SSRF in AWS hosted applications

+ some cloud introduction

Summary

1. About cloud and AWS
2. Most common services
3. Types of access credentials
4. How services get permissions
5. What is SSRF
6. Stealing access credentials from web app in EC2 instance
7. Stealing credentials from Lambda Functions
8. Read event data in Lambda Functions
9. Accessing services protected by ACLs
10. Demo SSRF + going extra
11. Learning resources
12. Final thoughts
13. Future research

whoami

- Eduard Agavriloae
- Penetration tester at KPMG
- Worked as web developer
- Area of expertise: web penetration testing
- But also know other things, including cloud penetration testing (mostly AWS)

1. About cloud and AWS

- A network of remote servers that offers computational power and various functionalities
- AWS is the cloud provider with the biggest market share, followed by Azure and then GCP
- Over 250 services: SDLC, IoT, virtual desktops, machine learning and more
- It offers scalability, high degree of configuration and availability

2. Most common services
2.1 EC2 (Elastic Compute Cloud)

- Remote VM
- All kind of images: Amazon Linux, macOS, Windows, RedHat, Ubuntu, Kali etc.
- You can upload your own images or choose from other sources
- After the instance is launched, you get an address that uniquely identify your instance (e.g., ec2-3-79-135-11.compute-1.amazonaws.com) along with a public (3.79.135.11) and private IP
- Here you can expose all kind of services on various ports
- Configurable inbound and outbound rules

2. Most common services
2.2 Lambda

- Create “serverless” functions in various runtimes (Python, NodeJS, PHP, .NET, Go etc.)
- The execution of the function can be triggered at various events, known as Triggers (HTTP request to API Gateway endpoint, message queue, S3, IoT, Database event etc.)
- The result of the execution is returned to the trigger in some cases, or it can be sent to various places, known as Destinations

2. Most common services
2.3 S3 (Simple Storage Service)

- Storing units commonly known as S3 buckets
- Each bucket contains files, each file can be configured as public or not
- If configured improperly, a bucket can be listed from the internet of from other AWS accounts
- Example of URIs:
  - s3://<bucket-name>/<path-to-file>
  - https://<bucket-name>.s3.<region>.amazonaws.com/<path-to-file>
- A bucket can be listed either from AWS CLI, by performing an HTTP request to the root path of the bucket or by using desktop tools (be careful what tool you’re using)

2. Most common services
2.4 API Gateway

- It mostly forwards requests and returns responses
- Supports as authorization (on basic level): IAM based and/or API key
- Can be REST (public or private), HTTP or WebSocket API
- A private REST API is not accessible outside the AWS network (but if misconfigured can be accessed from other AWS accounts)
- URI:
  - Format: https://<api-id>.execute-api.<region>.amazonaws.com/<stage>/<resource>
  - Example: https://t9ftymybp8.execute-api.eu-central-1.amazonaws.com/dev/content

2. Most common services
2.5 Others

- Monitoring: CloudTrail, CloudWatch
- Network: VPC (Virtual Private Network), Route53
- Access management: IAM (Identity & Access Management)
- Databases: DynamoDB, DocumentDB
- Services with asynchronous messaging and queues: SNS (Simple Notification Service), Amazon SQS (message queuing service)
- Containers: Elastic Container Service, Elastic Kubernetes Service

