

Wiz Technical Security Analyst – Interview Assignment

This PDF and **additional files** can be found here <https://github.com/danielnachumdev/wiz-cloud-security-analyst-asesment>

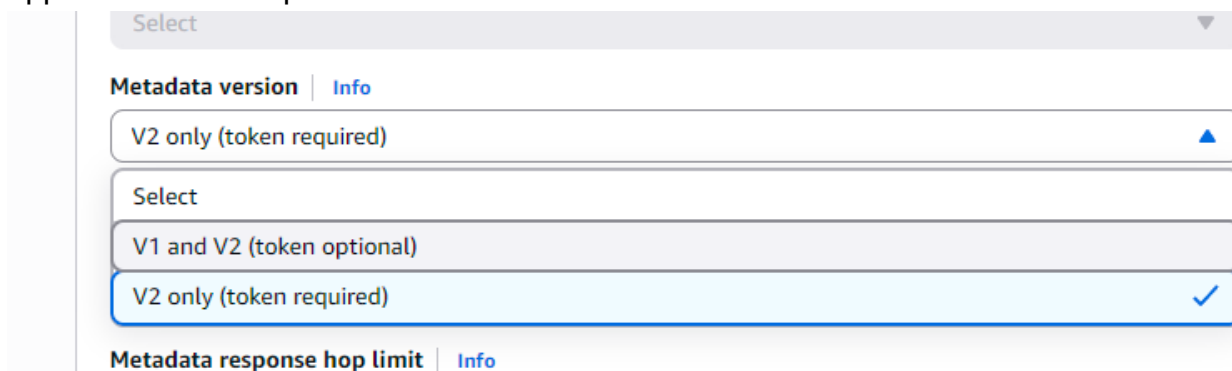
submitted by: danielnachumdev

Question 1: AWS (Amazon Web Services) EC2 Instance

1. Research the AWS API and provide an example JSON payload of a **specific** EC2 instance

I created my AWS account.

created an EC2 instance (ID: `i-0c6752430e4834e26`) through the AWS console. It was important to select the `V1` and `V2` (token optional) option to ensure the proper values appeared in the output JSON.



Afterwards, I created an IAM user named `nikkiw` and a permission group. I added `nikkiw` to the group and assigned it administrator permissions.

The IAM account allowed me to access resources using the AWS CLI. I set up a profile with `aws configure sso` , using the values for my IAM role.

I then ran:

```
aws ec2 describe-instances --instance-ids i-0c6752430e4834e26 --query
"Reservations[0].Instances[0].MetadataOptions" --profile AdministratorAccess-
975707452464
```

and got as a result

```
{
  "State": "applied",
  "HttpTokens": "optional",
  "HttpPutResponseHopLimit": 2,
  "HttpEndpoint": "enabled",
  "HttpProtocolIpv6": "disabled",
  "InstanceMetadataTags": "disabled"
}
```

As we can see we have a key called `HttpTokens` which has a value of `"optional"`. This is because we have selected the first option as shown in the picture above. If we would have selected the second option the value would be `"required"` instead.

2. Write an OPA (Open Policy Agent)

Below is an OPA Rego policy to evaluate if the instance metadata service is set to "optional":

```
# Namespace for this policy to avoid name collisions with other
policies/rules.
package ec2policy

# By default, the 'match' key will be false unless another rule sets it to
true.
default match = false

# This rule sets 'match' to true if the condition inside the block is
satisfied:
# Specifically, if the 'HttpTokens' field in the input JSON has the value
"optional".
match {
  input.HttpTokens == "optional"
}
```

I have also tested it on <https://play.openpolicyagent.org/> but the gist (in my own words rather than an LLM's words which wrote the comments) is that:

- We set a name prevent naming conflicts (but here we have just one policy)
- We set a default return value which will be returned in any case where we don't have an explicit rule to set to true (which in this case also handles the case where the `HttpTokens` field is missing)

- Lastly, we set the `match` result to be `true` iff in the input that we give the evaluator we have a field called `HttpTokens` and it has a value of `"optional"`

This above returned JSON will return for us

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

as can be seen in the test I performed

The screenshot shows the Rego Playground interface. On the left, a code editor contains the following Rego policy:

```
1 package ec2policy  
2  
3 default match = false  
4  
5 match {  
6   input.HttpTokens == "optional"  
7 }  
8
```

On the right, there are three panels: INPUT, DATA, and OUTPUT.

- INPUT:** A JSON object with the following fields:

```
{  
  "State": "applied",  
  "HttpTokens": "optional",  
  "HttpPutResponseHopLimit": 2,  
  "HttpEndpoint": "enabled",  
  "HttpProtocolIpv6": "disabled",  
  "InstanceMetadataTags": "disabled"  
}
```
- DATA:** This panel is currently empty.
- OUTPUT:** It shows the result of the evaluation:

```
Found 1 result in 35µs.  
1 {  
2   "match": true  
3 }
```

Sidenote: I am familiar with `regex` and use `regex101.com` regularly to develop them so I searched for a similar website to test `Rego` and found the same website you link in the PDF before I saw the link.

