# Wiz Home Assignment – Or Benson:

**Question 1:**

1.
The JSON payload for an instance is provided below. Please note that information which can be used to determine if this specific instance is using IMDSv1 is bolded and highlighted in the payload.

To identify whether an instance is using IMDSv1, if the HttpTokens output returns "optional" (instead of "required") Instance Metadata Service Version 1 (IMDSv1) is in use for the instance, so IMDSv2-only configuration is not enforced. In addition, if the 169.254.169.254 address is listed as a local IP address, this may indicate the use of IMDSv1 on the instance.

This JSON is the output of running the AWS CLI command `aws ec2 describe-instances --region us-east-1 --instance-ids i-057328b772102801d` (using the relevan instance ID and region) after configuring the CLI with private keys for the EC2 user.


```
{
    "Reservations": [
        {
            "Groups": [],
            "Instances": [
                {
                    "AmiLaunchIndex": 0,
                    "ImageId": "ami-0b5eea76982371e91",
                    "InstanceId": "i-057328b772102801d",
                    "InstanceType": "t2.micro",
                    "KeyName": "aws-ec2-orbenson",
                    "LaunchTime": "2022-12-21T08:31:40+00:00",
                    "Monitoring": {
                        "State": "disabled"
                    },
                    "Placement": {
                        "AvailabilityZone": "us-east-1d",
                        "GroupName": "",
                        "Tenancy": "default"
                    },
                    "PrivateDnsName": "ip-172-31-87-93.ec2.internal",
                    "PrivateIpAddress": "172.31.87.93",
                    "ProductCodes": [],
                    "PublicDnsName": "ec2-44-201-211-171.compute-1.amazonaws.com",
                    "PublicIpAddress": "44.201.211.171",
                    "State": {
                        "Code": 16,
                        "Name": "running"
                    },
                    "StateTransitionReason": "",
                    "SubnetId": "subnet-0cbdaebd8c8e2fe91",
                    "VpcId": "vpc-095498a76ffd2ba23",
```

```
                    "Architecture": "x86_64",
                    "BlockDeviceMappings": [
                        {
                            "DeviceName": "/dev/xvda",
                            "Ebs": {
                                "AttachTime": "2022-12-21T08:31:40+00:00",
                                "DeleteOnTermination": true,
                                "Status": "attached",
                                "VolumeId": "vol-0049355ab9f091016"
                            }
                        }
                    ],
                    "ClientToken": "f484ea8e-2b08-4cc2-92f8-b57438007200",
                    "EbsOptimized": false,
                    "EnaSupport": true,
                    "Hypervisor": "xen",
                    "NetworkInterfaces": [
                        {
                            "Association": {
                                "IpOwnerId": "amazon",
                                "PublicDnsName":
"ec2-44-201-211-171.compute-1.amazonaws.com",
                                "PublicIp": "44.201.211.171"
                            },
                            "Attachment": {
                                "AttachTime": "2022-12-21T08:31:40+00:00",
                                "AttachmentId": "eni-attach-055f4ae12dd224f8d",
                                "DeleteOnTermination": true,
                                "DeviceIndex": 0,
                                "Status": "attached",
                                "NetworkCardIndex": 0
                            },
                            "Description": "",
                            "Groups": [
                                {
                                    "GroupName": "launch-wizard-3",
                                    "GroupId": "sg-0cad4992ee8ec7a73"
                                }
                            ],
                            "Ipv6Addresses": [],
                            "MacAddress": "12:2a:19:7e:d8:27",
                            "NetworkInterfaceId": "eni-0005887e4e811bf9c",
                            "OwnerId": "465460162145",
                            "PrivateDnsName": "ip-172-31-87-93.ec2.internal",
                            "PrivateIpAddress": "172.31.87.93",
                            "PrivateIpAddresses": [
                                {
                                    "Association": {
                                        "IpOwnerId": "amazon",
                                        "PublicDnsName":
"ec2-44-201-211-171.compute-1.amazonaws.com",
                                        "PublicIp": "44.201.211.171"
                                    },
                                    "Primary": true,
                                    "PrivateDnsName": "ip-172-31-87-93.ec2.internal",
```

```
                    "PrivateIpAddress": "172.31.87.93"
                }
            ],
            "SourceDestCheck": true,
            "Status": "in-use",
            "SubnetId": "subnet-0cbdaebd8c8e2fe91",
            "VpcId": "vpc-095498a76ffd2ba23",
            "InterfaceType": "interface"
        }
    ],
    "RootDeviceName": "/dev/xvda",
    "RootDeviceType": "ebs",
    "SecurityGroups": [
        {
            "GroupName": "launch-wizard-3",
            "GroupId": "sg-0cad4992ee8ec7a73"
        }
    ],
    "SourceDestCheck": true,
    "Tags": [
        {
            "Key": "Name",
            "Value": "Wiz_orbenson"
        }
    ],
    "VirtualizationType": "hvm",
    "CpuOptions": {
        "CoreCount": 1,
        "ThreadsPerCore": 1
    },
    "CapacityReservationSpecification": {
        "CapacityReservationPreference": "open"
    },
    "HibernationOptions": {
        "Configured": false
    },
    "MetadataOptions": {
        "State": "applied",
        "HttpTokens": "optional",
        "HttpPutResponseHopLimit": 1,
        "HttpEndpoint": "enabled",
        "HttpProtocolIpv6": "disabled",
        "InstanceMetadataTags": "disabled"
    },
    "EnclaveOptions": {
        "Enabled": false
    },
    "PlatformDetails": "Linux/UNIX",
    "UsageOperation": "RunInstances",
    "UsageOperationUpdateTime": "2022-12-21T08:31:40+00:00",
    "PrivateDnsNameOptions": {
        "HostnameType": "ip-name",
        "EnableResourceNameDnsARecord": true,
        "EnableResourceNameDnsAAAARecord": false
    },
```

```
                    "MaintenanceOptions": {
                        "AutoRecovery": "default"
                    }
                }
            ],
            "OwnerId": "465460162145",
            "ReservationId": "r-0863d45450cdc3bf9"
        }
    ]
}
```


