# Readme

This python script was developed to pull the user’s public IP from the ident.me and checkip.amazonaws.com sites and update their security group rules accordingly. The intent is to automate the update of these rules to allow the support engineer to connect to their test databases without needing to manually adjust the security group rules every time they reconnect to the VPN.

Since the VPN adds an additional networking layer, the IP addresses are pulled from multiple sources to capture all public Ips for the user’s computer. When only one of these IPs is updated, some resources may refuse connections.

# How it Works

The script scrapes your public IPs from the ident.me and checkip.amazonaws.com websites using the *urllib* library. These IPs are stored in a list and used for updating the tagged resources.

The script utilizes the *boto3* library to create an EC2 client. This client is used to find and describe security groups that meet the tagging criteria. This information includes the current configuration for the ingress rules, including the type, protocol, ports, source IP, and description.

This information is then parsed down to the group’s ingress rules which are passed to a function that utilizes the information to delete the old rules and replace them with new rules. In this process, only the source IP changes, all other configuration settings remain the same.

If no ingress rules exist for the security group, the script creates new rules and defaults to allow all access to the user’s public IPs.

A new rule for each retrieved IP is created for every unique ingress rule that has a description value of ‘Automatic Update’. As such, if a rule with the proper description exists that allows ingress on port 5432 from IP 172.168.10.10/32, the script will create as many new ingress rules as needed to replicate this ingress configuration for the retrieved IPs.

This only occurs on the initial run of the script or if rules are manually deleted. Once the script is being run on a schedule, it will not add more rules than already exist. An example case can be seen in the examples section below.

Upon successful completion, the script will have updated security groups tagged with ‘auto-update: true’ by removing ingress rules with the description ‘Automatic Update’ and replacing them with new rules of the same configuration with updated source IPs.

# Prerequisites

The following must be downloaded to the user’s computer and configured:

* AWS CLI
  + Must be configured for the user’s account and the region that the security groups are in.
  + <https://docs.aws.amazon.com/cli/latest/userguide/getting-started-install.html>
* Python (version 3)
  + <https://www.python.org/downloads/>
* Pip
  + Included by default in Python version 3.4 and above

The following Python packages must be downloaded and available:

* *boto3*
  + <https://boto3.amazonaws.com/v1/documentation/api/latest/guide/quickstart.html>
* *logging[[1]](#footnote-1)*
* *urllib\**
* *os\**

The AWS Security Groups you wish to automatically update must be configured with the following tags:

* auto-update: true

The ingress rules for the tagged security groups must:

* Be configured to allow the access you need
  + The type, protocol, and ports (if applicable) will persist across updates, only the source IP will change
* Have their description set to ‘Automatic Update’. Only rules with that description will be updated.

# Setup

For MacOS, the setup instructions are as follows:

1. After unzipping the folder, move the updateSG.py file to the location you intend to store it, such as the home folder. Make note of the file path as you will need this later
2. Find and make note of the file path that leads to your python3 directory
   1. The default install path for this directory is */usr/bin/python3*
3. Generate a cron schedule expression the defines when you want to run the script
   1. *15 9 \* \* MON-FRI* will schedule the script to run at 9:15 AM (local time zone) Monday through Friday
   2. You can use [Crontab Guru](https://crontab.guru/) to generate other crontab expressions
4. In the MacOS terminal, create a new crontab file using the command *crontab -e*
5. Press ‘i’ to begin inserting text into the crontab file using the vim editor
6. Input your crontab expression, followed by the path to your python3 directory, followed by the path to the updateSG.py file.
   1. Example: 15 9 \* \* MON-FRI /usr/bin/python3 /Users/username/updateSG.py
7. Leave insert mode by pressing escape, then save and exit the editor using *:wq*

For Windows users, the Python script can be scheduled using the following steps:

1. After unzipping the folder, move the updateSG.py file to the location you intend to store it, such as the home folder. Make note of the file path as you will need this later
2. Find and make note of the file path that leads to your python.exe file
   1. An example path for windows would be *C:\Users\username\AppData\Local\Programs\Python\Python39\python.exe*
3. Create a new file in notepad and enter the path to the python.exe file in quotation marks followed by the path to the updateSG.py file in quotation marks followed by the word ‘pause’ on a new line.
4. Save the file with the .bat extension to your desired file location
5. Open the Windows Control Panel and then click on the “Administrative Tools”
6. Double-click on the “Task Scheduler”, and then choose the option to *‘Create Basic Task…’*
7. Type a name for your task (you can also type a description if needed), and then press next
8. Choose to start the task “Daily” since we wish to run the Python script daily. Then specify the start date and time for just after your scheduled shift start time.
9. Select “Start a Program” and then press next
10. Use the browse button to find the batch file that runs the Python script.
11. Finally, click on finish to complete the scheduling.

# Common Errors

The most common errors for this script are ones that are caused by missing or incorrectly installed python3 packages. When installing these packages, be sure to use the *python3 -m pip install* command, as this will install the packages in your python3 directory.

If further errors arise while running the script, you can download the IDLE Python IDE and use that to get more information on what errors are occurring. Running the script using IDLE will provide standard error output which can assist you in troubleshooting.

# Examples

Suppose there exists a security group with four ingress rules configured like the following:

Graphical user interface, application

Description automatically generated

After a successful run of the Python script, it can be expected that the original Oracle-RDS and PostgreSQL rules to be updated with correct IP values and for there to be two new rules, an additional Oracle-RDS and PostgreSQL rule for the second IP address, bringing the total number to six ingress rules.

The updated rules are shown below:

Graphical user interface, application

Description automatically generated

The rules that allow access for the application server and for HTTP traffic remain unchanged, but the additional PostgreSQL and Oracle-RDS rules have been added to provide access for both public IPs.

In the case where these rules are then modified, and the source IP address is changed to match the following:

Graphical user interface

Description automatically generated

When the script is run again, each rule marked for automatic update will not multiply again by two (which would bring the total to fifteen ingress rules, six originals and 8 new rules) as they did originally when only one unique rule existed for the automatically updated ingress rules. Instead, no additional rules will be created, and the source IP values will be changed according to the IPs retrieved by the script.

After the third run of the script, the rules will appear as follows:

A screenshot of a computer

Description automatically generated with medium confidence

As you can see, the rules marked for automatic update are changed to match the retrieved IP values, no additional rules are created, and the existing rules not marked for update remain unchanged, leaving the total ingress rules at six.

Finally, if I add a unique rule, like the following:

Graphical user interface, application

Description automatically generated

A fourth run of the script will create an additional rule to allow access to the resource from both public IPs, same as it did in the original run, bringing the total number of ingress rules to eight. This can be seen below:

Graphical user interface, application

Description automatically generated

# Changelog

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **User** | **Version** | **Updates** |
| 2/16/2023 | rappdb | 1.0.0 | Initial creation |
| 2/16/2023 | rappdb | 1.0.1 | Added picture guides and Windows set up guide |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

1. Denotes a default package. These packages should not require additional configuration once Python has been downloaded but are required for the script to successfully execute. [↑](#footnote-ref-1)