Infrastructure as Code (IaC) Implementation

What this module will do:

- Load Terraform files (.tf) before they are deployed.
- Detect simple misconfigurations like:

Action = "*" or **Resource** = "*" in IAM policy blocks.

Public S3 buckets (acl = "public-read" or public = true).

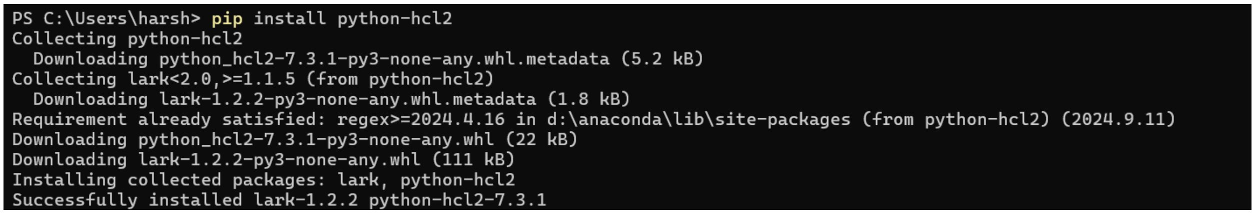
Security group with **wide-open ingress** (0.0.0.0/0).

- Save findings to a CSV/print them on screen.

Steps to reproduce implementation:

Step 1: Install a parser library for HCL (Terraform Language). HCL is a language used to describe infrastructure resources in machine friendly and human-readable format.

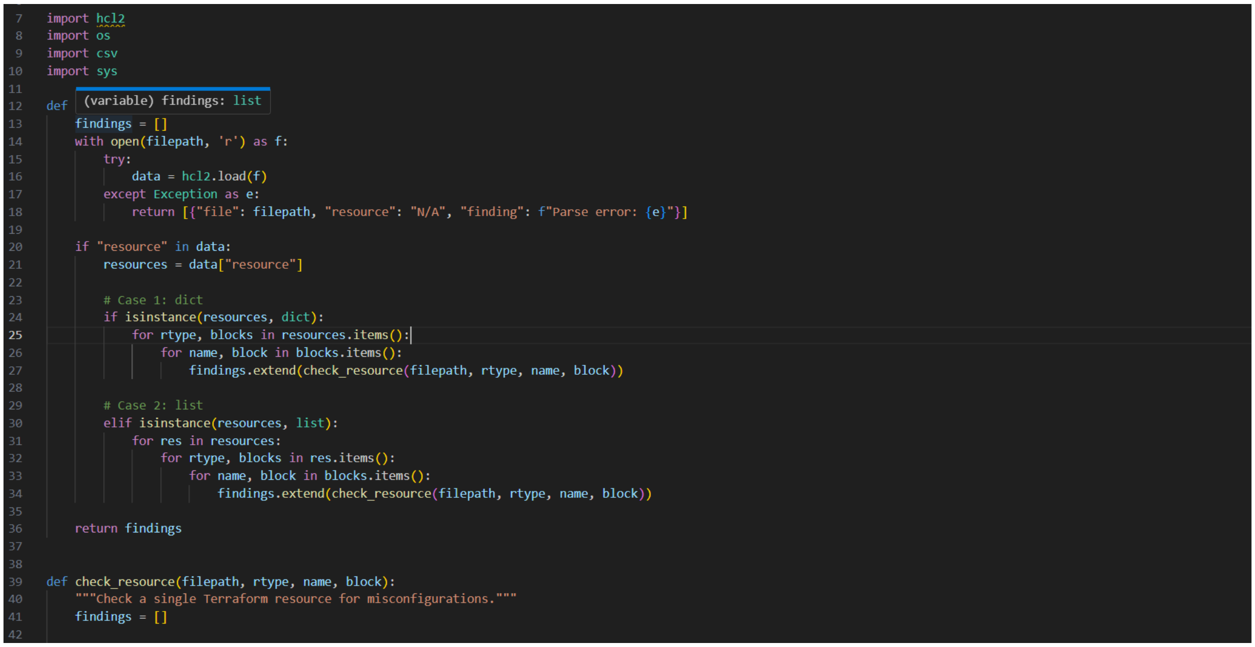
Terraform is an open-source IaC tool that uses HCL to define, provision and manage infrastructure resources such as cloud infrastructure in AWS using configuration files. (In short, a CLI for defining resources in cloud instead of clicking buttons).



