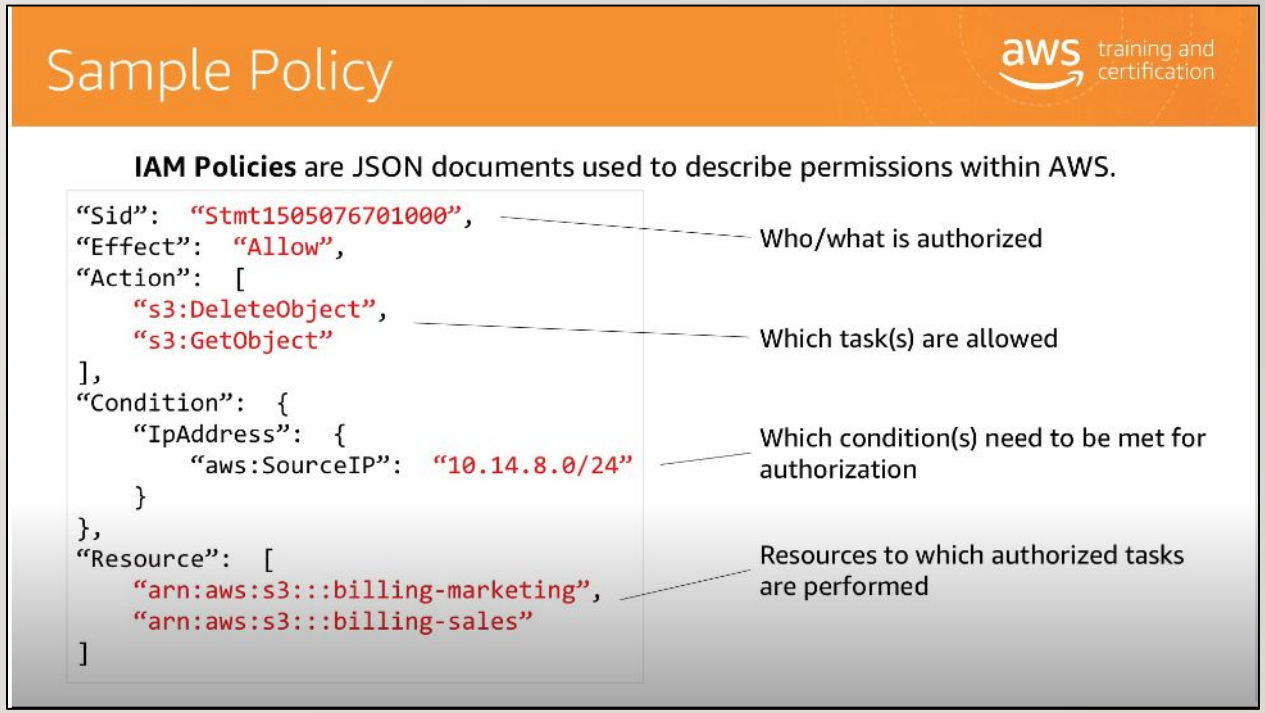
- A collection of IAM users is an IAM group
- IAM groups can be used to specify permissions for multiple users so that any permissions applied to the group are applied to the individual users in that group as well.

