Terraform AWS STS Lab - Student Guide

Overview

This lab demonstrates advanced AWS security concepts using **AWS Security Token Service (STS)** with Terraform. You'll learn how to create IAM roles, assume roles for temporary credentials, and manage AWS resources securely using role-based access control.

Learning Objectives

- Install and configure AWS CLI and Terraform CLI
- Understand AWS STS and role assumption concepts
- Create IAM users and roles using CloudFormation
- Configure Terraform to assume IAM roles
- Deploy AWS resources using assumed role credentials
- Practice secure credential management and cleanup procedures

---

Prerequisites

- Basic understanding of AWS IAM (users, roles, policies)
- Command line experience
- Text editor for configuration files
- AWS account with administrative privileges

Section 1: Environment Setup

1.1 Install AWS CLI

Download and Install AWS CLI v2

```bash
```

```
# Download AWS CLI v2
curl "https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip" -o "awscliv2.zip"

# Unzip the installer
unzip awscliv2.zip

# Install AWS CLI
sudo ./aws/install

# Verify installation
aws --version
```

Expected Output:

```
aws-cli/2.28.7 Python/3.13.4 Linux/6.14.0-27-generic exe/x86_64.ubuntu.24
```

Verify AWS Configuration Directory

```bash
ll ~/.aws/credentials
```

1.2 Install Terraform CLI

Add HashiCorp Repository

```bash
# Update package list and install dependencies
sudo apt-get update && sudo apt-get install -y gnupg software-properties-common curl

# Add HashiCorp GPG key (recommended method)
curl -fsSL https://apt.releases.hashicorp.com/gpg | sudo gpg --dearmor -o /usr/share/keyrings/hashicorp-archive-keyri

# Add HashiCorp repository
echo "deb [signed-by=/usr/share/keyrings/hashicorp-archive-keyring.gpg] https://apt.releases.hashicorp.com $(lsb_re
```

Install Terraform

```bash
```

```
# Update package list
sudo apt-get update

# Install Terraform
sudo apt-get install terraform

# Verify installation
terraform --version
```

Expected Output:

```
Terraform v1.13.1 on linux_amd64
```

Section 2: AWS Configuration and STS

2.1 Configure AWS Credentials

Set Up AWS Profile

```
bash

# Configure AWS CLI with your admin credentials
aws configure
```

Configuration Example:

```
AWS Access Key ID [****************XXXX]: YOUR_ACCESS_KEY_ID
AWS Secret Access Key [****************XXXX]: YOUR_SECRET_ACCESS_KEY
Default region name [il-central-1]: us-east-1
Default output format [None]: json
```

Verify Current Identity

```
bash

# Check current AWS identity
aws sts get-caller-identity
```

Expected Output:

```json
{
  "UserId": "AIDAT4ID72Q5IFIBCJ6XS",
  "Account": "266833220666",
  "Arn": "arn:aws:iam::266833220666:user/your-admin-user"
}
```

Section 3: Lab Files Overview

3.1 Project Structure

```
terraform-sts/
├── main.tf
├── providers.tf
├── role.yaml
└── .terraform.lock.hcl (generated)
```

3.2 Configuration Files

File: `main.tf`

```hcl
```

```hcl
# Data source to get current caller identity
data "aws_caller_identity" "current" {}

# Output current user information
output "user_info" {
  value = data.aws_caller_identity.current
}

# S3 bucket with unique naming using random ID
resource "aws_s3_bucket" "example_bucket" {
  bucket = "example-bucket-${random_id.s3_id.dec}"
  tags = {
    Environment = "dev"
    Project     = "TerraformSTS"
  }
}

# Random ID generator for unique bucket naming
resource "random_id" "s3_id" {
  byte_length = 2
}

# Terraform configuration block
terraform {
  required_providers {
    aws = {
      source  = "hashicorp/aws"
      version = "~>4.0"
    }
  }
}
```

File: providers.tf

```hcl
```

```
terraform {
  required_providers {
    aws = {
      source  = "hashicorp/aws"
      version = "~>4.0"
    }
  }
}

provider "aws" {
  region = "us-east-1"
  assume_role {
    role_arn     = "<role arn here>"  # To be updated
    session_name = "terraform-session"
  }
}
```