3. Types of access credentials
3.1 AWS account vs user account

AWS account

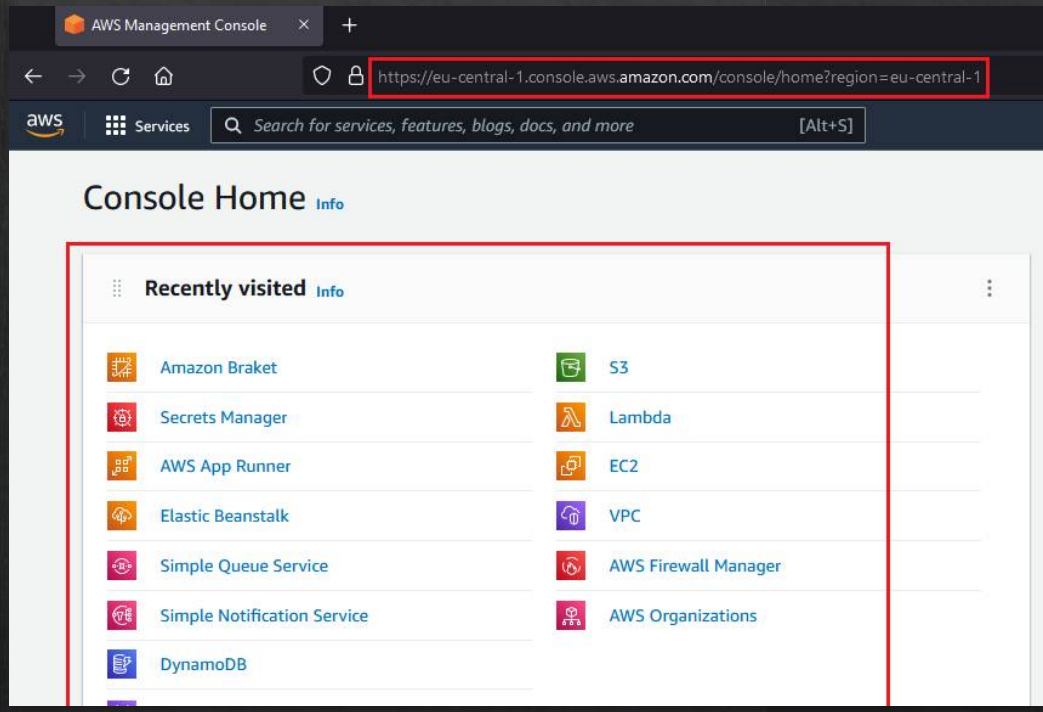
- Represents the environment composed of your configurations and resources
- In an AWS account you use services like EC2 and Lambda
- An AWS account contains user accounts

User account

- Is an identity used to login into an AWS account
- Is part of an AWS account
- Can access resources within the AWS account based on its permissions

3. Types of access credentials
3.2 Web console

- Browser based UI
- You need username and password (+ 2fa if configured)
- You can login either as root user or as a user that’s part of an AWS account (case in which you need to know the account id)
- Login at:
  - https://console.aws.amazon.com
  - https://<account-id>.console.aws.amazon.com



3. Types of access credentials
3.3 Persistent access credentials

- Allocated to users
- They do not expire and are static once created
- Used for interacting with AWS APIs either directly (request signing) or via AWS CLI
- Example of persistent access credentials:
  - Access Key ID: AKIAIOSFODNN7EXAMPLE
  - Secret Access Key: wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
- Using them provides the same level of permissions as when using the web console

3. Types of access credentials
3.4 Temporary access credentials

- Generated when assuming a role
- A role is an identity that can be assumed to gain a set of permissions
- Roles can be assumed by users or services if the proper permissions are in place
- Example of temporary access credentials:
  - Access Key ID: ASIAIOSFODNN7EXAMPLE
  - Secret Access Key: je7MtGbClwBF/2Zp9Utk/h3yCo8nvbEXAMPLEKEY
  - Session Token: AQoEXAMPLEH4aoAH0gNCAPyJxz4BlCFFxWNE1OPTgk5TthT+Fvwq[…]
- Configurable expiration time: 15 minutes to 12 hours
- A role can be assumed either from the web console or AWS CLI

4. How services get permissions
4.1 General aspects

- Through service roles
- Service roles are simple roles that are designated to a specific service type
- A Lambda function can’t have allocated a role designated to an EC2 instance
- Once a service was configured to use a role, the role assumption is handled by AWS in the backend

4. How services get permissions

4.1 EC2 instances

- An EC2 instance gets the temporary access credentials for its associated role by querying a local API called Metadata Service
- The API is always at the next address: http://169.254.169.254/
- A GET request is must be sent to the next address in order to retrieve the credentials:
  - http://169.254.169.254/latest/meta-data/iam/security-credentials/<role-name>/
- You can find the role name by making a GET request to:
  - http://169.254.169.254/latest/meta-data/iam/security-credentials/
- Two versions of Metadata Service:
  - IMDSv1 (default option in EC2) doesn’t provide any security mechanism to protect against SSRF
  - IMDSv2 (default option in Elastic Beanstalk) requires a security token to query for credentials, that is given by making a PUT request to http://169.254.269.254/latest/api/token with the custom header "X-aws-ec2-metadata-token-ttl-seconds: <TTL-in-seconds>“ (TTL max value is 21600 seconds = 6 hours)