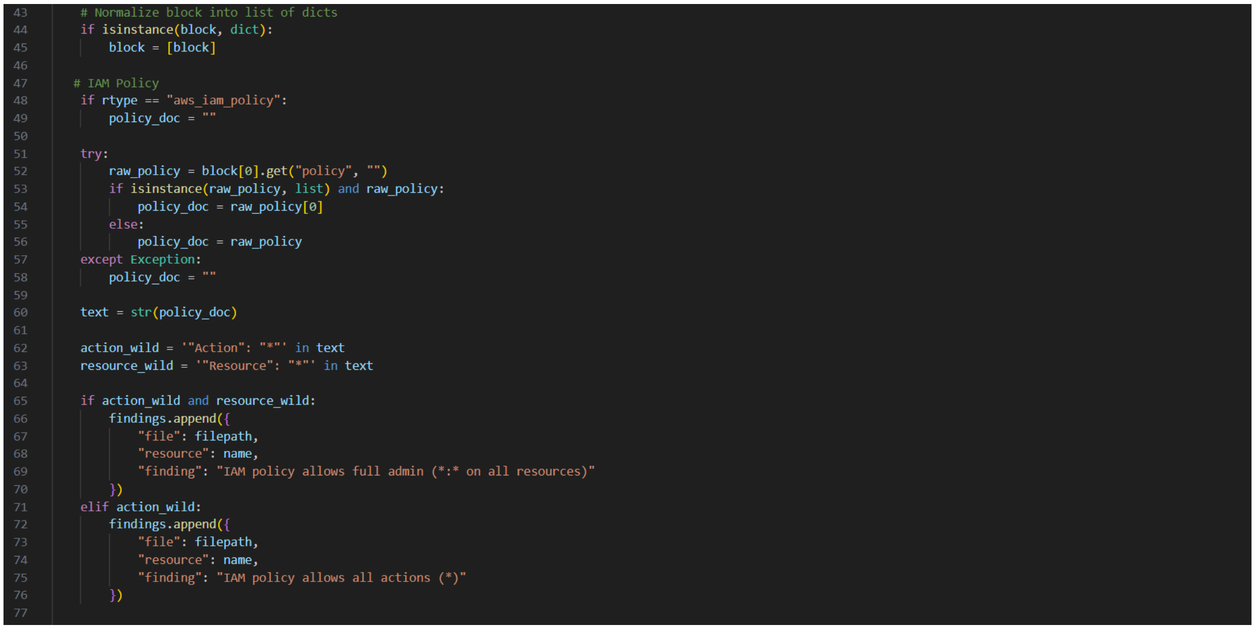
```
PS C:\Users\harsh> pip install python-hcl2
Collecting python-hcl2
  Downloading python_hcl2-7.3.1-py3-none-any.whl.metadata (5.2 kB)
Collecting lark<2.0,>=1.1.5 (from python-hcl2)
  Downloading lark-1.2.2-py3-none-any.whl.metadata (1.8 kB)
Requirement already satisfied: regex>=2024.4.16 in d:\anaconda\lib\site-packages (from python-hcl2) (2024.9.11)
Downloading python_hcl2-7.3.1-py3-none-any.whl (22 kB)
Downloading lark-1.2.2-py3-none-any.whl (111 kB)
Installing collected packages: lark, python-hcl2
Successfully installed lark-1.2.2 python-hcl2-7.3.1
```

Step 2: Write the python code for the scanner which detects and prevents insecure policies **before** deployment.