Roles

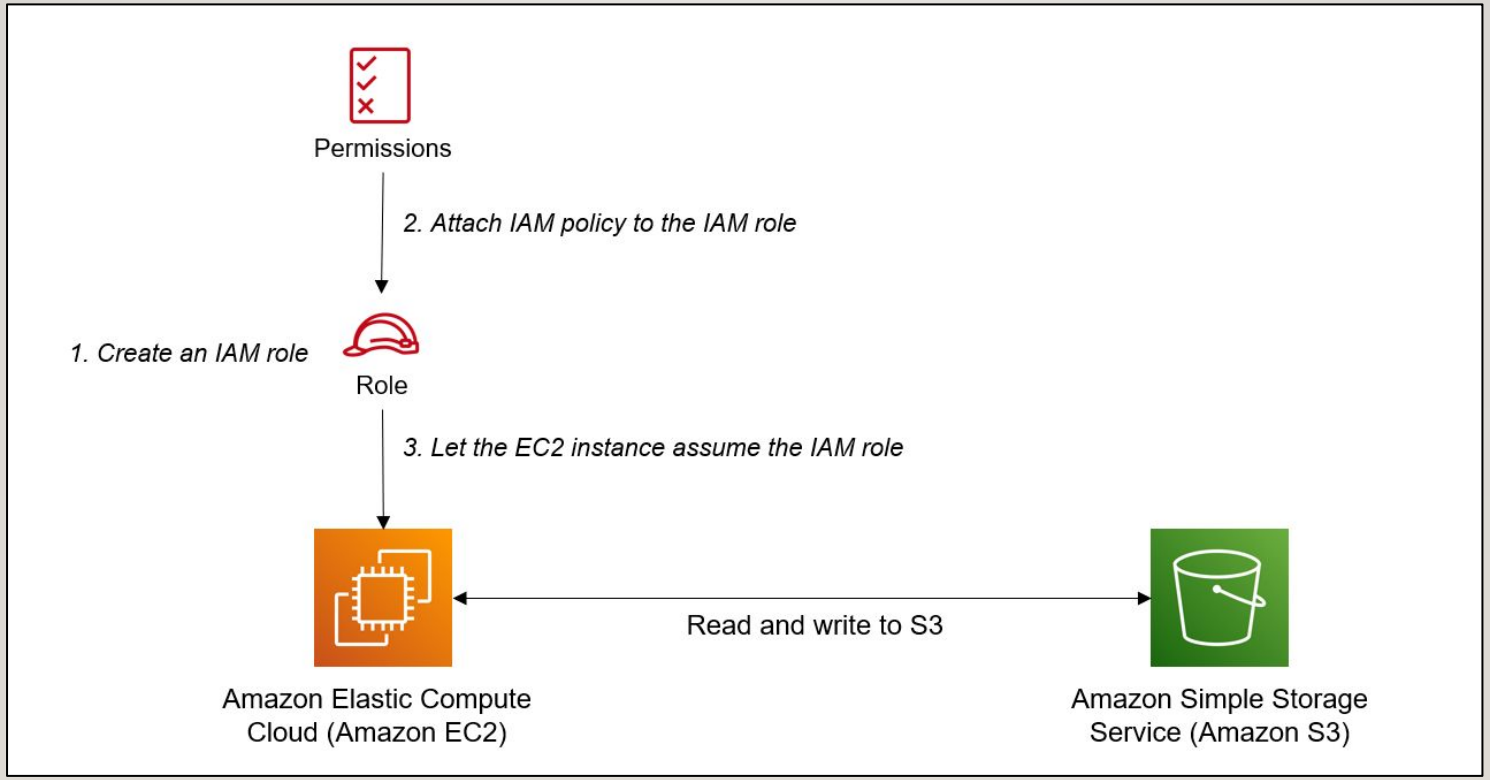
- IAM roles are used to grant **temporary access** to multiple identities including services, IAM users, or applications.
- These identities assume the role temporarily, and any permission policies attached to the role are by proxy applied to the identity assuming that role.

AWS – IAM POLICIES

- IAM policy sets permission and controls access to AWS resources by attaching them to the IAM Identities (users, groups or roles)
- IAM policies defines permissions of AWS identities and AWS resources when a user or any resource makes a request to AWS will validate these policies and confirms whether the request to be allowed or to be denied.
- Policies are stored in AWS as JSON documents.



AWS – IAM EXAMPLE



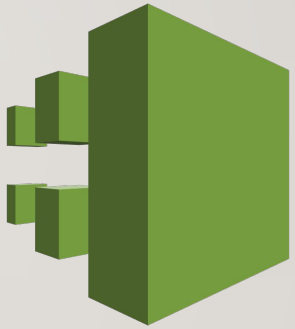
AWS – SERVICES

AWS – CLI

- The AWS Command Line Interface (AWS CLI) is a unified tool to manage your AWS services from the command line and automate the same via scripts
- Every action performed on the web interface of AWS has a corresponding API call. Under the hood AWS performs the api call on behalf of our browser
- These api calls can also be invoked via the cli.
- For example – simply opening up the IAM users page on the browser results in the `list-users` api call
- Corresponding aws cli command: `aws list-users –max-items 50`

AWS – CLOUD NATIVE LOGGING SERVICES

- **CloudTrail**
  - *CloudTrail is a service that records API calls and actions taken within your AWS account. It provides a history of changes made to resources, which can be used for security analysis, compliance auditing etc*
  - *CloudTrail captures API activity by monitoring and logging events triggered by AWS services and resources. It stores these logs in an Amazon S3 bucket, which can be further analysed using tools like AWS CloudWatch Logs or other logging and analytics services.*