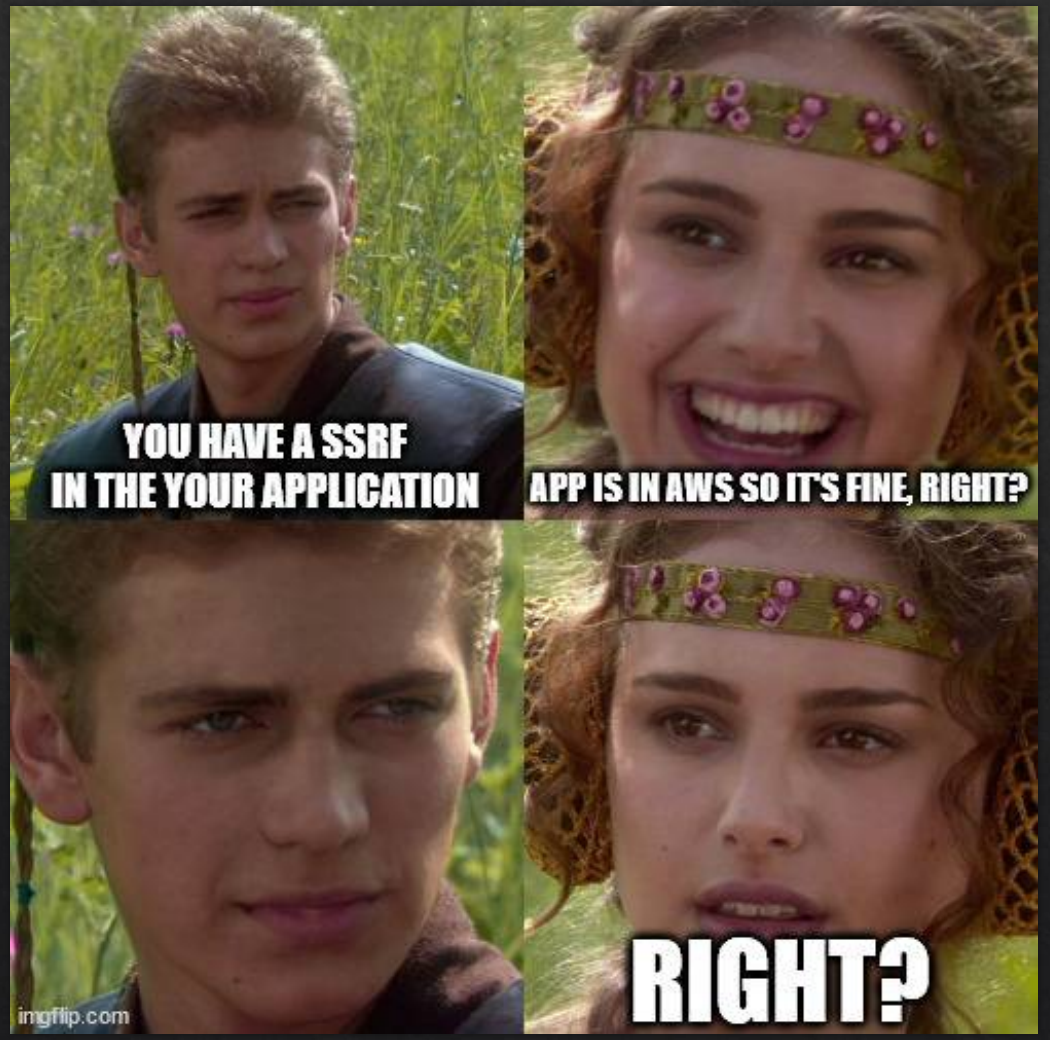
4. How services get permissions
4.2 Lambda Functions

