Fundamentals and Footguns of Cloud Security

Aditya Saligrama

Applied Cyber Weekly Meeting | March 8, 2024

Motivating Example

Request

Pretty Raw Hex

```
1 GET
  /api/dev/[redacted]/http://169.254.169.254/latest/meta-data/identity-credentials/ec2/
  security-credentials/ec2-instance HTTP/2
2 Host: [redacted]
3 Cookie: session=
  eyJlb[redacted]
  NKOS5
  VzFoS[redacted]
  4tdVhnMHZHdHZwbVJaS3NyN2siLCJ1c2VyaWQiOiJBZGl0eWEgU2FsaWdyYW1hIn0=; session.sig=
  5z1pkd1Bk3yjXGexyuuc1oByd6A
4 Sec-Ch-Ua: "Chromium";v="121", "Not A(Brand";v="99"
5 Sec-Ch-Ua-Platform: "macOS"
6 Sec-Ch-Ua-Mobile: ?0
7 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
  Chrome/121.0.6167.160 Safari/537.36
8 Content-Type: application/json
9 Accept: */*
10 Sec-Fetch-Site: same-origin
11 Sec-Fetch-Mode: cors
12 Sec-Fetch-Dest: empty
13 Referer: [redacted]
14 Accept-Encoding: gzip, deflate, br
15 Accept-Language: en-US,en;q=0.9
16 Priority: u=1, i
17
18
```

Response

Pretty Raw Hex Render

```
1 HTTP/2 200 OK
2 Content-Type: application/json; charset=utf-8
3 Date: Thu, 07 Mar 2024 23:54:39 GMT
4 Server: nginx/1.18.0 (Ubuntu)
5 Etag: W/"65a-GV7/gSuYpQNzHr0/4Rca6zuzkXY"
6 X-Powered-By: Express
7 X-Cache: Miss from cloudfront
8 Via: 1.1 b7621cdee138918b674c9cb957a70492.cloudfront.net (CloudFront)
9 X-Amz-Cf-Pop: SFO53-P6
10 Alt-Svc: h3=":443"; ma=86400
11 X-Amz-Cf-Id: unfEqRFliQ84UsEqB_TdGPgNXFPbVJSgWYroY5VAMaV0E6SVoSrFzg==
12
13 {
14   "bodytext":
   "{\n  \"Code\" : \"Success\",\n  \"LastUpdated\" : \"2024-03-07T23:04:17Z\",\n  \"Type\" : \"AWS-HMAC
   \",\n  \"AccessKeyId\" : \"ASIAQZTX7IWH3CT7QJFG\",\n  \"SecretAccessKey\" : \"[redacted]
   NKrH3doS6qNsF8NYj\",\n  \"Token\" : \"IQoJb3JpZ2luX2VjEM/////////wEaCXVzLXdlc3QtMiJHMEUCIE6rST0WmC1
   [redacted]
   5LU41KEd+dRTv/aDaoCQFU9l0VcIHIvbzvWfPQQ==\",\n  \"Expiration\" : \"2024-03-08T05:10:21Z\"\n}"
15 }
```

March 7, 2024

What is a footgun?

