Python(Boto3) for AWS

AWSome scripts **Python**ic way

Sanjeev Jaiswal (Jassi)

Who IAM

Sanjeev Jaiswal

 @jassics

 /in/jassics

 jassics@gmail.com

12+ years of Experience in

- Application Security
- AWS Security
- Threat Modeling
- Secure Code Review
- OWASP Top 10
- Automation using Python3

What we will cover

Fundamentals

- Introduction to AWS CLI
- AWS CLI setup
- AWS CLI Examples
- Introduction to Boto3
- Boto3 setup and verification
- Hands-on using Boto3

Hands-On

- AWS resource inventory
- List public S3 buckets
- List IAM details
- Security group exposed to public
- Get ELB Public IP for post scan
- Orphan Security Group

What you know in Python

Minimal concept to get going

- Data Types
- Control statements
- Function
- Know list, tuples and dictionary
- Basic Troubleshooting
- How to install module using pip
- How to run python scripts

Test your Python skills

1. Get employee details as user input: emp id, name, joined date, skills in list, projects in dictionary
2. Create a dictionary in below format and fill with above input values:

```
{
        'Employee ID':
        {       Name: 'string'
                Joined: 'yyyy-mm-dd'
                 Skills: [‘list’, ‘of’, ‘skills’]
                Project: {‘project_name’: ‘project description’}
        }
}
```

3. Print the employee dictionary

What you know in AWS

Minimal AWS knowledge is enough

- Have AWS Console access
- IAM features
- EC2 related operations
- How to work with S3 buckets
- Aware of ELB
- Used Security group before

Basic AWS Operations

1. IAM: create few users with aws-cli access and add in different groups, roles
2. EC2: create linux instances (min. 2) with different security groups, tags, name
3. S3: create 2 buckets, upload objects to them, make 1 bucket public and 1 private
4. S3: Encryption, make public bucket, static website
5. Create separate security groups for web, mysql, mongo, ssh
6. Create few load balancers and attach some instances to it
7. Able to verify AWS config non-compliant issues
8. CloudTrail and CloudWatch operations
9. Hands-on to VPC and subnets settings
10. AWS Shared security responsibilities for writing security scripts

Let's Learn AWS CLI

AWS CLI

- AWS Command Line Interface (CLI) is a unified tool to manage your AWS services. It can run in MacOS, Windows, Linux as well.
- *Available in 2 versions:*
  1. 1.x : for backward compatibility
  2. 2.x: recent available release for production
- AWS CLI Command reference: https://docs.aws.amazon.com/cli/latest/reference/
- Output format: text, yaml, json, table
- Security in the AWS CLI: https://docs.aws.amazon.com/cli/latest/userguide/security.html

AWS CLI setup

You have Python 3.x installed and access to shell/terminal

Install AWS CLI:

- Using pip: *$ python -m pip install --user awscli*
- Windows: https://awscli.amazonaws.com/AWSCLIV2.msi
- Linux: https://docs.aws.amazon.com/cli/latest/userguide/install-cliv2-linux.html
- MacOS: https://docs.aws.amazon.com/cli/latest/userguide/install-cliv2-mac.html

```
$ aws --version
$ aws configure
AWS Access Key ID [None]: AKIAIOSFODNN7EXAMPLEID
AWS Secret Access Key [None]: wJalrXUtFI/K7MDENG/bPxRYEXAMPLESECRETKEY
Default region name [None]: us-east-1
Default output format [None]:
```

Let's understand aws-cli structure

```
aws [options] <command> <subcommand> [parameters]
```

- `options:`
  - `--output`
  - `--region`
  - `--profile`
  - `--version`
- `parameters:`
  - `--filter`
  - `--max-items`
  - `--page-size`

- Command (top level aws services):
  - S3
  - EC2
  - IAM
  - Cloudwatch
  - DynamoDB
  - ELB
- Subcommand (commands work with top level command):
  - S3: cp, rb, mb, sync
  - EC2: describe-instances, create-tags, create-vpc
  - IAM: create-group, get-policy, list-users, update-user
  - Cloudwatch: get-dashboard, get-metric-data
  - DynamoDB: get-item, update-table
  - ELB: create-load-balancer, remove-tags

Let’s get help from AWS CLI

- aws help
- aws command help
  - aws s3 help
  - aws ec2 help
  - aws iam help
- aws command sub-command help
  - aws s3 mb help
  - aws iam get-policy help
  - aws ec2 describe-instances help

AWS CLI - Sample code

- `$ aws iam list-users --output json`
- `$ aws iam list-users --output text --query 'Users[*].[UserName,Arn,CreateDate,PasswordLastUsed,UserId]'`
- `$ aws iam list-groups-for-user --user-name sanjeev  --output text --query "Groups[].[GroupName]"`
- `$ aws ec2 describe-instances --query 'Reservations[*].Instances[*].[Placement.AvailabilityZone, State.Name, InstanceId]' --output text`
- `$ aws ec2 describe-volumes --query 'Volumes[*].{ID:VolumeId,AZ:AvailabilityZone,Size:Size}' --output table`
- `$ aws ec2 describe-instances --filters Name=instance-type,Values=t2.micro,t3.micro Name=availability-zone,Values=us-east-2c`
- `$ aws s3 website s3://bucket-name/ --index-document index.html --error-document error.html`
- `$ aws s3api get-bucket-website --bucket mysite.in`
- `$ aws logs get-log-events --log-group-name my-logs --log-stream-name 20200311`