- You can allocate a role to a Lambda Function in order to give it a set of permissions (known as execution role)
- At invocation time, the Lambda Function will assume the role
- The temporary access credentials are stored in the next environment variables:
  - AWS_ACCESS_KEY / AWS_ACCESS_KEY_ID
  - AWS_SECRET_ACCESS_KEY
  - AWS_SESSION_TOKEN

5. What is SSRF

- Server-Side Request Forgery is a vulnerability/attack that occurs when you can make the server to perform requests to arbitrary locations in your name
- Can occur from a logic flaw, missing input validation or from other vulnerabilities like:
  - XXE injection (XML external entity)
  - Server-Side template injection
  - Open redirect
  - Header injection
  - Code injection

5. What is SSRF

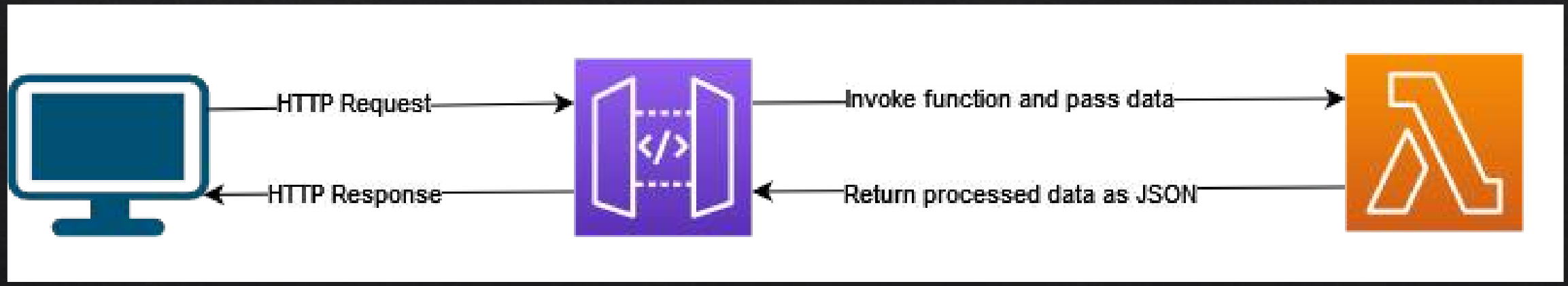


6. Stealing access credentials from web app in EC2 instance

- Metadata service version 1
- There is no way to know each version is in place without trial and error
- You need to exfiltrate data to steal credentials
- The web app might not have a role allocated, so no credentials to steal
- Only two requests necessary:
  - http://169.254.169.254/latest/meta-data/iam/security-credentials/
    - This will return the role's name
  - http://169.254.169.254/latest/meta-data/iam/security-credentials/<role-name>/
    - This will return the access credentials
- Put the credentials in your AWS CLI session and that's it, now you have the same permissions as the web application

7. Stealing credentials from Lambda Functions
7.1 Common way to expose a Lambda Function

- Lambda Functions can’t be accessed directly over the internet with HTTP requests
- They can however be integrated with an API Gateway that will invoke the function
- The API Gateway will forward all the information received by the client to the Lambda environment (headers, query parameters, body parameters etc.), plus additional information gathered by the API Gateway (client IP, API Key, AWS account ID etc.)