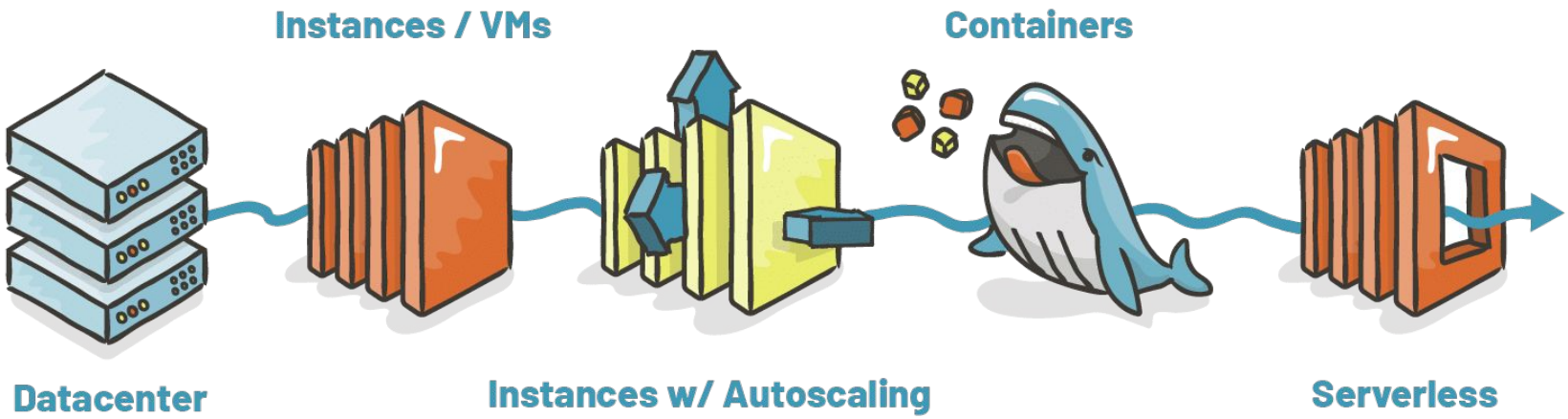
Footguns are features that are designed to do the wrong thing easily, and hard to do the right (safe) thing.

Examples:

- `strcpy`
- Firebase security rules
- 

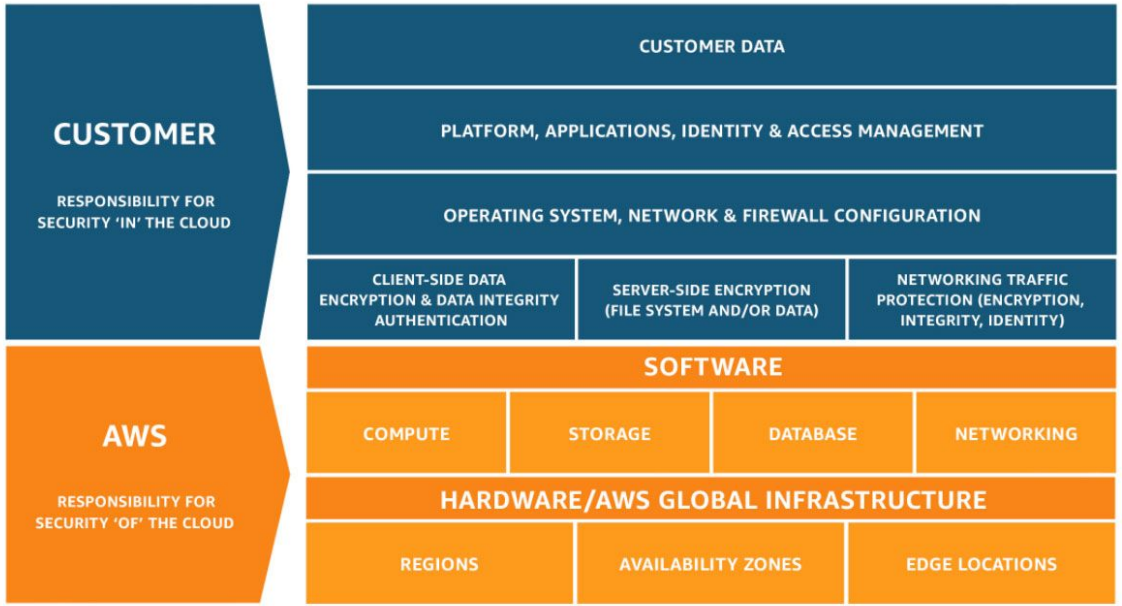
The Cloud Application Model

*The main value that public cloud providers offer is managed service abstractions above hardware. This is sometimes called **serverless computing**.*