You will love aws-shell

- Interactive shell for aws cli commands
- `pip install aws-shell`
- $aws-shell
- No need to remember sub-commands and parameters anymore
- Autocompletion of the commands are just awesome
- Autocompletes commands, shorthand syntax and server side
- Inline documents helps to understand the command easily
- You should try dot commands: .exit, .profile, .cd, .quit, .edit
- Fish-style autosuggestion makes your life easier
- F10 or Ctrl-D (Mac) to exit

Let's use Boto3

Boto3 Setup

- Install boto3: *pip3 install boto3* or *python3 -m pip install boto3*
- Include this line in your python code: *import boto3*
- Create an object using boto3 client or resource library
  - client = boto3.client(‘resource-name’) Ex: iam_client = boto3.client(‘iam’)
  - resource = boto3.resource(‘resource-name’) Ex: s3_resource = boto3.resource(‘s3’)
- It will take your Credentials which you set while running *`aws configure`*
- Save the script with .py extension and run it
- If it didn’t give any error means boto3 is working perfectly

Client vs resource

Client:

- low-level AWS service access
- generated from AWS **service** description
- exposes botocore client to the developer
- typically maps 1:1 with the AWS service API
- all AWS service operations are supported by clients

Resource:

- higher-level, object-oriented API
- generated from **resource** description
- uses identifiers and attributes
- has actions (operations on resources)
- exposes subresources and collections of AWS resources
- does not provide 100% API coverage of AWS services

IAM accounts details

1. Get IAM user details like username, group, policy
   a. `client.get_account_authorization_details()`
2. IAM user management like:
   a. add user: `client.create_user(UserName='name')`
   b. edit user: `client.update_user(UserName='name', NewUserName='newName')` and
   c. delete user: `client.delete_user(UserName='name')`
3. Get policy details:
   a. List attached user policy: `client.list_attached_user_policies(UserName='name')`
   b. Get Policy details: `client.get_policy(PolicyArn='policyarn')`

S3 Bucket management

1. Create a bucket
    a. s3client = boto3.client('s3')
    b. s3client.create_bucket(Bucket=bucket_name)
2. List all the buckets: list_buckets = s3client.list_buckets()
3. Search if your bucket is there
4. Upload a file to your bucket
5. Generate a presigned url of newly uploaded file to access online
6. Print file(object)details using s3 resource API
    a. s3resource = boto3.resource('s3')
    b. bucket = s3resource.Bucket(bucket_name)
    c. obj = bucket.Object(object_key)
7. Delete the bucket and any files(objects) inside it
    a. bucket = s3Resource.Bucket(bucket_name)
    b. delete_responses = bucket.objects.delete()

Get Running EC2 details region-wise

1. Get all ec2 regions in a list
2. Loop through the region list and
3. Get ec2 region wise and print below items for each ec2 instance:
    a. Instance id
    b. Instance type
    c. Image id
    d. Public ip
    e. Private ip
    f. Availability zone

Below code would help you get going:

```
import boto3
ec2client = boto3.client('ec2')
instances = ec2client.describe_instances()
```

Get Orphan Security Groups

1. Get security group of all the regions
2. Get Security group attached to different instances
3. Compare with existing security groups
4. If security group is not attached with any of the below instances, add in orphan list
    a. ec2
    b. rds
    c. vpc
    d. elb
    e. elbv2
5. Print orphan list and other useful info related to this scan.

List Security Group attached with EC2

1. Get all the regions associated with ec2
2. Loop through the regions
3. Find attached security group
4. Print below items:
    a. Instance id
    b. Security group name
    c. Security Group Id
    d. VpcId
    e. SubnetId
    f. Ec2 running status (State -> running)

Fetch Public IPs of ELBs

1. Connect to client ec2
2. Get all regions and loop through
3. Connect to elb and elbv2
4. Use describe_load_balancers()
5. Get public IP

Below code would get you going

```
for region in regions:
    profile   = boto3.session.Session(profile_name=env_type, region_name=region)
    elbList   = profile.client('elb')
    applbList = profile.client('elbv2')

    bals    = elbList.describe_load_balancers()
    appbals = applbList.describe_load_balancers()
```

What's Next

- AWS Security Automation: https://github.com/awslabs/aws-security-automation
- SANS SEC573: https://www.sans.org/course/automating-information-security-with-python
- Automate event-driven security stuffs using AWS Lambda in Python
- Automate AWS Services security assessment using Python
- Automate AWS CIS benchmarks
- Automate some AWS Exploits
- Automate/Solve AWS Based CTF challenges
- Use Pacu, Prowler, ScoutSuite for AWS Exploitation and Security Assessment
- Make command line tool using click module

Resources

Automate the boring stuff with Python

Hands-On enterprise Automation with Python

AWS CLI by Amazon

Boto3 Documentation

Credits

Image Credit

- Noun Project for icons

Content Credits

- Aws-labs
- Boto3 Doc

Thank You!