CloudTrail

AWS – CLOUD NATIVE LOGGING SERVICES

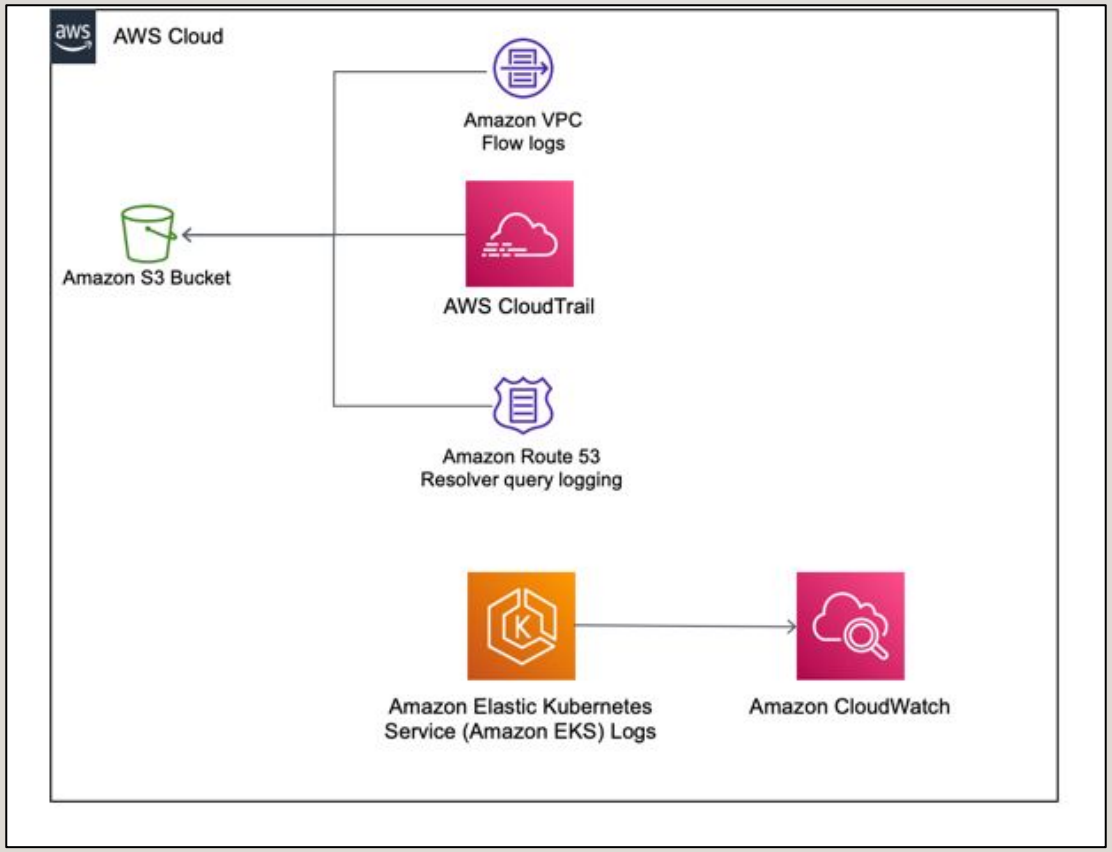
- **VPC Flow log**
  - *VPC Flow Logs enables to monitor and record traffic (layer 3 & layer 4) that enters and exits the Amazon Virtual Private Cloud (VPC), subnet, or a network interface within Amazon Web Services (AWS).*
  - *When a flow log is created for a VPC, it monitors every network interface within the VPC.*

VPC Flow log

AWS – CLOUD NATIVE LOGGING SERVICES

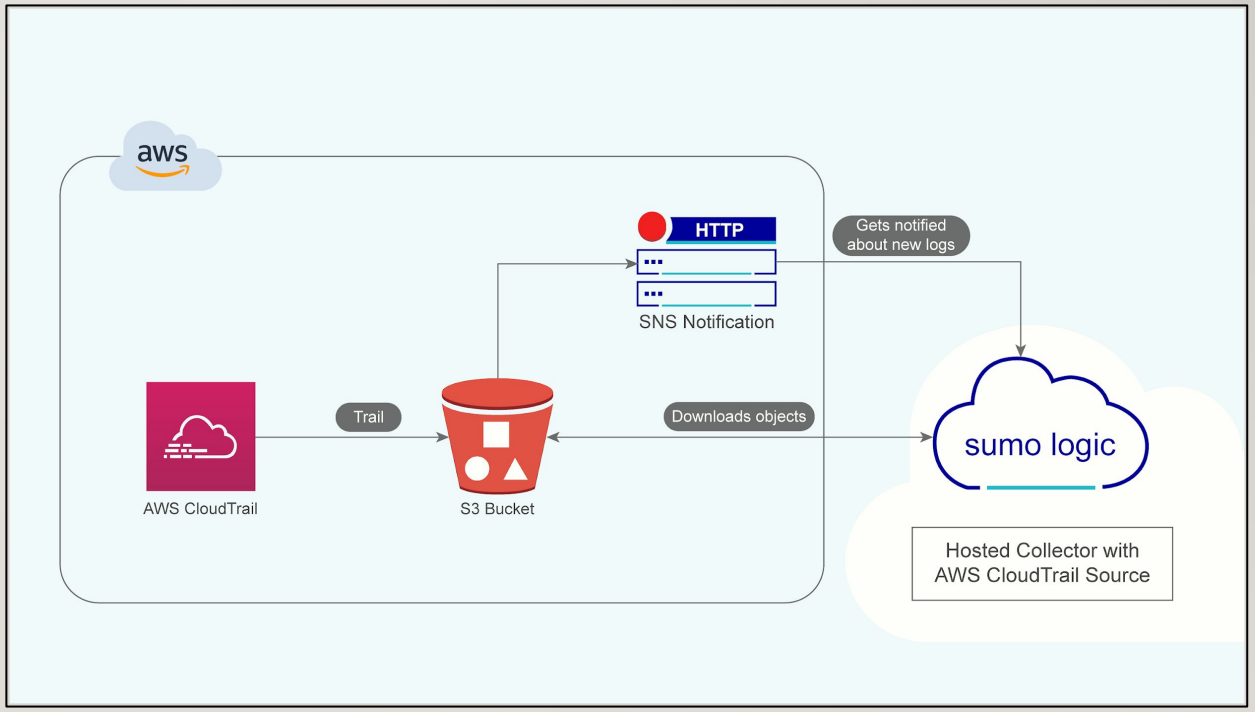
- **Route53**
  - *Amazon Route 53 is a highly available and scalable cloud Domain Name System (DNS) web service provided by AWS*
  - *From a logging perspective we will capture resolver query logs (DNS) from Route53*