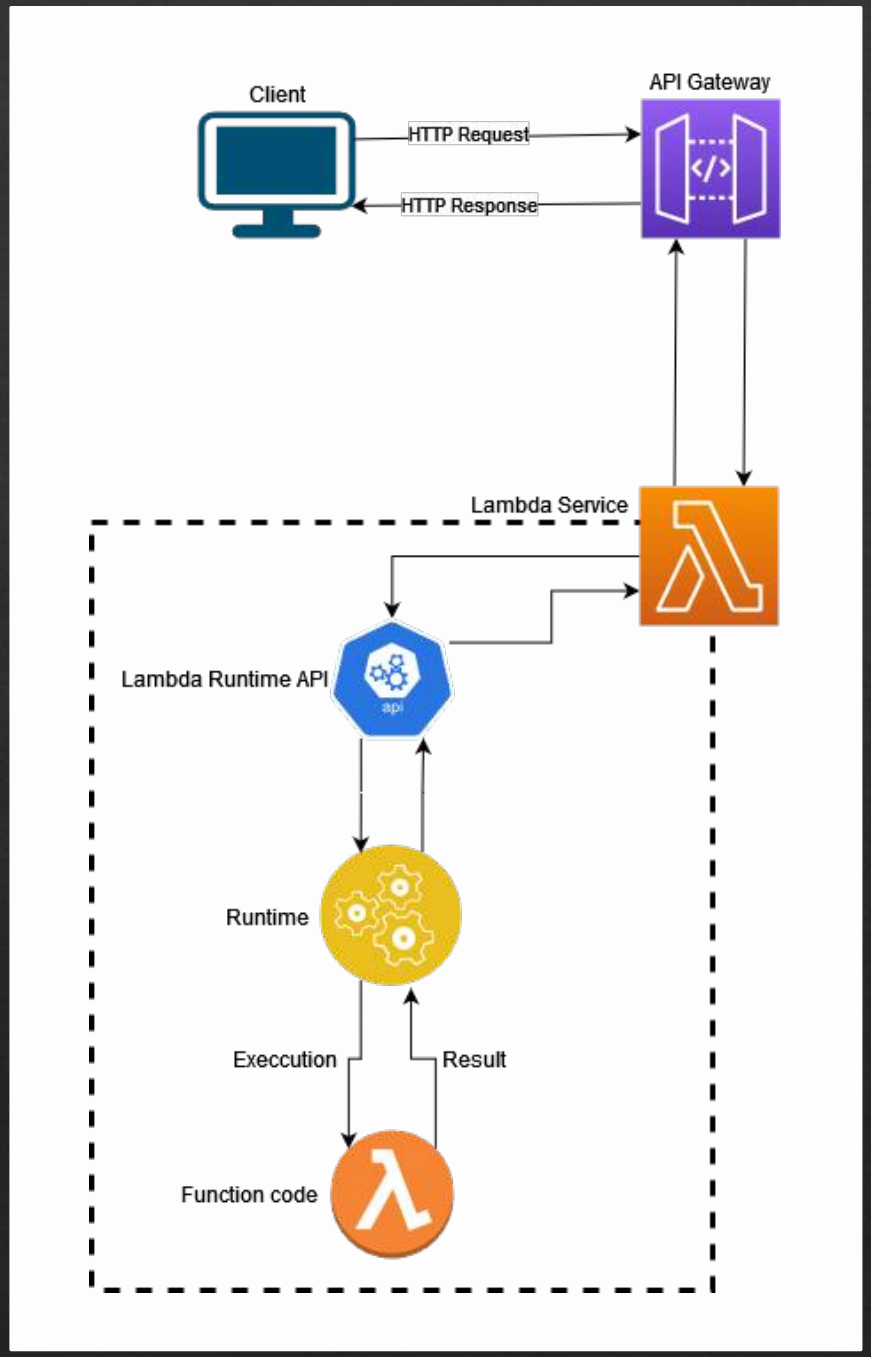


7. Stealing credentials from Lambda Functions
7.2 Get credentials through SSRF/File Inclusion

- You need to be able to exfiltrate the output
- If the function has a SSRF vulnerability that doesn’t validate schema you might be able to read the environment variables with the payload “file:///proc/self/environ”
- However, if the content is not encoded (e.g., with UTF-8), this will throw an error since it contains invalid characters
- But, if you’re lucky, you’ll get the environment variables of the Lambda Function, which will contain the temporary access credentials

8. Read event data in Lambda Functions

- Event data contains Client and API Gateway data
- The runtime needs to get the information required for the current invocation
- For this, it sends a GET request to the Lambda Runtime API
- The response will contain all the data send by the client, plus the information inserted by the API Gateway