The following command can be used in the AWS CLI to get the output:

```
aws ec2 describe-instances --region us-east-1 --instance-ids
i-057328b772102801d  --query
'Reservations[*].Instances[*].MetadataOptions.HttpTokens[]'
```

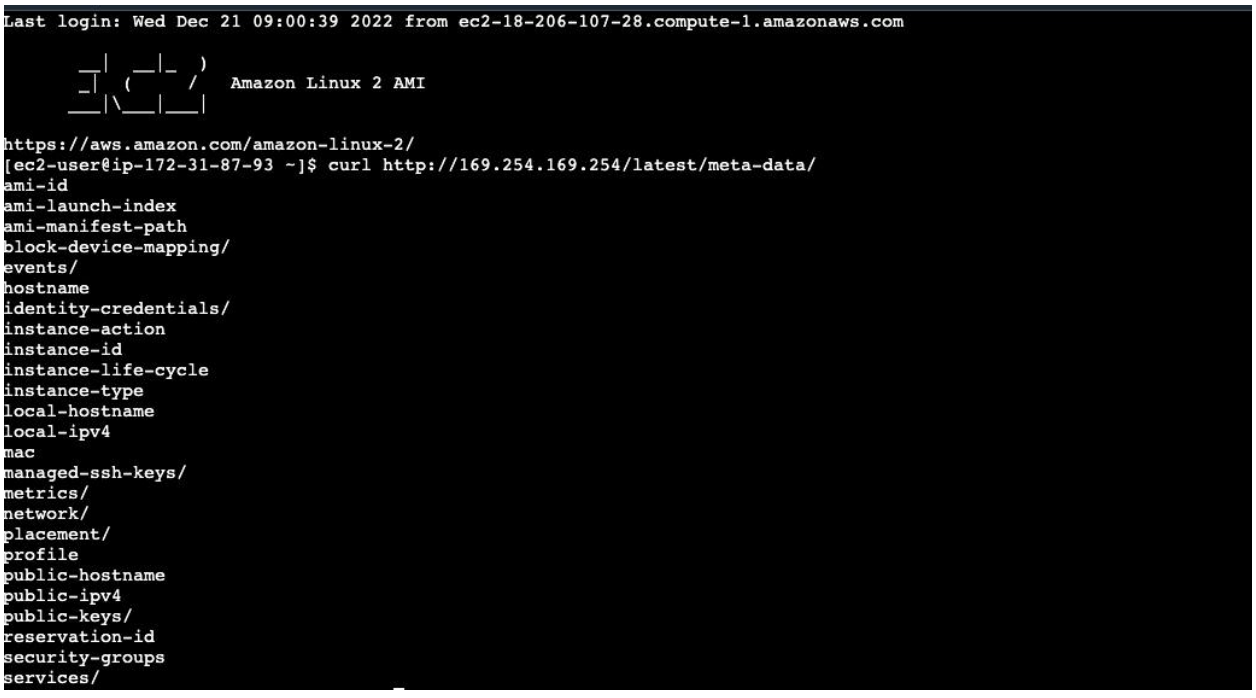
Output:


```
[
    "optional"
]
```


When connecting to the instance, if the following command is run,

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

you receive a list of items if your instance can use IMDSv1 requests. If you receive a “401” code then the instance uses IMDSv2 (or none). The IP addresses are link-local addresses and are valid only from the instance.