AWS – CLOUD NATIVE LOGGING



AWS – DECOY LOGGING STRATEGY

- Cloud Native logging services will store their respective logs into a s3 bucket
- These logs will be streamed to a 3rd party SIEM for monitoring and alerting
- Next, we will create decoy identities, resources and include corresponding detection rules to the SIEM monitor any interaction to those decoy objects

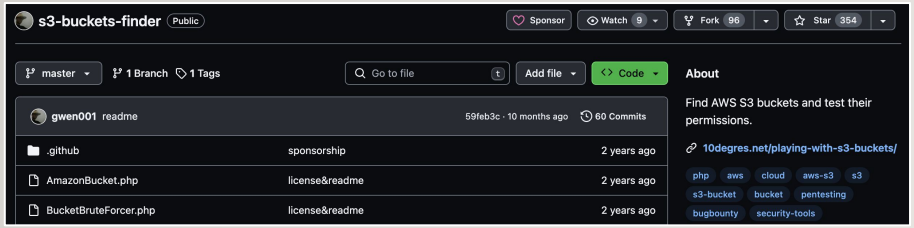


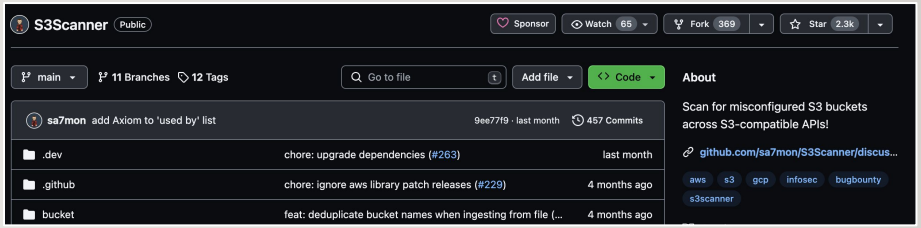
AWS – DECOY S3 BUCKETS (ATTACK VECTOR)

- AWS S3 (Simple Storage Service) is used for data storage in the cloud
- One of the most common AWS misconfiguration is Open S3 bucket (public)
- There are multiple open source tools that facilitate hunting of public s3 buckets that may belong to a specific target via brute force