3. What are the AWS CLI command for editing an existing EC2 instance to stop using IMDSv1 and use instead IMDSv2?

The command is

```
aws ec2 modify-instance-metadata-options --instance-id i-0c6752430e4834e26 --
```

```
http-tokens required --profile AdministratorAccess-975707452464
```

which returned

```
{
  "InstanceId": "i-0c6752430e4834e26",
  "InstanceMetadataOptions": {
    "State": "pending",
    "HttpTokens": "required",
    "HttpPutResponseHopLimit": 2,
    "HttpEndpoint": "enabled",
    "HttpProtocolIpv6": "disabled",
    "InstanceMetadataTags": "disabled"
  }
}
```

and as we can see the value is indeed changed.

I have double verified with the command from step 1 and got

```
{
  "State": "applied",
  "HttpTokens": "required",
  "HttpPutResponseHopLimit": 2,
  "HttpEndpoint": "enabled",
  "HttpProtocolIpv6": "disabled",
  "InstanceMetadataTags": "disabled"
}
```

4. Please connect to the EC2 instance - what files exist in the user directory by default?

I have connected to the instance using the web UI as it is easier and I haven't created a keypair during setup to download a PEM file and use SSH.

then I did

```
ls -ah
```

```
,      #_
~\    ####_   Amazon Linux 2023
~~ \  #####\_ 
~~  _#####|  
~~     \|###|   
~~       \|#/   https://aws.amazon.com/linux/amazon-linux-2023
~~~~        V~' '->
          ~~~
           ._. /
            |/_/_/
             /m/' -/
```

Last login: Thu Jul 17 10:10:54 2025 from 13.48.4.202
[ec2-user@ip-172-31-37-154 ~]\$ ls -ah
. .. .bash_history .bash_logout .bash_profile .bashrc .ssh
[ec2-user@ip-172-31-37-154 ~]\$ █

attacker could trick the instance into making an internal request, they could harvest credentials and escalate privileges. The ACM case study highlights that the trust-by-default model in IMDSv1 allowed such attacks if application or infrastructure misconfigurations were present.

Attack Stages:

1. **Reconnaissance:** The attacker used anonymizing services to scan and access Capital One's cloud infrastructure undetected.
2. **Exploit Misconfiguration:** A misconfigured ModSecurity Web Application Firewall (WAF) operating as a reverse proxy allowed internal requests to the AWS metadata service.
3. **Harvest Credentials:** Via SSRF or reverse proxy, the attacker accessed IMDSv1, retrieving temporary IAM role credentials with broad permissions.
4. **Privilege Abuse:** These credentials were used to list and extract sensitive S3 data, exploiting excessive privileges and inadequate encryption controls.
5. **Data Exfiltration:** Over 30 GB of customer data was downloaded without detection due to insufficient monitoring and failed intrusion detection.

Why IMDSv2 Is Needed:

IMDSv2 was introduced to mitigate the risks inherent in IMDSv1. IMDSv2 requires session-based authentication and use of a PUT request with a token for metadata retrieval, which makes it resilient against classic SSRF and reverse proxy attacks detailed in the Capital One breach. The ACM paper makes clear that the "openness" of IMDSv1 was a core vulnerability.

Mitigations and Controls:

- **Enable and enforce IMDSv2:** Organizations should deprecate IMDSv1 and require IMDSv2, which is resistant to unauthenticated requests.
- **Least Privilege IAM:** Limit the permissions assigned to roles to only what is strictly necessary to minimize data exposure if credentials are compromised.
- **WAF and Infrastructure Hardening:** Properly configure WAFs, using current vulnerability management and periodic review, and avoid unnecessary complexity in proxy configurations.
- **Intrusion Detection and Monitoring:** Implement detailed monitoring of IAM API calls and cloud resource access, and alert on anomalous behavior.
- **Organizational Controls:** Regular security assessments, better alignment between IT and security teams, robust training, and board-level engagement in cybersecurity risk management.

Question 2: Insecure Security Group Ports Used

1. Research the AWS API, and provide an example JSON payload for each of the above security

I have created a security group for this purpose named `wiz-sg1`

```
aws ec2 describe-security-groups --group-ids sg-005c4166bbcc29433 --profile AdministratorAccess-975707452464
```

which returns

```
{
  "SecurityGroups": [
    {
      "GroupId": "sg-005c4166bbcc29433",
      "IpPermissionsEgress": [
        {
          "IpProtocol": "-1",
          "UserIdGroupPairs": [],
          "IpRanges": [
            {
              "CidrIp": "0.0.0.0/0"
            }
          ],
          "Ipv6Ranges": [],
          "PrefixListIds": []
        }
      ],
      "VpcId": "REDACTED",
      "SecurityGroupArn": "REDACTED",
      "OwnerId": "REDACTED",
      "GroupName": "wiz-sg1",
      "Description": "sg for wiz p2q1",
      "IpPermissions": [
        {
          "IpProtocol": "tcp",
          "FromPort": 445,
          "ToPort": 445,
          "UserIdGroupPairs": [],
          "IpRanges": [
            {
              "CidrIp": "87.70.13.254/32"
            }
          ],
          "Ipv6Ranges": [],

```