2.
To check if an EC2 instance is using IMDSv1, you can use the MetadataOptions field of the Instance object in the input JSON. The MetadataOptions field contains information about the metadata service configuration for instance. If the HttpTokens field of the MetadataOptions object is set to "optional," this means that the instance is using IMDSv1.

One way to see this is to use the JSON file, and from that, we can extract the following

```
Import json
def is_using_imdsv1(instance_json):
    instance = json.loads(instance_json)
    metadata_options =
instance['Reservations']['Instances'][0]['MetadataOptions']
    if metadata_options and metadata_options['HttpTokens'] ==
'optional':
        return True
    return False
```

To more accurately address the question, OPA Rego policy that can be used to validate the client use case:

```
package ec2

import input.Reservations

default match = false

# Check if the instance uses IMDSv1
match = Reservations.Instances.MetadataOptions.HttpTokens ==
"optional"
```

The input is the JSON we got from the previous question.

In this code, the Reservations field of the input JSON is imported as the input variable. The Instances field of the Reservations object contains information about the instances in the input JSON. The MetadataOptions field of the Instance object contains information about the metadata service configuration for instance. If the HttpTokens field of the MetadataOptions object is set to "optional, " the instance is using IMDSv1.

The matching variable is true if the instance uses IMDSv1 and false otherwise.

To use this policy, you can pass the JSON representation of the EC2 instance as the input to the policy. The policy will then return a boolean value indicating whether the instance uses IMDSv1.

For the following JSON, we have:

```
{
  "match": true
}
```

And when we try a different :

```
{
  "match": false
}
```

3.
A customer should run this command for their instance, with ID i-12345abcd1234abcd and region us-east-1

```
aws ec2 modify-instance-metadata-options --region us-east-1
--instance-id  i-12345abcd1234abcd --http-tokens required
--http-endpoint enabled
```

And they can also use HttpPutResponseHopLimit and EnforceIMDSv2 (set to required).

4.
**.bash_logout** - file containing commands to be executed when the user logs out from the shell
**.bash_profile** - file containing commands to be executed when the user logs into the shell
**.bashrc** - file with commands to be executed each time the user initiates a new shell session
**.ssh** - directory that contains the user's SSH keys and other SSH configuration files.

These files are typically hidden from view in the file system, as they are intended to be used by the operating system and not directly modified by the user. A user can view them with the command `ls -a`

5.
The Amazon Web Services (AWS) cloud infrastructure is vulnerable to attacks by threat actors like UNC2903, who have been known to target exploitable web applications running Amazon's Instance Metadata Service (IMDS) version 1. This service allows web requests to be made to a specialized URL using the link's local IP address (169.254.169.254) to facilitate internal communication and troubleshooting within the hosting platform. The metadata that can be retrieved through this service includes information about configurations, topologies, and even user roles and credentials. IMDS metadata is intended only to be accessible from within an EC2 instance, but if the instance is exposed to the internet, attackers may be able to steal credentials through vulnerabilities such as Server-Side Request Forgery (SSRF) or XML External Entity Injection. For example, an attacker could exploit a vulnerable web server running on the

instance to send a GET request redirected to the IMDS endpoint, potentially allowing the attacker to access IAM credentials. Additionally, even having shell access to the instance may expose access credentials to an attacker. It is important to secure EC2 instances and ensure they are not exposed to the internet to protect against these attacks.

By targeting vulnerabilities in web applications running IMDSv1, threat actors can potentially gain access to sensitive metadata stored in the service. This can be achieved through various techniques such as scanning for open ports and vulnerabilities, exploiting known vulnerabilities in the web applications or the underlying operating system, or using social engineering tactics to trick users into divulging their login credentials. Once the threat actor has gained access to the web applications and the IMDS service, they can potentially compromise other systems within the hosting platform. In the UNC2903 campaign, the threat actor was observed targeting vulnerabilities in web applications running IMDSv1 to gain access to the metadata stored in the service. It is unclear how the threat actor gained access to the web applications or what specific vulnerabilities they exploited however. To address this type of attack, Amazon released IMDS version 2, which includes additional protective measures and alerts through AWS security features like GuardDuty. IMDS version 2 mitigates attacks by requiring users to obtain a token through a PUT request to a specific URL, which must then be included in the header of all subsequent requests to IMDS. This added layer of protection helps to prevent unauthorized access to the metadata stored in the service and helps to secure the cloud infrastructure.