File: `role.yaml` (CloudFormation Template)

```yaml
```

```yaml
AWSTemplateFormatVersion: "2010-09-09"
Description: CloudFormation template to create a role with superuser permissions assigned to a specific user and allo

Resources:
 SuperuserRole:
  Type: AWS::IAM::Role
  Properties:
   AssumeRolePolicyDocument:
    Version: "2012-10-17"
    Statement:
     - Effect: Allow
       Principal:
        AWS: !GetAtt NewUser.Arn
       Action:
        - sts:AssumeRole
   Policies:
    - PolicyName: VideoUserAccess
      PolicyDocument:
       Version: "2012-10-17"
       Statement:
        - Effect: Allow
          Action: "*"
          Resource: "*"

 NewUser:
  Type: AWS::IAM::User
  Properties:
   UserName: VideoUser
   Policies:
    - PolicyName: AssumeAnyRolePolicy
      PolicyDocument:
       Version: "2012-10-17"
       Statement:
        - Effect: Allow
          Action:
           - sts:AssumeRole
          Resource: "*"

Outputs:
 SuperuserRoleArn:
  Description: The ARN of the Videouser Role
  Value: !GetAtt SuperuserRole.Arn
```

```
NewUserName:
  Description: The name of the new IAM user
  Value: !Ref NewUser
```

Section 4: CloudFormation Stack Deployment

4.1 Deploy IAM Resources

Create CloudFormation Stack

```bash
aws cloudformation create-stack \
  --stack-name TerraformAuthStack \
  --template-body file://role.yaml \
  --capabilities CAPABILITY_NAMED_IAM
```

Expected Output:

```json
{
    "StackId": "arn:aws:cloudformation:us-east-1:266833220666:stack/TerraformAuthStack/bc107fe0-84cf-11f0-baa8-0
}
```

4.2 Verify Resource Creation

AWS Console Verification

1. **IAM Users**: Navigate to IAM → Users → Find `VideoUser`
2. **IAM Roles**: Navigate to IAM → Roles → Find `TerraformAuthStack-SuperuserRole-*`
3. **CloudFormation**: Navigate to CloudFormation → Stacks → Find `TerraformAuthStack`
4. **Stack Outputs**: Click on stack → Outputs tab → View `NewUserName` and `SuperuserRoleArn`

Section 5: User Access Key Management

5.1 Create Access Keys for VideoUser

Generate Access Keys

```bash
aws iam create-access-key --user-name VideoUser
```

Expected Output:

```json
{
  "AccessKey": {
    "UserName": "VideoUser",
    "AccessKeyId": "AKIAT4ID72Q5DNDQ23QJ",
    "Status": "Active",
    "SecretAccessKey": "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",
    "CreateDate": "2025-08-29T12:15:22+00:00"
  }
}
```

> **Security Note**: These are temporary credentials for lab purposes. In production, use IAM roles and temporary credentials whenever possible.

5.2 Configure VideoUser Credentials

Update AWS Configuration

```bash
aws configure
```

Configuration with VideoUser:

```
AWS Access Key ID [**********]: AKIAT4ID72Q5DNDQ23QJ
AWS Secret Access Key [**********]: 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
Default region name [us-east-1]: us-east-1
Default output format [json]: json
```

Verify VideoUser Identity

```bash
aws sts get-caller-identity
```

Expected Output:

```json
{
    "UserId": "AIDAT4ID72Q5B2QD3754E",
    "Account": "266833220666",
    "Arn": "arn:aws:iam::266833220666:user/VideoUser"
}
```

Section 6: Terraform Configuration with Role Assumption

6.1 Update Provider Configuration

Modify `providers.tf`

```hcl
terraform {
  required_providers {
    aws = {
      source  = "hashicorp/aws"
      version = "~>4.0"
    }
  }
}

provider "aws" {
  region = "us-east-1"
  assume_role {
    role_arn     = "arn:aws:iam::266833220666:role/TerraformAuthStack-SuperuserRole-CU58d6ffCH8F"
    session_name = "terraform-session"
  }
}
```

> **Note**: Replace the `role_arn` value with the actual ARN from your CloudFormation stack outputs.