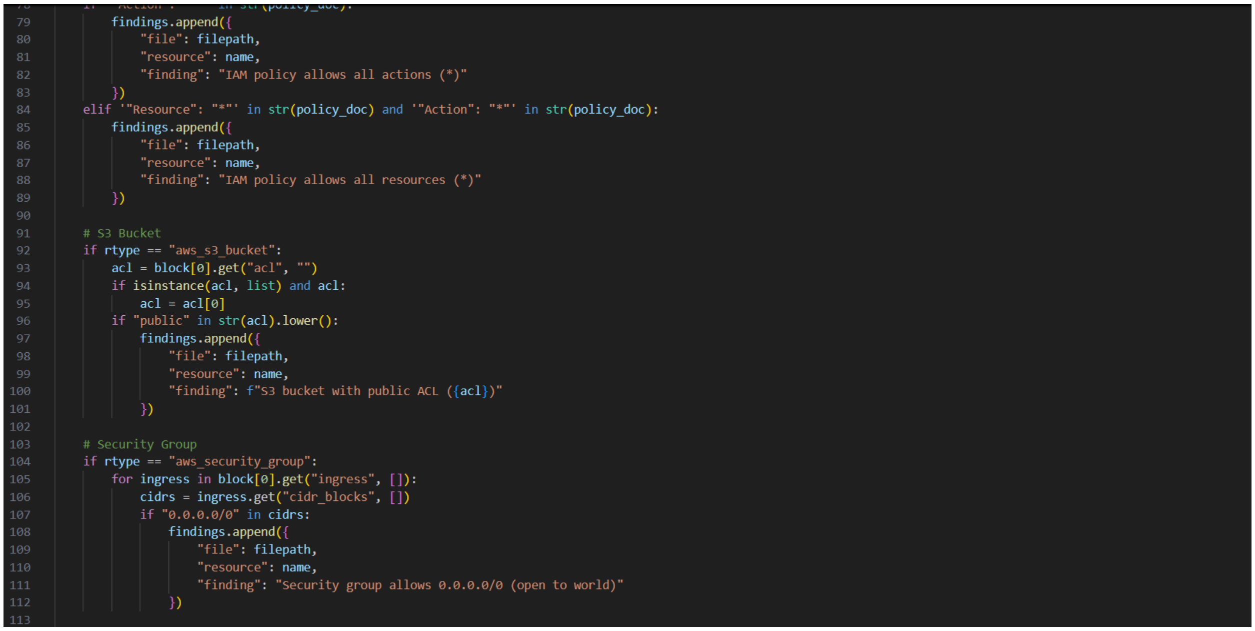
Lets name the file iac_monitor.py:



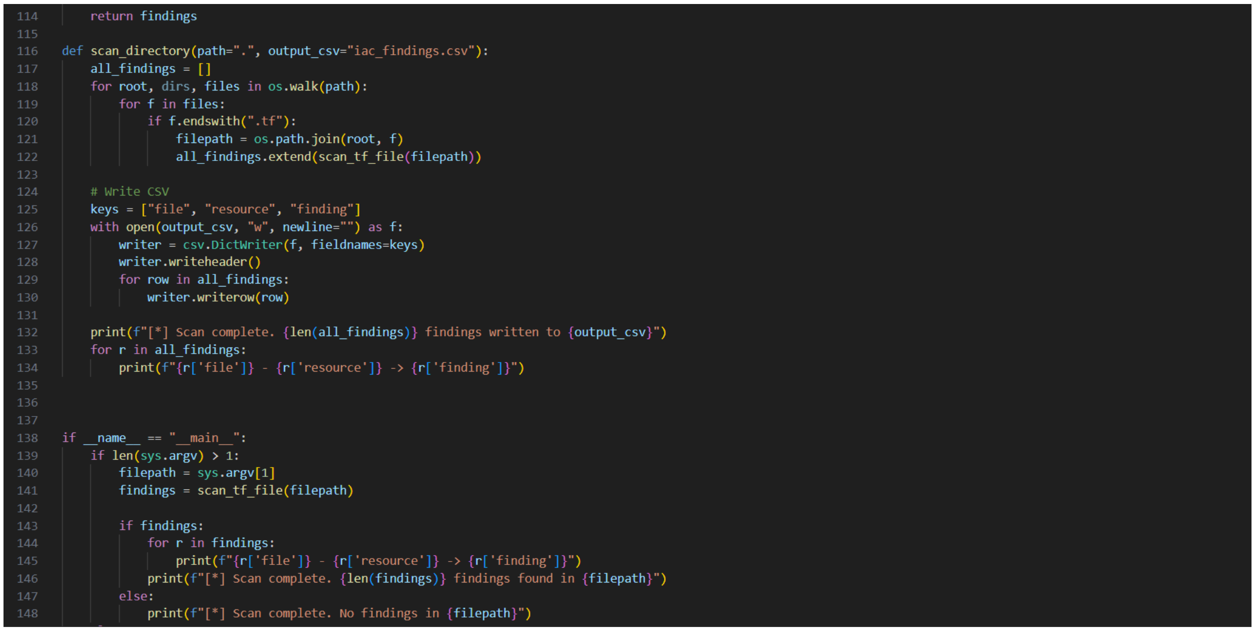
```python
import hcl2
import os
import csv
import sys

def  (variable) findings: list
    findings = []
    with open(filepath, 'r') as f:
        try:
            data = hcl2.load(f)
        except Exception as e:
            return [{"file": filepath, "resource": "N/A", "finding": f"Parse error: {e}"}]

    if "resource" in data:
        resources = data["resource"]

        # Case 1: dict
        if isinstance(resources, dict):
            for rtype, blocks in resources.items():
                for name, block in blocks.items():
                    findings.extend(check_resource(filepath, rtype, name, block))

        # Case 2: list
        elif isinstance(resources, list):
            for res in resources:
                for rtype, blocks in res.items():
                    for name, block in blocks.items():
                        findings.extend(check_resource(filepath, rtype, name, block))

    return findings


def check_resource(filepath, rtype, name, block):
    """Check a single Terraform resource for misconfigurations."""
    findings = []