```

        "PrefixListIds": []
    }
}
]
}
]
}

```

and as can be seen there is one **outbound** rule evident by one entry in the array of `IpPermissionsEgress` , and similarly one **inbound** rule evident by one entry in the array of `IpPermissions` .

Note that port `445` is the port for `SMB` as requested..

Similarly I created another one called `wiz-sg2` which returned

```

{
  "SecurityGroups": [
    {
      "GroupId": "sg-06c63c2bffb60f5e8",
      "IpPermissionsEgress": [],
      "VpcId": "REDACTED",
      "SecurityGroupArn": "REDACTED",
      "OwnerId": "REDACTED",
      "GroupName": "wiz-sg2",
      "Description": "sg for wiz p2q1p2",
      "IpPermissions": [
        {
          "IpProtocol": "tcp",
          "FromPort": 993,
          "ToPort": 993,
          "UserIdGroupPairs": [],
          "IpRanges": [],
          "Ipv6Ranges": [
            {
              "CidrIpv6": ":::/0"
            }
          ],
          "PrefixListIds": []
        },
        {
          "IpProtocol": "tcp",
          "FromPort": 389,
          "ToPort": 389,
          "UserIdGroupPairs": [],
          "IpRanges": [

```



```

        {
            "CidrIp": "0.0.0.0/0"
        }
    ],
    "Ipv6Ranges": [],
    "PrefixListIds": []
}
]
}
]
}

```

LDAP port is 389 and IMAPS port is 993

Lastly, I created `wiz-sg3` which returned

```

{
  "SecurityGroups": [
    {
      "GroupId": "sg-04b5a87438c7e2bb1",
      "IpPermissionsEgress": [
        {
          "IpProtocol": "-1",
          "UserIdGroupPairs": [],
          "IpRanges": [
            {
              "CidrIp": "0.0.0.0/0"
            }
          ],
          "Ipv6Ranges": [
            {
              "CidrIpv6": ":::/0"
            }
          ],
          "PrefixListIds": []
        }
      ],
      "VpcId": "REDACTED",
      "SecurityGroupArn": "REDACTED",
      "OwnerId": "REDACTED",
      "GroupName": "wiz-sg3",
      "Description": "sg for wiz p2q1p3",
      "IpPermissions": []
    }
  ]
}

```

```
]
}
```

2. Write an OPA (Open Policy Agent) Rego code whose input is all the examples above (a), that validates the customer's use case. Output must contain a "match" Boolean key

In this section the phrasing was a little ambiguous (caught the nuance late which expensed a lot of work time)

The confusing part is about "...all the examples above (a),..."

which could refer to the previous section and it's three subsections 1.i, 1.ii, 1.iii OR it could refer to the example given in triple quotes at the start of the question.

Initially I understood it as the first option which is considerably more complicated. but upon re-checking my solution against the instruction prior to submission I saw that section 2.1 refers to "...allows the customer insecure ports,...", and specifically, only the ports are mentions rather than the protocols, IPv4/IPv6 inbound/outbound which would be more resembling to the second interpretations. Additionally the examples in 2.iii support the second interpretation.

Given that, here is the attached solution for the second interpretation which is what I think was the intended result.

(Even so, the solution for the first implementation can be found on my [github](#) at `scripts/script2_extra.rego` or by clicking [here](#))

For both interpretations of the question I have created over 60(!) tests using PYTHON + JSON. I have tested by getting help from AI to create a test script in python (as I did a quick search to see if there are debuggers or something that I can use - but I found this to be insufficient)

[script2.rego](#)

```
package scripts

import rego.v1

default match := false

safe_tcp_ports := [22, 53, 135, 443, 445, 563, 993]

# check if protocol is icmp OR if protocol tcp and safe port
```

```

is_valid(rule) if {
    rule.IpProtocol == "icmp"
    rule.FromPort == rule.ToPort # assuming that this is intended}
is_valid(rule) if {
    rule.IpProtocol == "tcp"
    rule.FromPort == rule.ToPort # assuming that this is intended
rule.FromPort in safe_tcp_ports
}

# check if rule is valid and port is lower than 1024
is_rule_safe(rule) if {
    rule.FromPort < 1024
    rule.ToPort < 1024
    is_valid(rule)
}

# if even one rule is unsafe the result is False
is_inbound_rules_unsafe(sg) if {
    some rule in sg.IpPermissions
    not is_rule_safe(rule)
}

# if even one SG has False reutrn True
match if {
    some sg in input.SecurityGroups
    is_inbound_rules_unsafe(sg)
}

```