Section 7: Terraform Deployment

7.1 Initialize Terraform

Run Terraform Init

```
bash

terraform init
```

Expected Output:

```
Initializing the backend...
Initializing provider plugins...
- Finding hashicorp/aws versions matching "~> 4.0"...
- Finding latest version of hashicorp/random...
- Installing hashicorp/aws v4.67.0...
- Installed hashicorp/aws v4.67.0 (signed by HashiCorp)
- Installing hashicorp/random v3.7.2...
- Installed hashicorp/random v3.7.2 (signed by HashiCorp)

Terraform has been successfully initialized!
```

7.2 Plan Infrastructure Changes

Run Terraform Plan

```
bash

terraform plan
```

Key Output Sections:

```
data.aws_caller_identity.current: Reading...
data.aws_caller_identity.current: Read complete after 1s [id=266833220666]

Terraform will perform the following actions:

  # aws_s3_bucket.example_bucket will be created
  + resource "aws_s3_bucket" "example_bucket" {
      + bucket = (known after apply)
      + tags = {
          + "Environment" = "dev"
          + "Project"     = "TerraformSTS"
        }
    }

  # random_id.s3_id will be created
  + resource "random_id" "s3_id" {
      + byte_length = 2
      + dec         = (known after apply)
    }

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

Changes to Outputs:
  + user_info = {
      + account_id = "266833220666"
      + arn        = "arn:aws:sts::266833220666:assumed-role/TerraformAuthStack-SuperuserRole-
CU58d6ffCH8F/terraform-session"
      + id         = "266833220666"
      + user_id    = "AROAT4ID72Q5IJVPRQER4:terraform-session"
    }
```

7.3 Apply Infrastructure Changes

Deploy Resources

```
bash

terraform apply
```

Expected Output:

```
random_id.s3_id: Creating...
random_id.s3_id: Creation complete after 0s [id=4Mk]
aws_s3_bucket.example_bucket: Creating...
aws_s3_bucket.example_bucket: Creation complete after 5s [id=example-bucket-57545]

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

Outputs:

user_info = {
  "account_id" = "266833220666"
  "arn" = "arn:aws:sts::266833220666:assumed-role/TerraformAuthStack-SuperuserRole-CU58d6ffCH8F/terraform-session"
  "id" = "266833220666"
  "user_id" = "AROAT4ID72Q5IJVPRQER4:terraform-session"
}
```

Verify S3 Bucket Creation

1. Navigate to AWS S3 console
2. Find bucket: `example-bucket-57545` (number will vary)
3. Verify tags: Environment=dev, Project=TerraformSTS

Section 8: Understanding the Output

8.1 STS Information Analysis

Key Output Elements

- **account_id**: AWS account number
- **arn**: Shows assumed role session ARN
- **user_id**: Temporary role session identifier

Role Assumption Verification

The ARN shows: `arn:aws:sts::266833220666:assumed-role/TerraformAuthStack-SuperuserRole-CU58d6ffCH8F/terraform-session`

This confirms:

- VideoUser successfully assumed the SuperuserRole

- Session name is "terraform-session"
- Operations are performed with role permissions, not user permissions

---

Section 9: Cleanup Process

9.1 Destroy Terraform Resources

Remove Created Infrastructure

```
bash

terraform destroy
```

Expected Output:

```
aws_s3_bucket.example_bucket: Destroying... [id=example-bucket-57545]
aws_s3_bucket.example_bucket: Destruction complete after 1s
random_id.s3_id: Destroying... [id=4Mk]
random_id.s3_id: Destruction complete after 0s

Destroy complete! Resources: 2 destroyed.
```

9.2 Restore Admin Credentials

Reconfigure AWS CLI

```
bash

aws configure
```

Restore Admin Credentials:

```
AWS Access Key ID [****************XXXX]: YOUR_ADMIN_ACCESS_KEY
AWS Secret Access Key [****************XXXX]: YOUR_ADMIN_SECRET_KEY
Default region name [us-east-1]: us-east-1
Default output format [json]: json
```

9.3 Delete CloudFormation Stack

Remove IAM Resources

```bash
aws cloudformation delete-stack --stack-name TerraformAuthStack
```

Verify Deletion

1. Navigate to AWS CloudFormation console
2. Check stack status: `DELETE_IN_PROGRESS`
3. Wait for complete deletion

Section 10: Key Concepts and Best Practices

10.1 AWS STS Concepts

Security Token Service (STS)

- **Purpose**: Provides temporary, limited-privilege credentials
- **Benefits**: Enhanced security through credential rotation
- **Use Cases**: Cross-account access, role assumption, federation

Role Assumption Process

1. **Principal**: Entity that wants to assume the role (VideoUser)
2. **Role**: Target role with specific permissions (SuperuserRole)
3. **Trust Policy**: Defines who can assume the role
4. **Permissions**: What actions the assumed role can perform