```



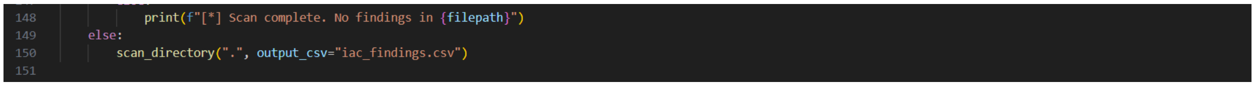
```python
    # Normalize block into list of dicts
    if isinstance(block, dict):
        block = [block]

   # IAM Policy
    if rtype == "aws_iam_policy":
        policy_doc = ""

    try:
        raw_policy = block[0].get("policy", "")
        if isinstance(raw_policy, list) and raw_policy:
            policy_doc = raw_policy[0]
        else:
            policy_doc = raw_policy
    except Exception:
        policy_doc = ""

    text = str(policy_doc)

    action_wild = '"Action": "*"' in text
    resource_wild = '"Resource": "*"' in text

    if action_wild and resource_wild:
        findings.append({
            "file": filepath,
            "resource": name,
            "finding": "IAM policy allows full admin (*:* on all resources)"
        })
    elif action_wild:
        findings.append({
            "file": filepath,
            "resource": name,
            "finding": "IAM policy allows all actions (*)"
        })

```



```python
        if '"Action": "*"' in str(policy_doc):
            findings.append({
                "file": filepath,
                "resource": name,
                "finding": "IAM policy allows all actions (*)"
            })
        elif '"Resource": "*"' in str(policy_doc) and '"Action": "*"' in str(policy_doc):
            findings.append({
                "file": filepath,
                "resource": name,
                "finding": "IAM policy allows all resources (*)"
            })

        # S3 Bucket
        if rtype == "aws_s3_bucket":
            acl = block[0].get("acl", "")
            if isinstance(acl, list) and acl:
                acl = acl[0]
            if "public" in str(acl).lower():
                findings.append({
                    "file": filepath,
                    "resource": name,
                    "finding": f"S3 bucket with public ACL ({acl})"
                })

        # Security Group
        if rtype == "aws_security_group":
            for ingress in block[0].get("ingress", []):
                cidrs = ingress.get("cidr_blocks", [])
                if "0.0.0.0/0" in cidrs:
                    findings.append({
                        "file": filepath,
                        "resource": name,
                        "finding": "Security group allows 0.0.0.0/0 (open to world)"
                    })