Another attack that utilized these vulnerabilities is the 2019 Capital One data breach in 2019 caused by a server-side request forgery (SSRF) vulnerability in a web application. The attacker exploited this vulnerability to access the AWS metadata service and obtain AWS access keys, which they used to access and download data from S3 buckets. It is believed that the web application was protected by a Web Application Firewall (WAF), but the attacker was able to bypass this protection or the WAF was not configured to block attacks. The SSRF vulnerability allowed the attacker to make a request to the AWS metadata endpoint, which is normally only accessible from within the instance itself. This endpoint includes information about the instance and its attached IAM role, including credentials for this role from the local address `http://169.254.169.254/latest/meta-data/iam/security-credentials/role-name`. In this case, the role had excessive privileges that allowed the attacker to list and access the S3 buckets. The attacker used these privileges to download the data and used the TOR network to hide their IP address.

**Question 2:**

1.

a. An SG with one inbound rule that allows ingress traffic for specific IPv4 IP for SMB protocol, and one outbound rule that allows All traffic for any IPv4 IP:


```
{
  "IpPermissions": [
    {
      "IpProtocol": "tcp",
      "FromPort": 445,
      "ToPort": 445,
      "IpRanges": [
        {
          "CidrIp": "10.0.0.1/32"
        }
      ]
    }
  ],
  "IpPermissionsEgress": [
    {
      "IpProtocol": "all",
      "IpRanges": [
        {
          "CidrIp": "0.0.0.0/0"
        }
      ]
    }
  ]
}
```


b. An SG with two inbound rules that allow ingress traffic for Any IPv4 in LDAP protocol and any IPv6 IP for IMAPS protocol (No outbound rules should be set).


```
{
  "IpPermissions": [
    {
      "IpProtocol": "tcp",
      "FromPort": 389,
      "ToPort": 389,
      "IpRanges": [
        {
          "CidrIp": "0.0.0.0/0"
        }
      ]
    },
    {
      "IpProtocol": "tcp",
      "FromPort": 993,
      "ToPort": 993,
      "Ipv6Ranges": [
        {
          "CidrIpv6": "::/0"
```

```
        }
      ]
    }
  ]
}
```

c. An SG with one outbound rule that allows all traffic in any port and IP; no inbound rules should be set.

```
{
  "IpPermissionsEgress": [
    {
      "IpProtocol": "all",
      "IpRanges": [
        {
          "CidrIp": "0.0.0.0/0"
        }
      ]
    }
  ]
}
```

2. We were asked to use OPA Rego policy to validate the client use case:

```
package sg


import input.IpPermissions
import input.IpPermissionsEgress


# A list of secure ports we got in the question
secure_ports = [22, 53, 135, 443, 445, 563, 993]


# Check if the SG allows any insecure ports in the inbound rules we got
allow_insecure_inbound = false


# Check if the SG allows any insecure ports in the outbound rules we got
allow_insecure_outbound = false


# Iterate through the inbound rules with a loop
```

```
for i in IpPermissions {
  # Check if the protocol is not ICMP and the port is lower than 1024
  if i.IpProtocol != "icmp" and i.FromPort < 1024 {
     # If the port is not in the list of secure ports, set allow_insecure_inbound to
true
    if i.FromPort not in secure_ports {
      allow_insecure_inbound = true
      break
    }
  }
}

# Iterate through the outbound rules
for i in IpPermissionsEgress {
  # Check if the protocol is not ICMP and the port is lower than 1024
  if i.IpProtocol != "icmp" and i.FromPort < 1024 {
     # If the port is not in the list of secure ports, set allow_insecure_outbound to
true
    if i.FromPort not in secure_ports {
      allow_insecure_outbound = true
      break
    }
  }
}

# If the SG allows any insecure ports in the inbound or outbound rules, set match to
true
match = allow_insecure_inbound or allow_insecure_outbound
```

This policy evaluates the inbound and outbound rules of an AWS Security Group (SG), iterating through the rules and checking if they allow any connections to insecure ports, which are defined as any port lower than 1024 that is not using the ICMP protocol or any port using the tcp protocol lower than 1024 except for the specified secure ports (22, 53, 135, 443, 445, 563, 993). The variable “match” evaluates to true if the SG allows any insecure ports, otherwise, false.

**References:**


1. https://www.darkreading.com/attacks-breaches/capital-one-attacker-exploited-misconfigured-aws-databases
2. https://krebsonsecurity.com/2019/08/what-we-can-learn-from-the-capital-one-hack/
3. https://www.trendmicro.com/cloudoneconformity/knowledge-base/aws/EC2/require-imds-v2.html
4. https://www.mandiant.com/resources/blog/cloud-metadata-abuse-unc2903
5. https://aws.amazon.com/premiumsupport/knowledge-center/ssm-ec2-enforce-imdsv2/
6. https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/configuring-instance-metadata-service.html
7. https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/instancedata-data-retrieval.html