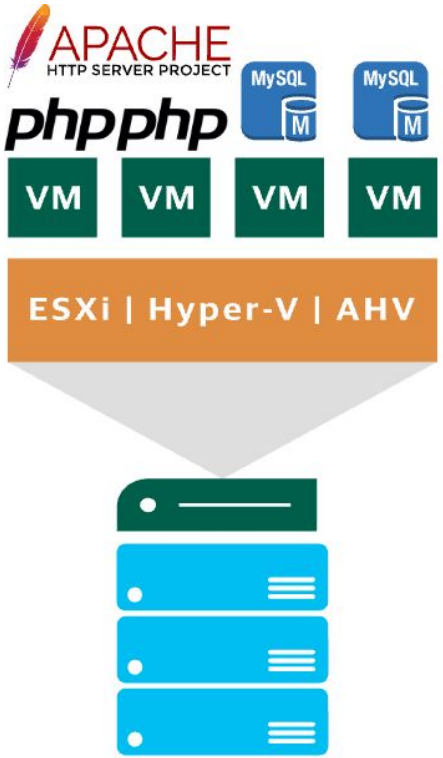
Duckbill Group – A Simple, Yet Effective Cost Optimization Framework, 2023

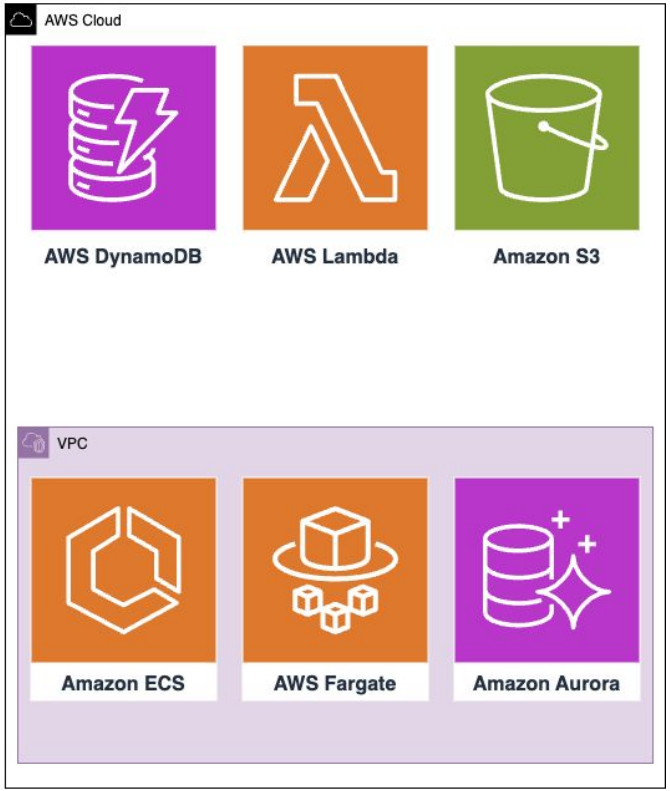
The Shared Responsibility Model

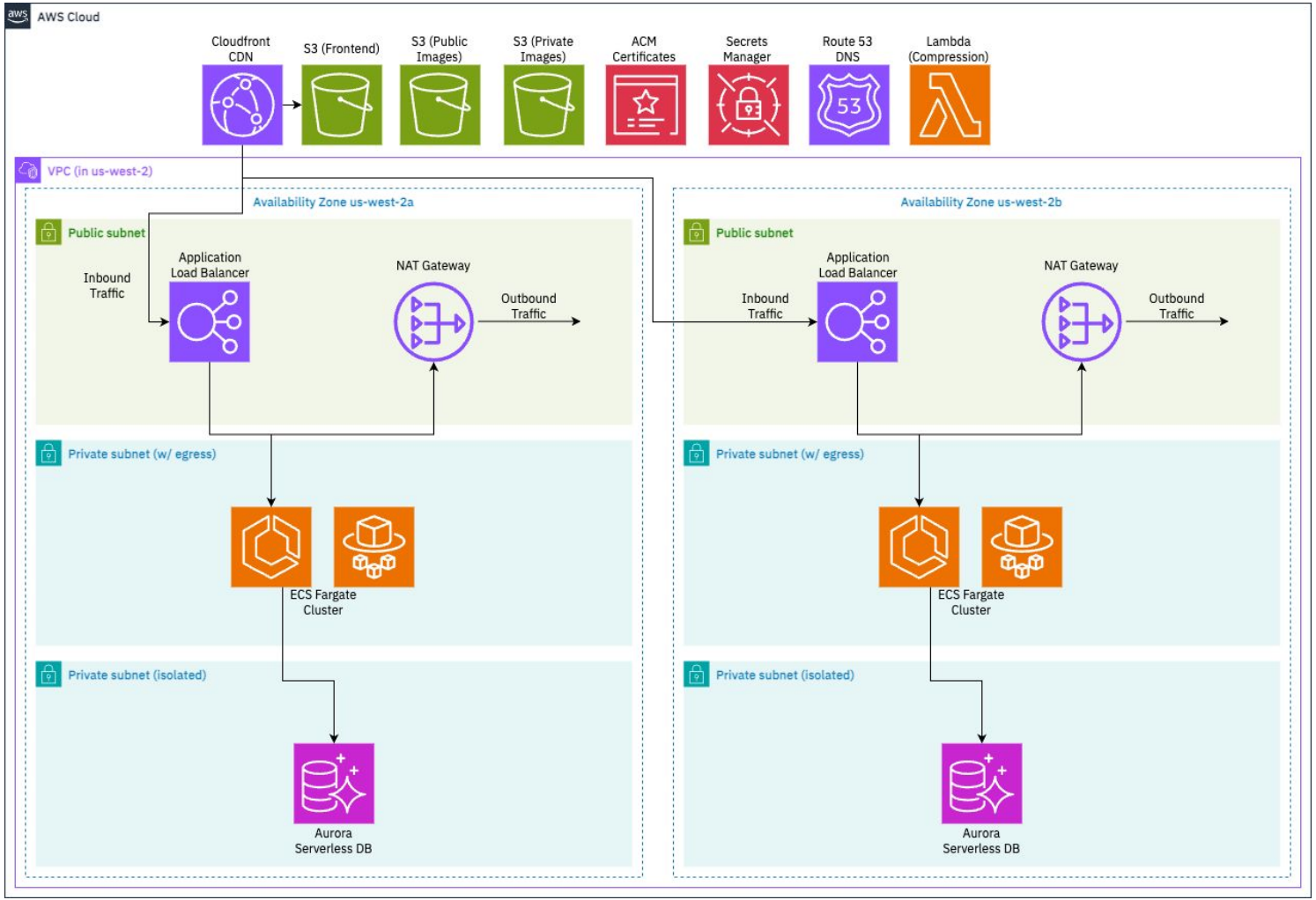


AWS assumes responsibility for its own infrastructure.
You assume responsibility for how you use AWS's infrastructure.

Traditional vs Serverless Application Architecture






AWS Cloud
Cloudfront CDN
S3 (Frontend)
S3 (Public Images)
S3 (Private Images)
ACM Certificates
Secrets Manager
Route 53 DNS
Lambda (Compression)
VPC (in us-west-2)
Availability Zone us-west-2a
Availability Zone us-west-2b
Public subnet
Inbound Traffic
Application Load Balancer
NAT Gateway
Outbound Traffic
Private subnet (w/ egress)
ECS Fargate Cluster
Private subnet (isolated)
Aurora Serverless DB

What are some common security challenges and vulnerabilities facing deployed web applications?

The OWASP Top Ten (2021)

1. Broken Access Control
2. Cryptographic Failures
3. Injection
4. Insecure Design
5. Security Misconfiguration
6. Vulnerable and Outdated Components
7. Identification and Authentication Failures
8. Software and Data Integrity Failures
9. Security Logging and Monitoring Failures
10. Server Side Request Forgery (SSRF)

Which of these does serverless design help with?

The OWASP Top Ten (2021)

1. Broken Access Control
~~2. Cryptographic Failures~~
~~3. Injection~~
4. Insecure Design
5. Security Misconfiguration
6. Vulnerable and Outdated Components
7. Identification and Authentication Failures
8. Software and Data Integrity Failures
9. Security Logging and Monitoring Failures
~~10. Server Side Request Forgery (SSRF)~~

The OWASP Top Ten (2021)