8. Read event data in Lambda Functions

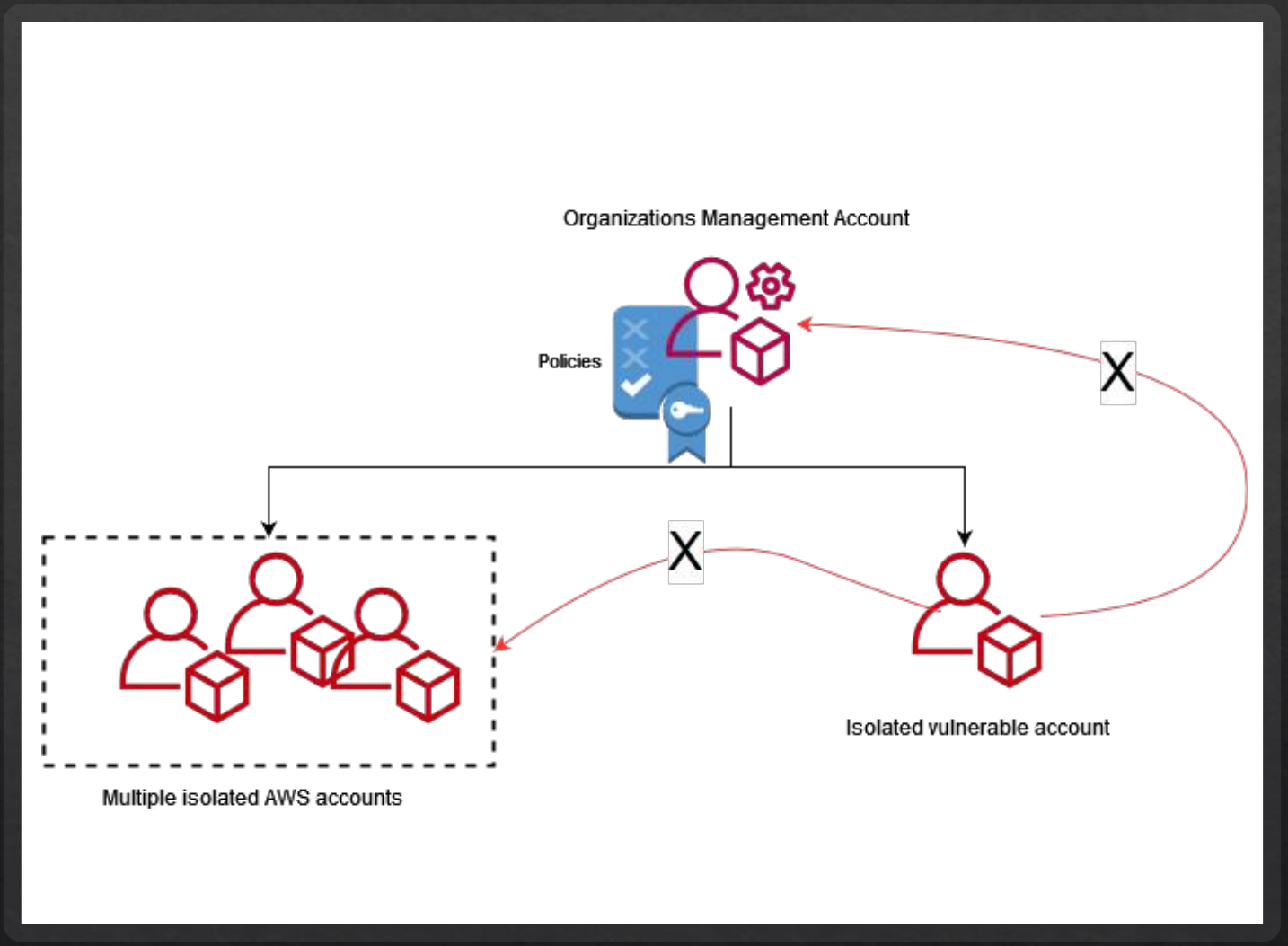
- However, if a SSRF vulnerability is present, we can query the Lambda Runtime API in order to disclose various information like account ID, stage variable, API keys, API id, access key ID and more
- By default, the Lambda Runtime API runs at http://127.0.0.1:9001/
- To get the event data, we need to make a GET request to:
  - http://127.0.0.1:9001/2018-06-01/runtime/invocation/next
- The response contains the information for the current invocation, meaning that querying multiple times the mentioned endpoint would result in the same output for the current execution

9. Accessing services protected by ACLs

- People often configure ACLs to allow only specific IPs to certain services
- If you know the IP/domain of another service that can’t be accessed from your IP, you might be able to target it using a SSRF in another service
- Another thing people do is allowing network communication with all AWS IPs
  - In this case it’s enough to launch an EC2 instance in your account, connect to it and interact with that service