AWS S3

AWS – DECOY S3 BUCKETS (ATTACK VECTOR)






```
└─$ s3scanner -bucket-file random.txt -enumerate
INFO not_exist | ubmotors
INFO exists    | gram-test | us-east-1 | AuthUsers: [] | AllUsers: [READ] | 64 objects (1.6 GB)
INFO exists    | espgb.com | eu-west-1 | AuthUsers: [] | AllUsers: [READ] | 7 objects (4.0 GB)
INFO exists    | rcscloud | us-east-1 | AuthUsers: [] | AllUsers: [READ] | 0 objects (0 B)
INFO exists    | raceview | us-east-1 | AuthUsers: [] | AllUsers: [READ] | 4217 objects (758 MB)
INFO exists    | ss-pics | eu-west-1 | AuthUsers: [] | AllUsers: []
INFO not_exist | thebanner
INFO not_exist | vidyotest
INFO exists    | sjfoto | us-east-1 | AuthUsers: [] | AllUsers: [READ] | 0 objects (0 B)
INFO not_exist | edumeme
INFO exists    | glosofobia | eu-west-2 | AuthUsers: [] | AllUsers: [READ, READ_ACP] | 806 objects (4.6 GB)
INFO not_exist | mightyeshare.com
INFO exists    | tophunter-dev | sa-east-1 | AuthUsers: [] | AllUsers: [READ] | 4 objects (17 MB)
INFO exists    | lanternarius-carrierwave-storage | us-east-1 | AuthUsers: [] | AllUsers: [READ, READ_ACP] | 4 objects (57 kB)
INFO exists    | www.mattfraser.co.nz | ap-southeast-2 | AuthUsers: [] | AllUsers: [READ] | 40 objects (495 MB)
INFO exists    | getmailhive.com | eu-west-1 | AuthUsers: [] | AllUsers: [READ, READ ACP] | 25 objects (201 kB)
INFO exists    | anus | ap-south-1 | AuthUsers: [] | AllUsers: [READ] | 2 objects (245 B)
INFO exists    | 123 | us-east-1 | AuthUsers: [] | AllUsers: []
INFO not_exist | irlshooter
INFO exists    | gt40 | us-east-1 | AuthUsers: [] | AllUsers: [READ, READ_ACP] | 2 objects (57 kB)
INFO not_exist | phamix.com
INFO exists    | bennetto | us-west-2 | AuthUsers: [] | AllUsers: [READ, READ_ACP] | 3 objects (57 kB)
INFO not_exist | alhayat
INFO exists    | storiesbot | eu-central-1 | AuthUsers: [] | AllUsers: [READ] | 2 objects (57 kB)
INFO exists    | developers | us-east-1 | AuthUsers: [] | AllUsers: []
INFO exists    | servest | eu-west-2 | AuthUsers: [] | AllUsers: [READ, READ_ACP] | 19 objects (1.3 MB)
INFO exists    | arocha | sa-east-1 | AuthUsers: [] | AllUsers: [READ, READ_ACP] | 19 objects (4.5 MB)
INFO exists    | lifetech | ap-south-1 | AuthUsers: [] | AllUsers: []
INFO not_exist | gabster-dev
INFO exists    | hdf | us-west-2 | AuthUsers: [] | AllUsers: []
INFO not_exist | files.menshealthnews.com
INFO exists    | glimages | us-east-1 | AuthUsers: [] | AllUsers: [READ] | 8515 objects (1.7 GB)
INFO not_exist | knowbe.jp
INFO exists    | static-ottera.com | us-west-2 | AuthUsers: [] | AllUsers: [READ] | 1488 objects (10 GB)
INFO exists    | diariodeobra | sa-east-1 | AuthUsers: [] | AllUsers: [READ, READ_ACP] | 4 objects (920 kB)
INFO exists    | lokalleads | eu-west-1 | AuthUsers: [] | AllUsers: [READ] | 15636 objects (11 GB)
```

```
$ php s3-buckets-bruteforcer.php --bucket ~/security/SecLists/mine/test
38267 buckets to test.

Testing: uk , not found (400)
Testing: listcat , not found (404)
Testing: c_news_show , not found (404)
Testing: theins , not found (404)
Testing: salve FOUND! (403)
Testing permissions: put ACL failed , get ACL failed , list failed , HTTP list failed , write failed ,
Testing: 1e5 , not found (404)
Testing: selections FOUND! (403)
Testing permissions: put ACL failed , get ACL failed , list failed , HTTP list failed , write failed ,
Testing: bermeo FOUND! (403)
Testing permissions: put ACL failed , get ACL failed , list failed , HTTP list failed , write failed ,
Testing: healthnotes , not found (404)
Testing: kids FOUND! (403)
Testing permissions: put ACL failed , get ACL failed , list failed , HTTP list failed , write failed ,
Testing: leoevtart FOUND! (403)
Testing permissions: put ACL failed , get ACL failed , list success , HTTP list failed , write failed ,
Testing: molodenkie , not found (404)
Testing: .master , not found (0)
Testing: rdf FOUND! (403)
Testing permissions: put ACL failed , get ACL failed , list failed , HTTP list failed , write failed ,
Testing: caroline FOUND! (403)
Testing permissions: put ACL failed , get ACL failed , list failed , HTTP list failed , write failed ,
Testing: propiedades FOUND! (200)
Testing permissions: put ACL failed , get ACL failed , list success , HTTP list success , write success ,
Testing: phpad FOUND! (403)
Testing permissions: put ACL failed , get ACL failed , list success , HTTP list failed , write failed ,
Testing: 6653 , not found (404)
Testing: seach , not found (404)
Testing: open_pub , not found (404)
Testing: 1867 , not found (404)
Testing: search-this-site FOUND! (200)
Testing permissions: put ACL failed , get ACL failed , list success , HTTP list success , write failed ,
Testing: campuslife , not found (404)
Testing: album_modcp , not found (404)
Testing: aj , not found (400)
```

AWS – DECOY S3 BUCKETS (DECEPTION)

- Usually brute-forcing works in the way that the target name is commonly used as a prefix or a suffix to commonly used keywords.
- For example – myowncompany-{brute force keyword} OR {brute force keyword}-myowncompany
- What if we deliberately create public s3 buckets with common keywords that closely monitor their activities
- For example - myowncompany-dev and myowncompany-prod
- What if we put a decoy AWS user credential file inside that s3 bucket ??

Hunting query - event.provider: "s3.amazonaws.com" and event.action: ("HeadBucket" OR GetBucket*) and aws.s3.bucket.name: "myowncompany-dev"

AWS – DECOY SES CREDENTIAL (ATTACK VECTOR)

- AWS SES (Simple Email Service) is a cloud based email service provider that can be used for high volume email automation
- Compromised SES credentials have been used in the past by threat actors for spam or phishing

BrowserStack

AWS | SES | Access Keys | Forgotten Cloud Resources

In November 2014 BrowserStack, a cloud testing platform for cross-platform testing of different applications, was breached through an old prototype machine that had not been updated and was vulnerable to the shellshock exploit. The attacker created an IAM user and generated a keypair. The attacker accessed the email list and used AWS Simple Email Service to send emails to 5,000 users falsely stating BrowserStack was shutting down.