```



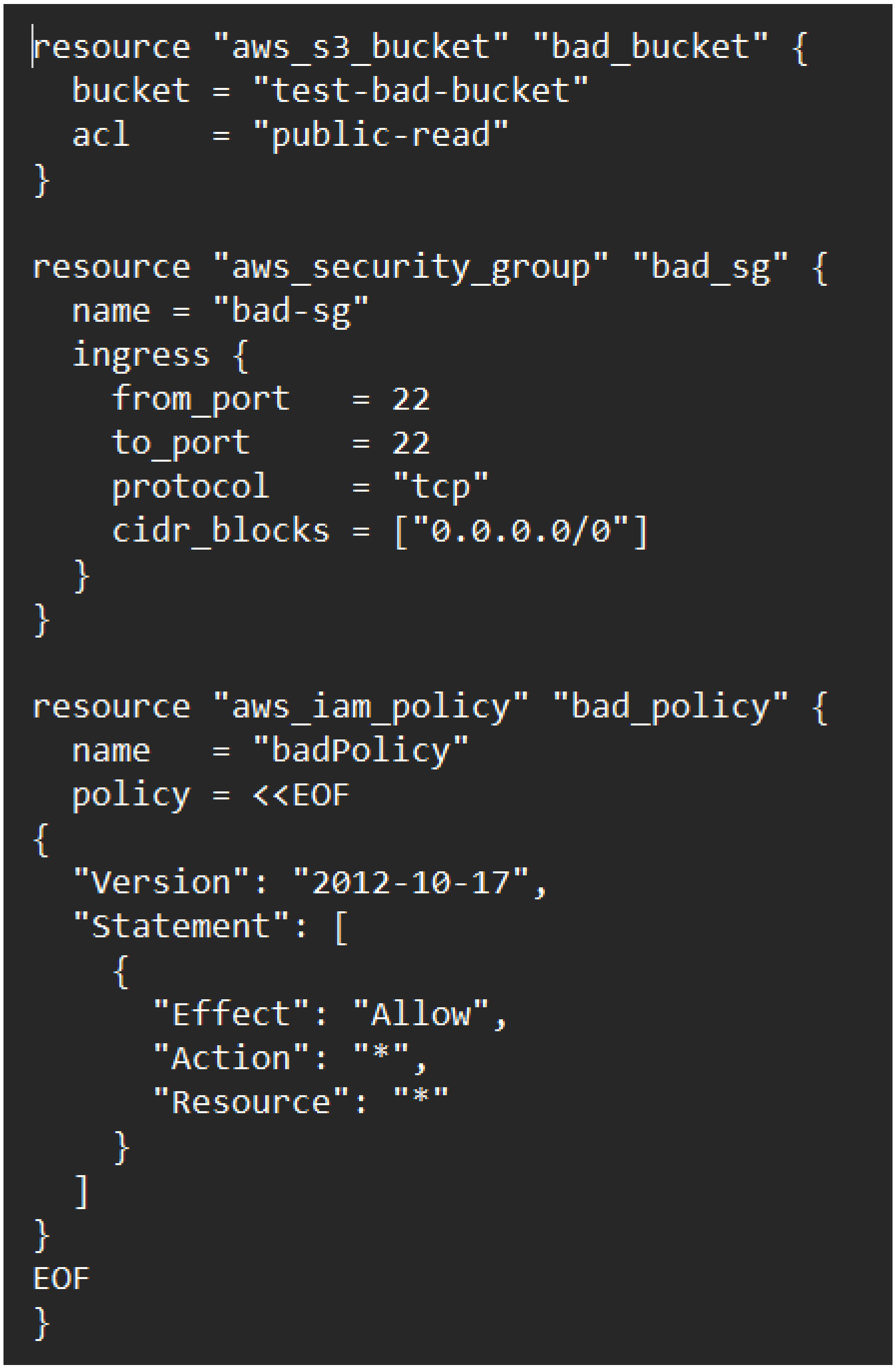
```python
    return findings

def scan_directory(path=".", output_csv="iac_findings.csv"):
    all_findings = []
    for root, dirs, files in os.walk(path):
        for f in files:
            if f.endswith(".tf"):
                filepath = os.path.join(root, f)
                all_findings.extend(scan_tf_file(filepath))

    # Write CSV
    keys = ["file", "resource", "finding"]
    with open(output_csv, "w", newline="") as f:
        writer = csv.DictWriter(f, fieldnames=keys)
        writer.writeheader()
        for row in all_findings:
            writer.writerow(row)

    print(f"[*] Scan complete. {len(all_findings)} findings written to {output_csv}")
    for r in all_findings:
        print(f"{r['file']} - {r['resource']} -> {r['finding']}")



if __name__ == "__main__":
    if len(sys.argv) > 1:
        filepath = sys.argv[1]
        findings = scan_tf_file(filepath)

        if findings:
            for r in findings:
                print(f"{r['file']} - {r['resource']} -> {r['finding']}")
            print(f"[*] Scan complete. {len(findings)} findings found in {filepath}")
        else:
            print(f"[*] Scan complete. No findings in {filepath}")
```



```python
            print(f"[*] Scan complete. No findings in {filepath}")
    else:
        scan_directory(".", output_csv="iac_findings.csv")