10. Demo

- Using Organizations
- You shouldn’t be able to do much even if you elevate privileges to administrator
- But if you do, let me know how you did it, the beer is on me

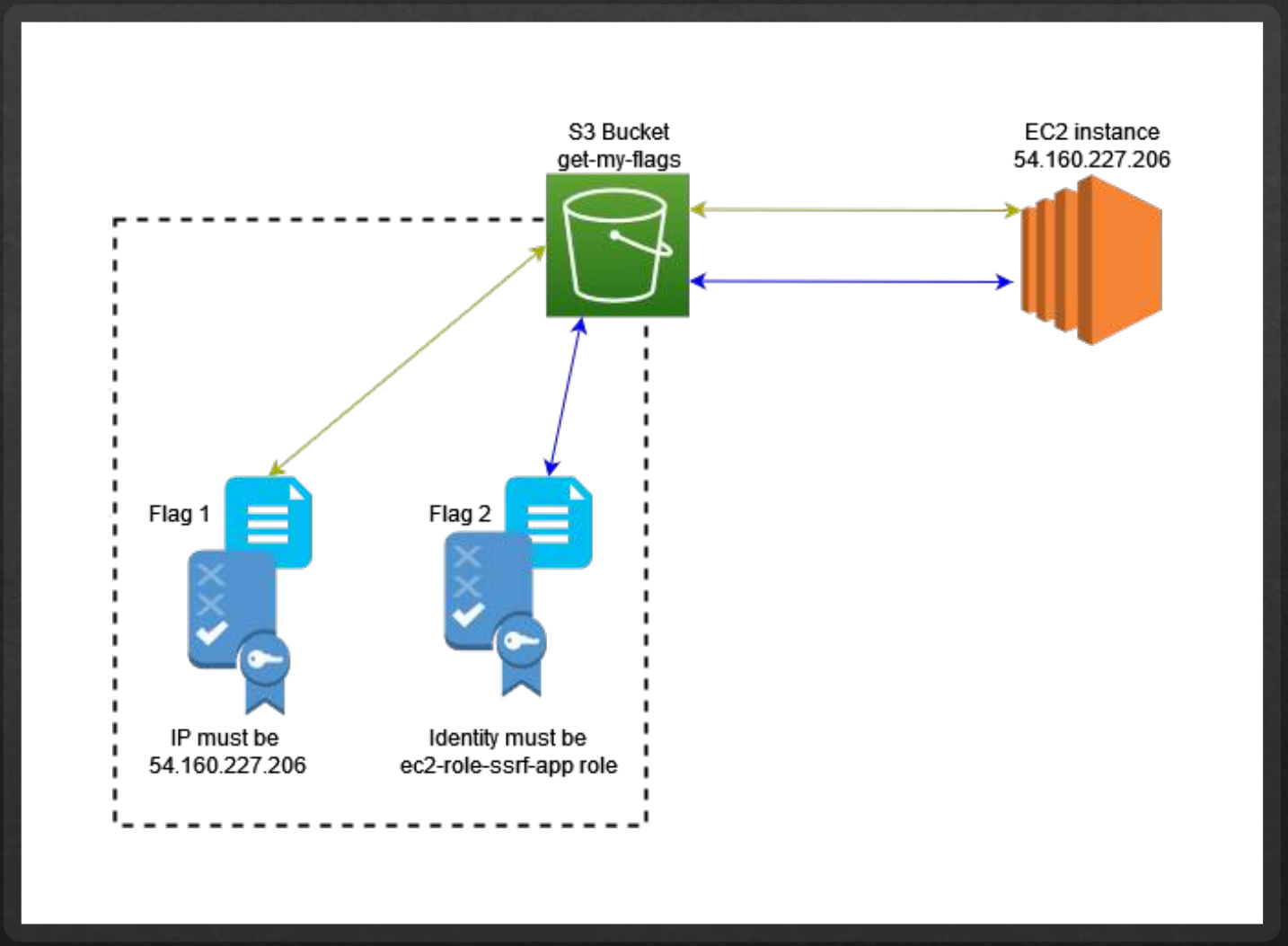


10. Demo

- Stealing credentials from an EC2 instance
- Stealing credentials from Lambda Function integrated in API Gateway
- Exfiltrating event data from Lambda Function
- Bonus: Privilege Escalation scenario