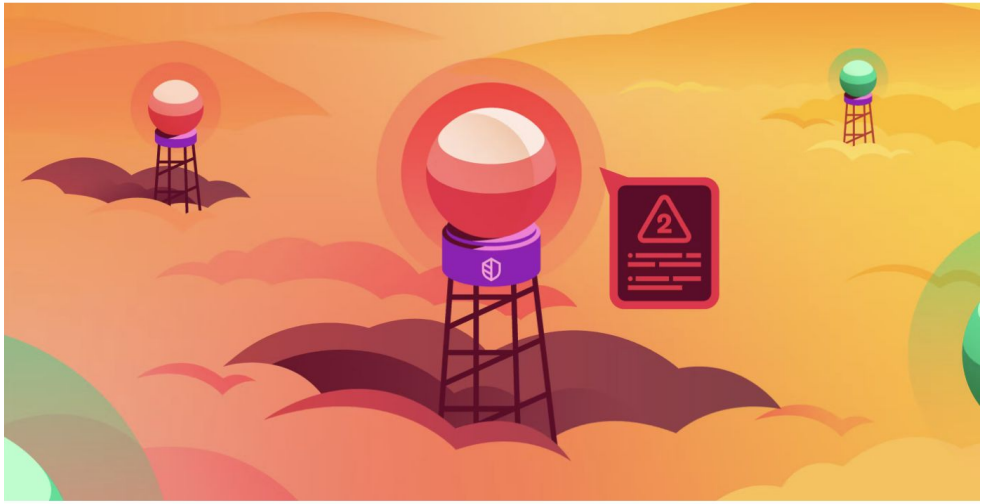
1. **Broken Access Control**
2. ~~Cryptographic Failures~~
3. ~~Injection~~
4. **Insecure Design**
5. Security Misconfiguration
6. Vulnerable and Outdated Components
7. Identification and Authentication Failures
8. Software and Data Integrity Failures
9. Security Logging and Monitoring Failures
10. ~~Server Side Request Forgery (SSRF)~~

RESEARCH

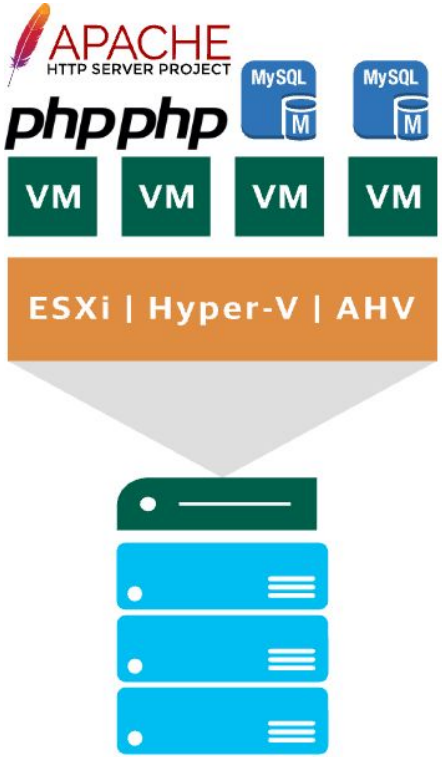
Tales from the cloud trenches: Amazon ECS is the new EC2 for crypto mining

January 19, 2024

AWS THREAT DETECTION



Why are access controls and permissions important?