AWS – DECOY SES CREDENTIAL (DECEPTION)

- What if we leak SES credentials of a verified sender account (decoy SES identity) ourselves but disable email sending from the same account?
- As soon as the attacker attempts to use the credential or validates the authenticity of the credential, alerts will be triggered!

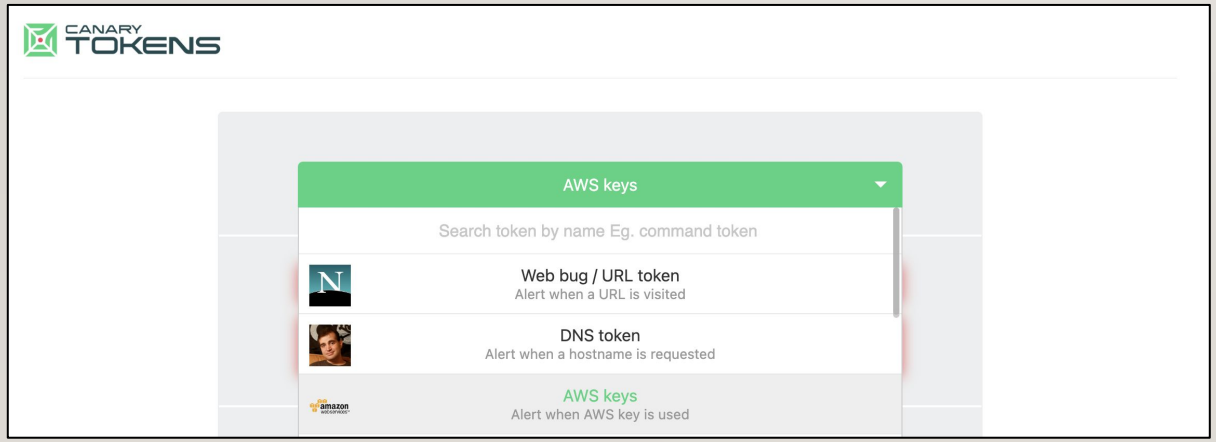
Hunting query - event.provider: "ses.amazonaws.com" and event.action: ("`ListServiceQuotas`" OR "`GetSendQuota`" OR "`GetAccount`" OR "`ListIdentities`" OR "`GetIdentityVerificationAttributes`" OR " `GetAccountSendingEnabled`" OR "`UpdateAccountSendingEnabled`") and user.id: "decoy-ses-user"

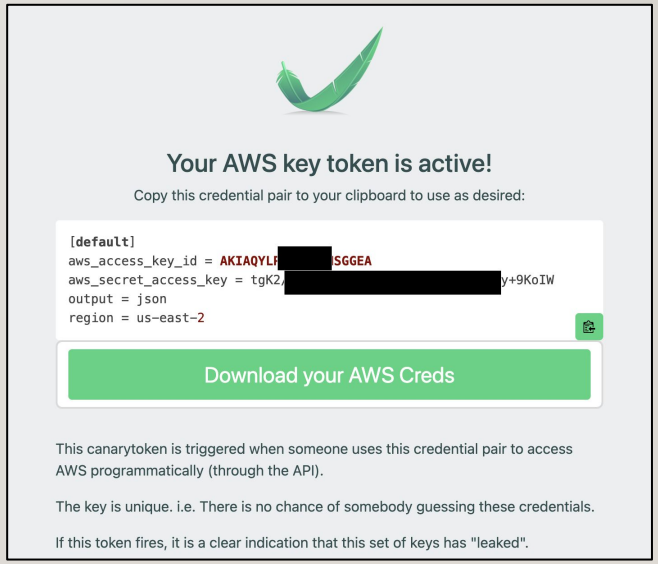
AWS – DECOY ACCESS KEYS (ATTACK VECTOR)

- AWS Access Keys are long-term credentials for an IAM user or the AWS account root user. Access keys can be used to sign programmatic requests to the AWS CLI or AWS API
- Access keys consist of two parts: an access key ID (for example, *AKIAIOSFODNN7EXAMPLE*) and a secret access key (for example, *wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY*).
- Access key leakage is one of the most prevalent cause of incidents in the cloud
- Usually its one of either accidental commitment of keys to a public code repository or developer endpoints getting compromised
- The threat actor in possession of the access key now effectively inherits all of the permissions that the user who owns the access key has been assigned

AWS – DECOY ACCESS KEYS (DECEPTION)

- Let's create a fake AWS credential pair implant it on developer machines.
- If someone tries to access AWS with the generated key pair it is highly likely that the corresponding developer machine has been compromised
- Can use canarytokens.org (is it actually Opsec safe?)





AWS – DECOY ACCESS KEYS (DECEPTION)

- Its mathematically possible to get the AWS account number given the Access key ID offline ie without interacting with AWS Apis
- Naturally *canarytokens* and other public honeypot services have a fixed number of Aws accounts (small set of AWS account numbers) that has been published by researchers
- Hence an attacker with possession of a leaked access key can determine whether it has generated using a public honeypot service

Decode the access key

As originally discovered by Aidan Steele, and later improved upon by Tal Be'ery, the account ID is actually encoded into the access key itself.