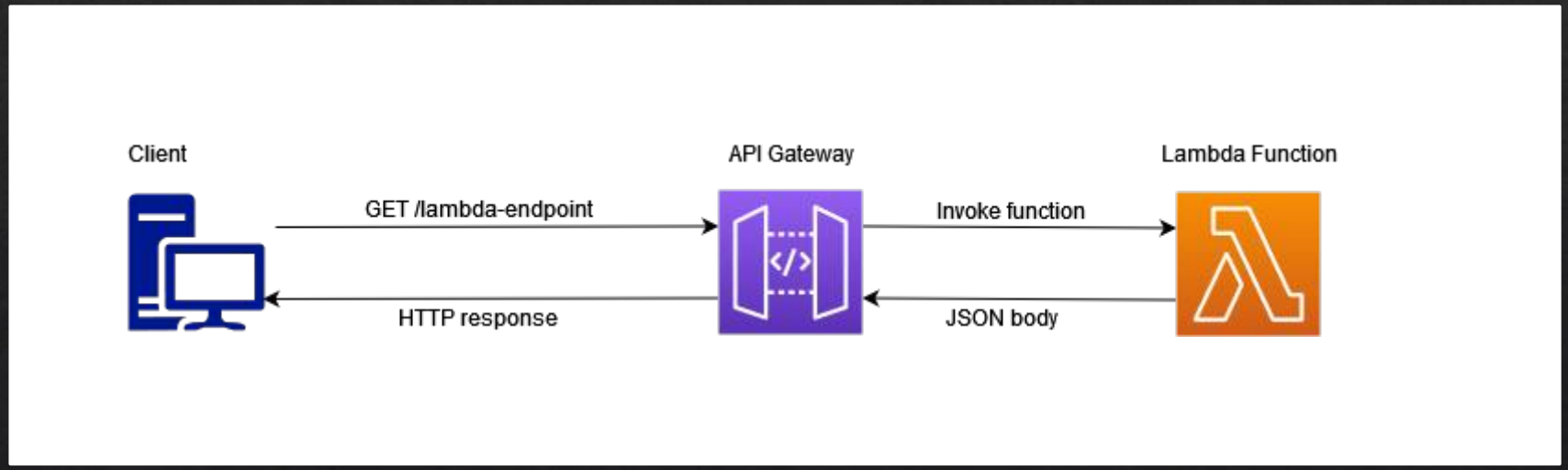
10. Demo
10.1 EC2 instance

- A vulnerable web application at http://ec2-54-160-227-206.compute-1.amazonaws.com/
- Must get 2 flags:
  - First flag: from the EC2's IP or using valid credentials
  - Second flag: only by valid credentials
- Privilege escalation scenario based on the permissions allocated to the EC2 instance



10. Demo
10.2 Steal event data from Lambda Function

- Endpoint within an API Gateway that’s integrated with a Lambda Function in the backend
- Impact is greater if the API Gateway is behind another application that adds API key and other information



11. Learning resources

Theory

- Rhino Security Labs Blog
- Privilege escalation vectors
- AWS Security Documentation
- PentesterAcademy AWS Bootcamp recordings – paid

Practice

- http://flaws.cloud
- http://flaws2.cloud
- CloudGoat
- PentesterAcademy AWS Labs - paid

12. Final thoughts

- Using cloud doesn’t mean you are secure by default
- A SSRF in cloud can have a level of impact close to a code injection vulnerability
- Compromising an AWS account is not the same as infiltrating inside an organization’s network, but having a central way to enumerate and compromise everything in the environment falls in the attacker’s advantage
- A classic penetration test against a web application hosted in cloud is not sufficient
  - Configuration review of the environment is required to identify misconfigurations that can be leveraged by attackers in the further compromise of the account

13. Future Research

- SSRF impact in cloud applications running inside docker containers
- SSRF impact in cloud applications running in a node from a Kubernetes cluster

Q&A