```

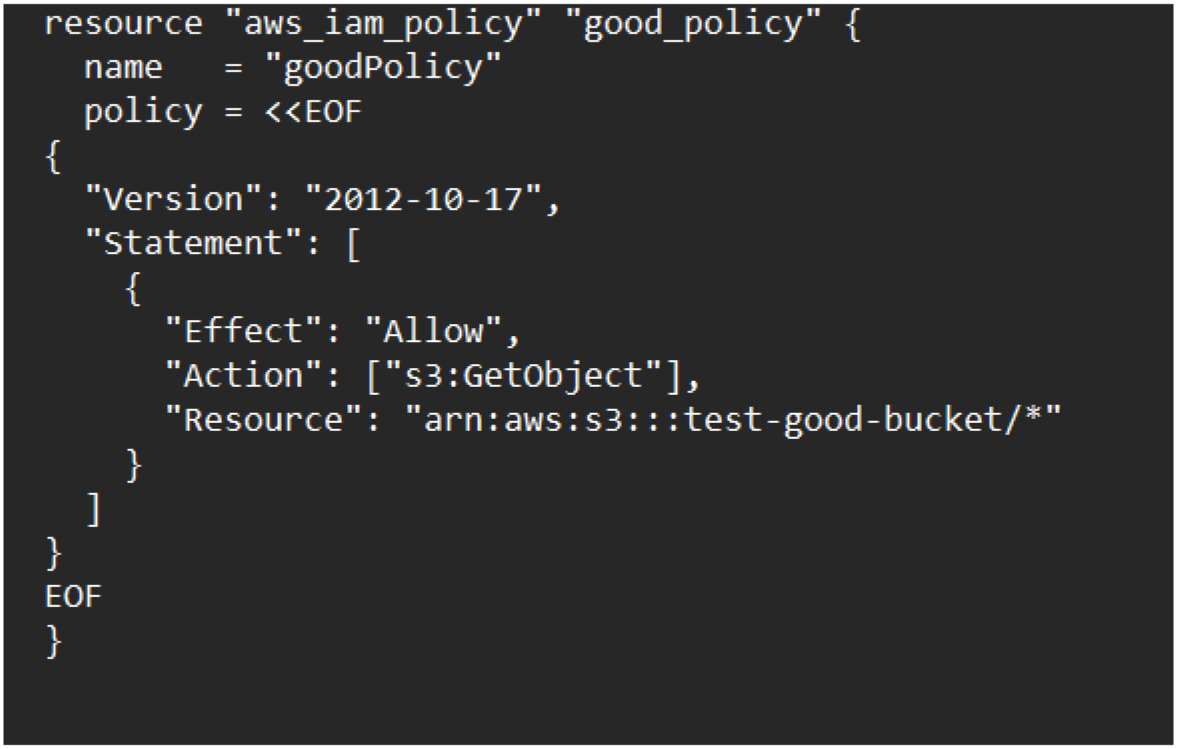
Step 3: We will create three Terraform files to check the scanner's efficiency and ability to identify misconfigurations.

bad.tf:



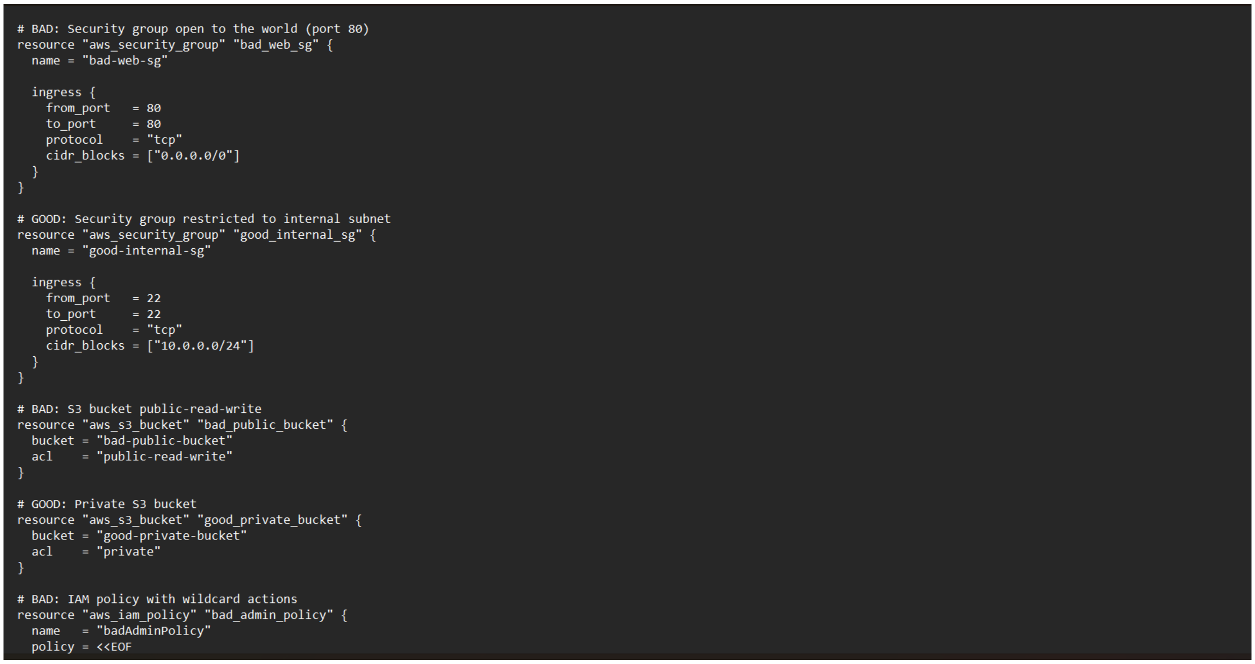
```
resource "aws_s3_bucket" "bad_bucket" {
  bucket = "test-bad-bucket"
  acl    = "public-read"
}

resource "aws_security_group" "bad_sg" {
  name = "bad-sg"
  ingress {
    from_port   = 22
    to_port     = 22
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }
}

resource "aws_iam_policy" "bad_policy" {
  name   = "badPolicy"
  policy = <<EOF
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "*",
      "Resource": "*"
    }
  ]
}
EOF
}
```