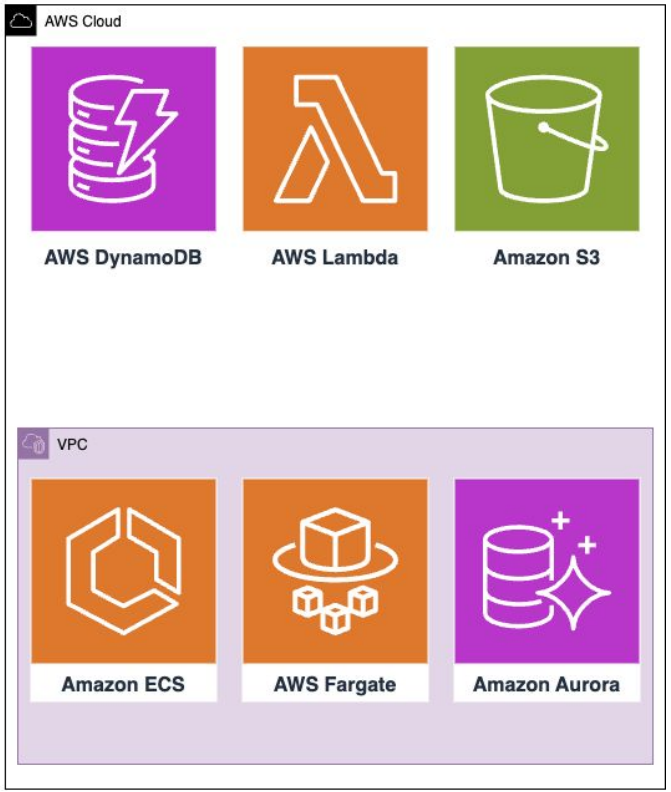


To get sensitive data from this setup, we need to compromise the PHP web app to get DB credentials (and maybe a network posture we can hit the DB from).

***Relevant initial attack vectors**:*

- *Command injection*
- *LFI + upload filter bypass*
- *SQL injection*
- *etc.*

Why are access controls and permissions important?



To get sensitive data from this setup, we need to get authentication to the AWS account and authorization to access the cloud data (DB and S3) resources.

***Relevant initial attack vectors**:*

- *Exposed AWS credentials*
- *S3 buckets with public access*
- *CI/CD code compromise/supply chain*

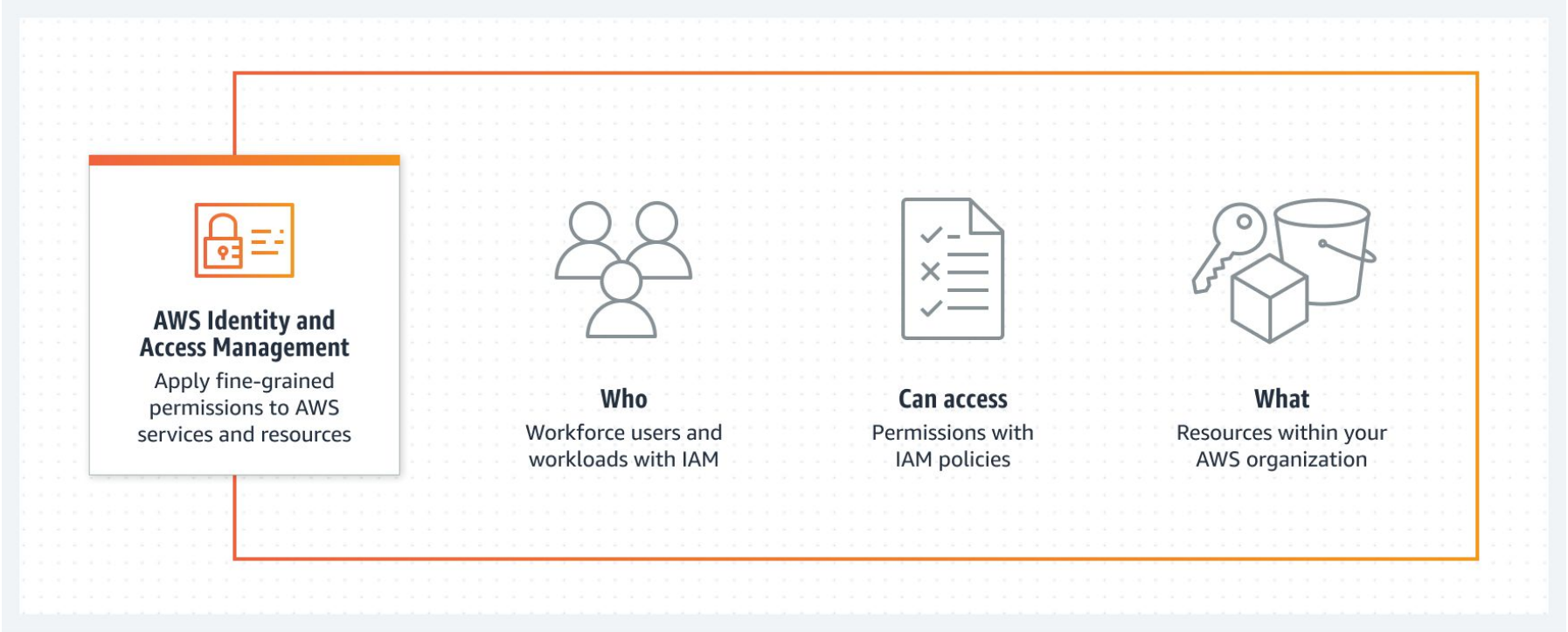
Methods of Accessing AWS

- AWS Console (“clickops”)
- AWS CLI
- AWS Software Development Kit (SDK)