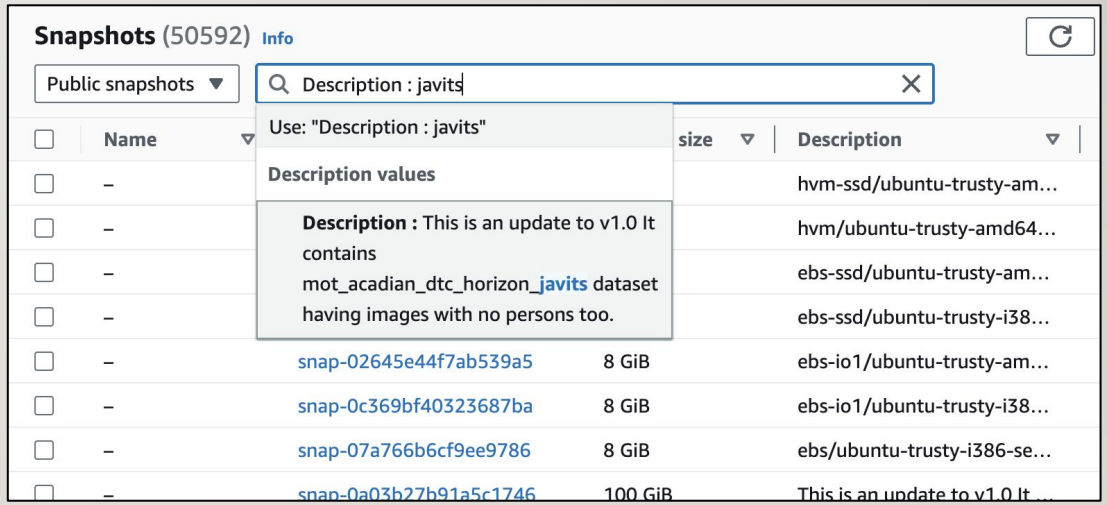
By decoding the access key using Base32 and doing a little bit shifting, we can get the account ID. Tal wrote the handy Python script below to do this:

```
import base64
import binascii

def AWSAccount_from_AWSKeyID(AWSKeyID):

    trimmed_AWSKeyID = AWSKeyID[4:] #remove KeyID prefix
    x = base64.b32decode(trimmed_AWSKeyID) #base32 decode
    y = x[0:6]

    z = int.from_bytes(y, byteorder='big', signed=False)
    mask = int.from_bytes(binascii.unhexlify(b'7fffffffff80'), byteorder='big', si

    e = (z & mask)>>7
    return (e)


print ("account id:" + "{:012d}".format(AWSAccount_from_AWSKeyID("ASIAQNZGKIQY56JQ
```

Hunting query - event.provider: ”iam.amazonaws.com” and access_key.id: ”{*decoy_access_key*}"

AWS – DECOY EBS SNAPSHOT (ATTACK VECTOR)

- Amazon Elastic Block Store (Amazon EBS) is a block-storage service designed for Amazon Elastic Compute Cloud (Amazon EC2).
- AWS allows us to take a point-in-time copy of an EBS volume (as a backup) known as a EBS snapshot that can be restored later by creating a new EBS volume from the snapshot
- EBS snapshots are usually a treasure trove of juicy info – leaked source code, private ssh keys, PII data, aws keys etc
- Countless EBS snapshots are inadvertently left publicly shared and unencrypted
- Evidently, anyone can hunt for public EBS snapshots pertaining to a particular org

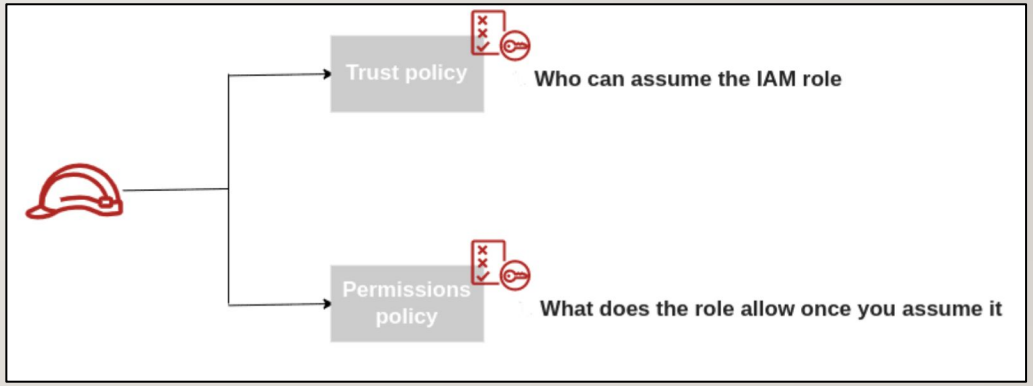


AWS – DECOY EBS SNAPSHOT(DECEPTION)