good.tf

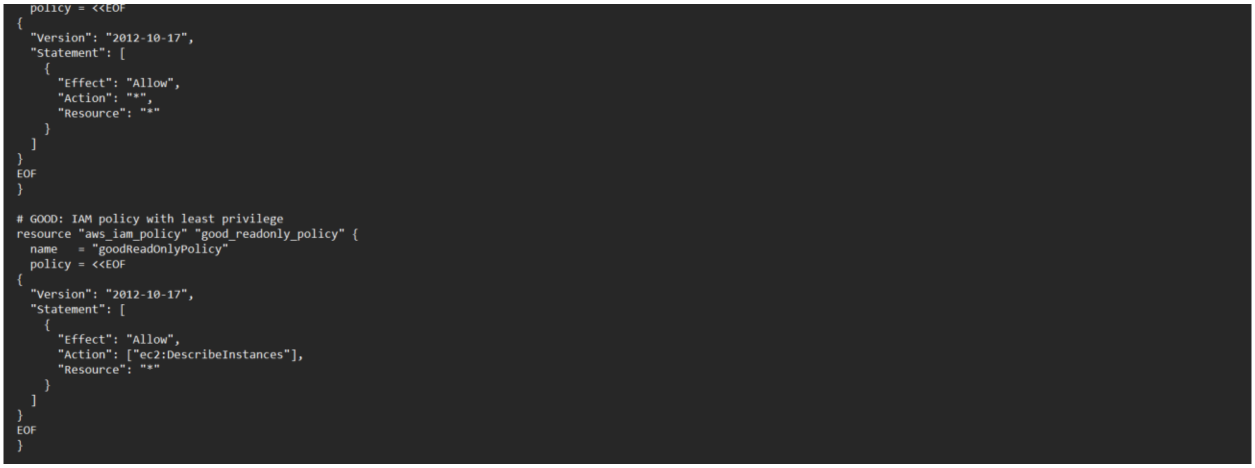


```
resource "aws_iam_policy" "good_policy" {
  name   = "goodPolicy"
  policy = <<EOF
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": ["s3:GetObject"],
      "Resource": "arn:aws:s3:::test-good-bucket/*"
    }
  ]
}
EOF
}
```