10.2 Terraform Integration

Provider Configuration

```hcl
```

```
provider "aws" {
  assume_role {
    role_arn     = "arn:aws:iam::ACCOUNT:role/ROLE_NAME"
    session_name = "terraform-session"
  }
}
```

Benefits of Role Assumption

- **Security**: No need to store long-term credentials
- **Auditing**: Clear audit trail of who performed actions
- **Permissions**: Granular control over what Terraform can do

10.3 Security Best Practices

Credential Management

- Use IAM roles instead of user access keys when possible
- Rotate access keys regularly
- Apply principle of least privilege
- Monitor role assumption activities

CloudFormation Security

- Use `CAPABILITY_NAMED_IAM` for IAM resource creation
- Review trust policies carefully
- Implement resource tagging for tracking

Section 11: Troubleshooting Guide

11.1 Common Issues

Issue: "AccessDenied" during role assumption

Cause: Trust policy doesn't allow the user to assume the role **Solution**: Verify the trust policy in the CloudFormation template

Issue: Terraform plan fails with authentication error

Cause: Incorrect role ARN or VideoUser lacks assume role permissions **Solution**:

1. Check CloudFormation outputs for correct role ARN
2. Verify VideoUser has assume role policy

Issue: S3 bucket creation fails

Cause: Bucket name already exists globally **Solution**: Random ID should prevent this, but check the generated name

Issue: CloudFormation stack creation fails

Cause: Missing capabilities or IAM permissions **Solution**: Ensure `--capabilities CAPABILITY_NAMED_IAM` is included

11.2 Verification Commands

Check Current Identity

```bash
aws sts get-caller-identity
```

List CloudFormation Stacks

```bash
aws cloudformation list-stacks --stack-status-filter CREATE_COMPLETE
```

Verify Role Existence

```bash
aws iam get-role --role-name TerraformAuthStack-SuperuserRole-XXXXXXXX
```

Section 12: Learning Extensions

12.1 Advanced Scenarios

Multiple Role Assumption

- Create additional roles with different permissions
- Practice switching between roles
- Implement role chaining scenarios

Cross-Account Access

- Set up role assumption between different AWS accounts
- Configure external ID for enhanced security
- Practice federated access patterns

Session Duration Control

```hcl
provider "aws" {
  assume_role {
    role_arn     = "arn:aws:iam::ACCOUNT:role/ROLE_NAME"
    session_name = "terraform-session"
    duration_seconds = 3600  # 1 hour
  }
}
```

12.2 Production Considerations

Environment Separation

- Use different roles for different environments
- Implement environment-specific permissions
- Practice blue-green deployment strategies

Automation Integration

- Integrate with CI/CD pipelines
- Use AWS credentials in automated workflows
- Implement secrets management solutions

Section 13: Lab Completion Checklist

13.1 Setup Verification

- [ ] AWS CLI installed and configured
- [ ] Terraform CLI installed and verified
- [ ] Admin credentials configured and tested

13.2 Core Lab Activities

- [ ] CloudFormation stack deployed successfully
- [ ] VideoUser created with appropriate permissions
- [ ] VideoUser access keys generated and configured
- [ ] Terraform provider configured for role assumption
- [ ] Infrastructure deployed using assumed role
- [ ] S3 bucket created with correct naming and tags
- [ ] Role assumption verified through outputs

13.3 Cleanup Verification

- [ ] Terraform resources destroyed
- [ ] Admin credentials restored
- [ ] CloudFormation stack deleted
- [ ] No orphaned resources remaining

13.4 Concept Understanding

- [ ] STS role assumption process understood
- [ ] Difference between user and role permissions clear
- [ ] Security benefits of temporary credentials appreciated
- [ ] Terraform integration with AWS STS demonstrated

Conclusion

This lab demonstrates advanced AWS security patterns using STS and role assumption. You've learned how to:

- Create IAM users and roles programmatically
- Configure Terraform to assume roles for enhanced security
- Deploy infrastructure using temporary credentials
- Implement proper cleanup procedures

These patterns are essential for secure AWS automation and are widely used in enterprise environments for maintaining security boundaries while enabling infrastructure automation.

Additional Resources

- AWS STS Documentation

- Terraform AWS Provider - Assume Role
- AWS IAM Best Practices
- CloudFormation IAM Template Reference