*An interaction with AWS via any of these methods creates an API call (an **Action**).*

Identity and Access Management on AWS

IAM Conceptual Model



Key IAM Definitions (Agent-Side)

- **Principal**: A human user or workload that can make a request for an action or operation on an AWS resource
  - e.g. *Your user account using the AWS CLI*, or *code running on an EC2 instance*

- **Role**: An IAM construct that can be assigned scoped permissions
  - Principals can be assigned, or *assume*, roles; multiple principals can assume a single role
  - Each principal can only assume one IAM role at a time, but may have permissions for multiple

- **Policy**: A listing of the permissions that IAM roles are given
  - Written in JSON
  - e.g. *Allow read and write to all S3 buckets starting with `applied-cyber`*

Key IAM Definitions (Resource-Side)

- **Resource**: Objects within AWS services
  - e.g. *EC2 VMs*, *S3 buckets*

- **Action**: Operations performed on resources, specific to services
  - e.g. *create an EC2 VM*, *list objects in an S3 bucket*

- **Policy**: A listing of the permissions that govern access to the resource itself
  - e.g. *deny public downloads from the S3 bucket*

*Given a **principal*** assuming a ***role*** *who wants to perform an **action** on a given **resource**, AWS decides whether to authorize or deny the request by evaluating the role's or resource's **policy**.*

Example Role Policy (1)

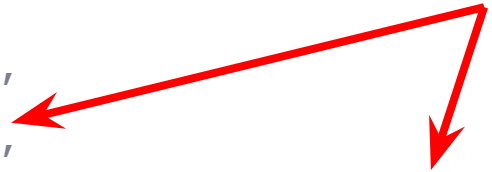
```
{
  "Version": "2012-10-17",
  "Statement": [{
    "Effect": "Allow",
    "Action": "*",
    "Resource": "*"
   }]
}
```

AdministratorAccess: Allow every action on every resource

Example Role Policy (2)

```
{ "Version": "2012-10-17",
  "Statement": [{
     "Effect": "Allow",
     "Action": "rds:*",
     "Resource": ["arn:aws:rds:region:*:*"]
   }, {
     "Effect": "Allow",
     "Action": ["rds:Describe*"],
     "Resource": ["*"]
   }]
}
```

Wildcards allowed in ARNs and actions



IAM Roles: Attaching Policies to Principals

- IAM roles are a way to temporarily grant specific permissions to specific principals
  - Principal *assumes* role that has policies (allow / deny) *attached*

- Two components
  - **Permission Policy**: *What can the role do?* (previous slides)
  - **Trust Policy**: *Who can assume the role?*

Assuming IAM Roles

- Access to roles is granted via *Security Token Service* (STS)

```
aws sts assume-role \
    --role-arn arn:aws::iam:123456789012:role/my_role \
    --role-session-name my_session
```

- Outputs:
  - *Access Key ID*
  - *Access Key Secret*
  - *Session Token*
  - Setting as environment variables for AWS API calls (via CLI) grants access to role permissions

Note: AWS services assume roles through internal STS API calls.

IAM Role Trust Policies

Motivation: don't want arbitrary principals to assume roles with access to sensitive resources.

```
{
  "Effect": "Allow",
  "Principal": {
    "AWS": "arn:aws:iam::111122223333:user/saligrama"
  },
  "Action": "sts:AssumeRole"
}
```

All trust policies apply to **Principals** and allow the **`sts:AssumeRole`** action.

IAM Role Trust Policies

Motivation: don't want arbitrary principals to assume roles with access to sensitive resources.

```
{
  "Effect": "Allow",
  "Principal": {
    "Service": "ecs.amazonaws.com"
  },
  "Action": "sts:AssumeRole"
}
```

Trust policy principals can be services, too!

Let's play around with cloud (in)security! https://flaws2.cloud