mix.tf

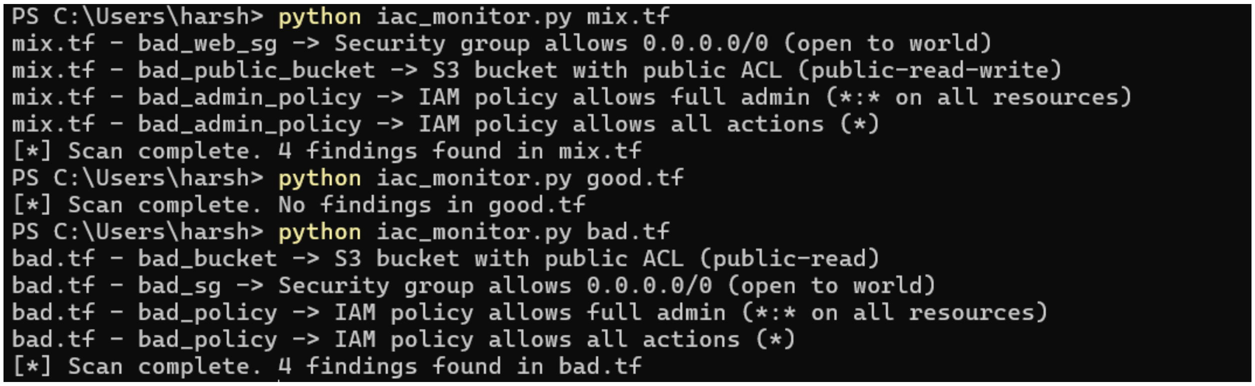


```
# BAD: Security group open to the world (port 80)
resource "aws_security_group" "bad_web_sg" {
  name = "bad-web-sg"

  ingress {
    from_port   = 80
    to_port     = 80
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }
}

# GOOD: Security group restricted to internal subnet
resource "aws_security_group" "good_internal_sg" {
  name = "good-internal-sg"

  ingress {
    from_port   = 22
    to_port     = 22
    protocol    = "tcp"
    cidr_blocks = ["10.0.0.0/24"]
  }
}

# BAD: S3 bucket public-read-write
resource "aws_s3_bucket" "bad_public_bucket" {
  bucket = "bad-public-bucket"
  acl    = "public-read-write"
}

# GOOD: Private S3 bucket
resource "aws_s3_bucket" "good_private_bucket" {
  bucket = "good-private-bucket"
  acl    = "private"
}

# BAD: IAM policy with wildcard actions
resource "aws_iam_policy" "bad_admin_policy" {
  name   = "badAdminPolicy"
  policy = <<EOF
```



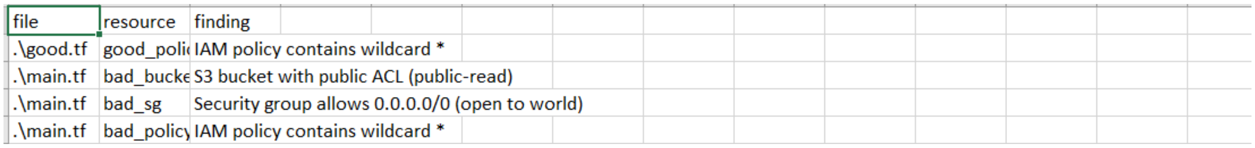
```
  policy = <<EOF
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "*",
      "Resource": "*"
    }
  ]
}
EOF
}

# GOOD: IAM policy with least privilege
resource "aws_iam_policy" "good_readonly_policy" {
  name   = "goodReadOnlyPolicy"
  policy = <<EOF
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": ["ec2:DescribeInstances"],
      "Resource": "*"
    }
  ]
}
EOF
}
```

Step 4: Run the scanner against these .tf files and we can see the findings and vulnerabilities detected in the infrastructure of the resource.



```
PS C:\Users\harsh> python iac_monitor.py mix.tf
mix.tf - bad_web_sg -> Security group allows 0.0.0.0/0 (open to world)
mix.tf - bad_public_bucket -> S3 bucket with public ACL (public-read-write)
mix.tf - bad_admin_policy -> IAM policy allows full admin (*:* on all resources)
mix.tf - bad_admin_policy -> IAM policy allows all actions (*)
[*] Scan complete. 4 findings found in mix.tf
PS C:\Users\harsh> python iac_monitor.py good.tf
[*] Scan complete. No findings in good.tf
PS C:\Users\harsh> python iac_monitor.py bad.tf
bad.tf - bad_bucket -> S3 bucket with public ACL (public-read)
bad.tf - bad_sg -> Security group allows 0.0.0.0/0 (open to world)
bad.tf - bad_policy -> IAM policy allows full admin (*:* on all resources)
bad.tf - bad_policy -> IAM policy allows all actions (*)
[*] Scan complete. 4 findings found in bad.tf
```

Step 5: The findings are documented in the iac_findings.csv file which can be used to analyze and visualize (using Grafana).



file	resource	finding
.\good.tf	good_poli	IAM policy contains wildcard *
.\main.tf	bad_bucke	S3 bucket with public ACL (public-read)
.\main.tf	bad_sg	Security group allows 0.0.0.0/0 (open to world)
.\main.tf	bad_policy	IAM policy contains wildcard *

Future Scope:

1. **Multi-Cloud Expansion**: (Azure, GCP etc).

2. **Severity-Based Risk Scoring**: High, Medium, Low.

3. **Continuous Monitoring**: Deploy as a Lambda function to automatically run on a schedule.

4. **Visualization & Reporting**: Dashboards using AWS QuickSight, Grafana.