- What if we create an EBS volume with embedded decoy credentials create a snapshot add a relevant description make the snapshot public
- There is no direct way on our end to ascertain which AWS account imported the snapshot however we can alert on decoy creds/secrets planted inside the snapshot (2nd order detection)

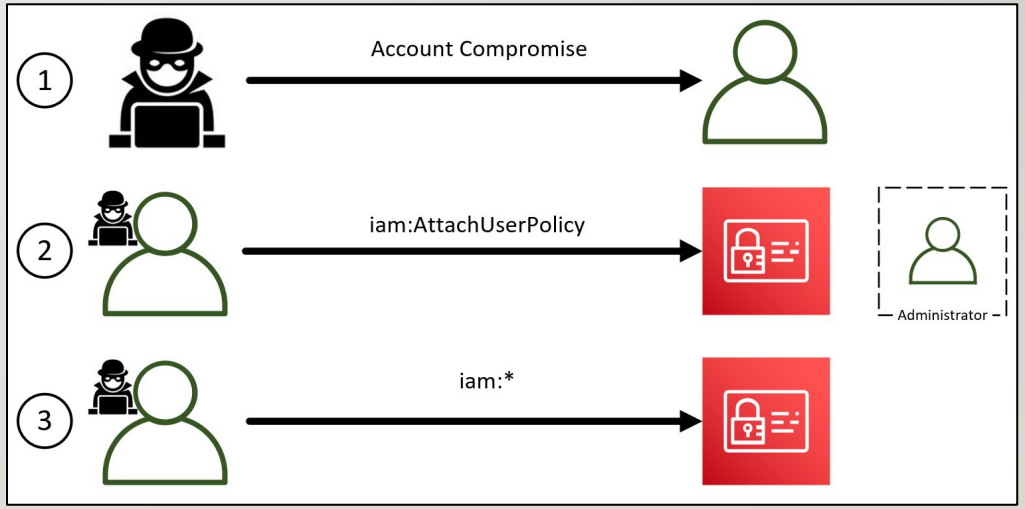
AWS – DECOY IAM ROLE (ATTACK VECTOR)

- IAM roles are entities that provide access to different AWS services based on the level of permissions they have, which makes them similar to AWS users.
- Roles do not have passwords or access keys associated with them. Instead, roles provide temporary security credentials to whomever has the ability to assume that role.
- Misconfigured IAM roles are a very common occurrence in an AWS env



AWS – DECOY IAM ROLE (DECEPTION)

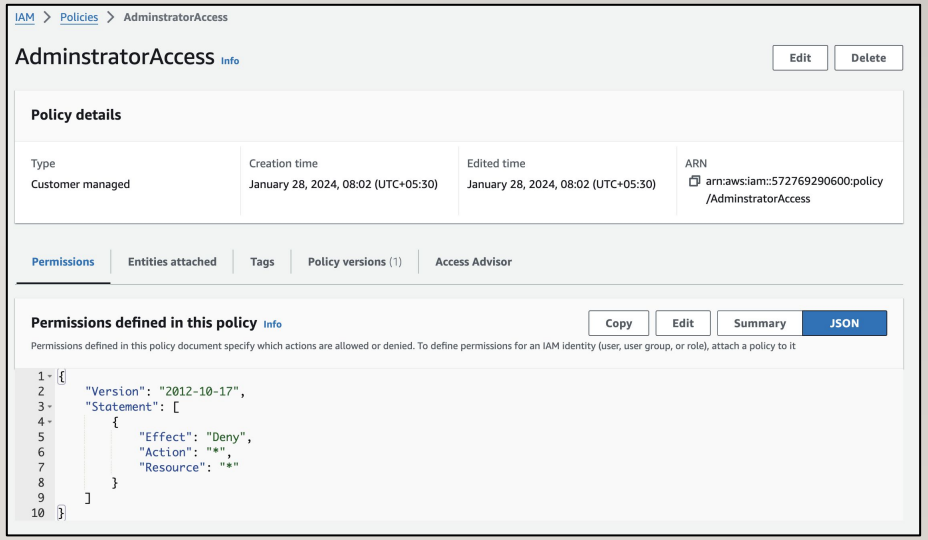
- Misconfigured IAM roles can be broadly classified into:
  - Overly Permissive Permission Policy: In this case, the JSON Policy Document of the IAM Entity, due to being overly permissive allow privesc opportunities to anyone assuming the IAM role
  - Misconfigured Trust Policy: IAM Trust policy of the role is too relaxed and allows any Principal or IAM entity assume the role and access the resources.



AWS – DECOY IAM ROLE (DECEPTION)

- The first step an attacker does in AWS env is look for lateral movement/privesc opportunities – this usually leads to assuming a vulnerable (misconfigured) role
- So why not intentionally create some vulnerable roles in our env (with restricted down permissions) and lure the attacker to assume the same

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "*"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```



Hunting query - event.provider: ”iam.amazonaws.com” and event.action: “AssumeRole” assumed_role: ”{*decoy_role*}"

AWS – DECOYS

- There are ample other avenues to introduce deception in AWS environment
- BE CREATIVE!

THANK YOU!